Oracle® Database

Oracle AI Vector Search User's Guide

23ai
F87786-21
April 2025

Oracle Database Oracle AI Vector Search User's Guide, 23ai

F87786-21

Primary Authors: Sarah Hirschfeld, Subha Vikram, Jean-Francois Verrier

Contributing Authors: Douglas Williams, Frederick Kush, Gunjan Jain, Jessica True, Jody Glover, Maitreyee Chaliha, Mamata Basapur, Prakash Jashnani, Ramya P, Sarika Surampudi, Spurthi SV, Suresh Rajan, Tulika Das, Usha Krishnamurthy

Contributors: Agnivo Saha, Ajay Sunnyhith Chidurala, Aleksandra Czarlinska, Angela Amor, Aurosish Mishra, Bonnie Xia, Boriana Milenova, David Jiang, Dinesh Das, Doug Hood, George Krupka, Harichandan Roy, Malavika S P, Mark Hornick, Rohan Aggarwal, Roger Ford, Sean Stacey, Sebastian DeLaHoz, Shasank Chavan, Sudhir Kumar, Tirthankar Lahiri, Teck Hua Lee, Valentin Leonard Tabacaru Cristache, Vinita Subramanian, Weiwei Gong, Yuan Zhou

Contents

3 Get Started

4 Generate Vector Embeddings

5 Store Vector Embeddings

6 Create Vector Indexes and Hybrid Vector Indexes

7 Use SQL Functions for Vector Operations

8 Query Data With Similarity and Hybrid Searches

Preface

Oracle Database AI Vector Search User's Guide provides information about querying semantic and business data with Oracle AI Vector Search.

Audience

This guide is intended for application developers, database administrators, data users, and others who perform the following tasks:

- Implement artificial intelligence (AI) solutions for websites and unstructured or structured data
- Build query applications by using natural language processing and machine learning techniques
- Perform similarity searches on content, such as words, documents, audio tracks, or images

To use this document, you must have a basic familiarity with vector embedding and machine learning concepts, SQL, SQL*Plus, and PL/SQL.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customer access to and use of Oracle support services will be pursuant to the terms and conditions specified in their Oracle order for the applicable services.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

What's New for Oracle AI Vector Search

This chapter lists the following changes in *Oracle Database AI Vector Search User's Guide* for Oracle Database 23ai:

- Oracle Database 23ai Release Updates
  The following sections include new AI Vector Search features introduced in Oracle Database 23ai as part of the listed Release Update.
- Autonomous Database Updates
  The following sections include new AI Vector Search features introduced in Oracle Autonomous Database as part of the listed update.
- Deprecated Features
  The following features are deprecated, and may be desupported in a future release.

Oracle Database 23ai Release Updates

The following sections include new AI Vector Search features introduced in Oracle Database 23ai as part of the listed Release Update.

- July 2024, Release Update 23.5
  Included are some notable Oracle AI Vector Search updates with Oracle Database 23ai, Release Update 23.5.
- October 2024, Release Update 23.6
  Included are some notable Oracle AI Vector Search updates with Oracle Database 23ai, Release Update 23.6.
- January 2025, Release Update 23.7
  Included are some notable Oracle AI Vector Search updates with Oracle Database 23ai, Release Update 23.7.
- April 2025, Release Update 23.8
  Included are some notable Oracle AI Vector Search updates with Oracle Database 23ai, Release Update 23.8.

July 2024, Release Update 23.5

Included are some notable Oracle AI Vector Search updates with Oracle Database 23ai, Release Update 23.5.

Feature	Description
Binary Vectors	You can use the `BINARY` vector dimension format to declare vectors and vector columns. Create Tables Using the VECTOR Data Type
LangChain Integration	Oracle AI Vector Search's integration with LangChain allows you to use LangChain's powerful open source orchestration framework seamlessly with vector search capabilities. Oracle AI Vector Search Integration with LangChain

Feature	Description
Optimizer Plans for Vector Indexes	More information has been added about optimizer plans for HNSW and IVF indexes along with vector index hints. • Optimizer Plans for HNSW Vector Indexes • Optimizer Plans for IVF Vector Indexes • Vector Index Hints
Documentation Map to GenAI Prompts	Use the documentation map to find prompts that will help you get tailored answers from your preferred GenAI Chatbot to your Oracle AI Vector Search questions. Your Vector Documentation Map to GenAI Prompts

October 2024, Release Update 23.6

Included are some notable Oracle AI Vector Search updates with Oracle Database 23ai, Release Update 23.6.

> **Note:**
>
> Certain features require the `COMPATIBLE` initialization parameter to be manually updated in order to be available for use. For information about the `COMPATIBLE` parameter and how to change it, see *Oracle Database Upgrade Guide*.
>
> If you are using RAC instances, certain features must be enabled using Oracle RAC two-stage rolling updates to ensure that any patches are enabled on all nodes. For more information, see *Oracle Real Application Clusters Administration and Deployment Guide*.

Feature	Description
Hybrid Search using Hybrid Vector Indexes	You can use a hybrid vector index to index and query data using a combination of full-text search and vector similarity search. Hybrid searches can enhance the relevance (quality) of your search results by integrating the keyword matching capabilities of Oracle Text indexes with the semantic precision of vector indexes. An end-to-end indexing pipeline facilitates the indexing of documents without requiring you to be an expert in various text processing, chunking, or embedding strategies. As part of this feature, new preference helper procedures have been added to the existing `DBMS_VECTOR` and `DBMS_VECTOR_CHAIN` PL/SQL packages to manage vector index-related settings. A new PL/SQL package, `DBMS_HYBRID_VECTOR`, has also been added. The package contains a `SEARCH` API that allows you to query against hybrid vector indexes in multiple ways. A new dictionary view `<index name>$VECTORS` provides information on row identifiers, chunks, and embeddings for all indexed documents. • Manage Hybrid Vector Indexes • Perform Hybrid Search • DBMS_HYBRID_VECTOR • CREATE_PREFERENCE • DROP_PREFERENCE • <index name>$VECTORS

Feature	Description
Updated AI Vector Search Workflow	The AI Vector Search Workflow included in this guide has been updated to include support for hybrid vector indexes and hybrid search. Oracle AI Vector Search Workflow
Updated Documentation Map to GenAI Prompts	The documentation map to GenAI prompts has been updated to include the new features available in Release Update 23.6. Your Vector Documentation Map to GenAI Prompts
Extended SQL Quick Start	The *SQL Quick Start Using a Vector Embedding Model Uploaded into the Database* has been extended to include steps for creating a hybrid vector index and running a hybrid search. SQL Quick Start Using a Vector Embedding Model Uploaded into the Database
Support for Ollama	You can use open embedding models (such as mxbai-embed-large, nomic-embed-text, or all-minilm) and large language models (such as Llama 3, Phi 3, Mistral, or Gemma 2) using the local host REST endpoint provider, Ollama. Ollama is a free and open-source command-line interface tool that allows you to run these models locally and privately on your Linux, Windows, and macOS systems. • Convert Text String to Embedding Using the Local REST Provider Ollama • Generate Summary Using the Local REST Provider Ollama • Generate Text Using the Local REST Provider Ollama
Support for `SPARSE` Vectors	In addition to `DENSE` vectors, you have the option to create `SPARSE` vectors. Sparse vectors are vectors that typically have a large number of dimensions but only a few of those dimensions have non-zero values. Because sparse vectors only store non-zero values, their use can improve efficiency and save storage space. The `COMPATIBLE` initialization parameter must be set to 23.6.0 to use this feature. Create Tables Using the VECTOR Data Type
Integration with LlamaIndex	Oracle AI Vector Search's integration with LlamaIndex allows you to use LlamaIndex's open-source data framework with vector search capabilities. This provides a powerful foundation for building sophisticated AI applications that can leverage both structured and unstructured data within the Oracle ecosystem. Oracle AI Vector Search Integration with LlamaIndex
Jaccard Distance	`JACCARD` is available as a distance metric and `JACCARD_DISTANCE` as a shorthand for the `VECTOR_DISTANCE` function to calculate the Jaccard distance between two `BINARY` vectors. • Jaccard Similarity • JACCARD_DISTANCE
Hamming Distance	`HAMMING_DISTANCE` is available as a shorthand for the `VECTOR_DISTANCE` function to calculate the Hamming similarity between two vectors. HAMMING_DISTANCE

Feature	Description
Transaction Support for HNSW Indexes	Transactions are maintained on tables with Hierarchical Navigable Small World (HNSW) index graphs by using data structures, such as private journal and shared journal. A private journal records vectors that are added or deleted by a transaction, whereas a shared journal records all the commit system change numbers (SCNs) and corresponding modified rows. If using RAC instances, you must enable Patch 36932885 using Oracle RAC two-stage rolling updates. Understand Transaction Support for Tables with HNSW Indexes
HNSW Index Duplication and Reload	By default, a duplication mechanism is used to create HNSW indexes in an Oracle RAC environment or when an HNSW index repopulation operation is triggered. When an Oracle Database instance starts again, a reload mechanism is triggered to recreate the HNSW graph in memory as quickly as possible. HNSW full checkpoints are used to reload the graph in memory. You can use the `VECTOR_INDEX_NEIGHBOR_GRAPH_RELOAD` initialization parameter to manage duplication and reload mechanisms for HNSW indexes. If using RAC instances, you must enable Patch 36932885 using Oracle RAC two-stage rolling updates. Understand HNSW Index Population Mechanisms in Oracle RAC or Single Instance
Update to the default for `VECTOR_INDEX_NEIGHBOR_GRAPH_RELOAD`	The default value for the `VECTOR_INDEX_NEIGHBOR_GRAPH_RELOAD` initialization parameter has been changed from `OFF` to `RESTART`. With this update, HNSW duplication and reload mechanisms are enabled by default. Understand HNSW Index Population Mechanisms in Oracle RAC or Single Instance
Global and Local Partitioning for IVF Indexes	Inverted File Flat (IVF) vector indexes support both global and local indexes on partitioned tables. By default, IVF indexes are globally partitioned by centroid. You can choose to create a local IVF index, which provides a one-to-one relationship between the base table partitions or subpartitions and the index partitions. If using RAC instances, you must enable Patch 36932885 using Oracle RAC two-stage rolling updates. • Inverted File Flat Vector Indexes Partitioning Schemes • Inverted File Flat Index Syntax and Parameters
`GET_INDEX_STATUS`, `ENABLE_CHECKPOINT`, `DISABLE_CHECKPOINT`, and `INDEX_VECTOR_MEMORY_ADVISOR` Procedures	New procedures are available with the `DBMS_VECTOR` PL/SQL package: • `GET_INDEX_STATUS` to query the status of a vector index creation • `ENABLE_CHECKPOINT` and `DISABLE_CHECKPOINT` to enable or disable the Checkpoint feature for HNSW indexes • `INDEX_VECTOR_MEMORY_ADVISOR` to determine the vector memory size needed for a vector index Vector Index Status, Checkpoint, and Advisor Procedures
`VECSYS.VECTOR$INDEX$CHECKPOINTS` View	A new dictionary view `VECSYS.VECTOR$INDEX$CHECKPOINTS` provides information about HNSW full checkpoints at the database level. VECSYS.VECTOR$INDEX$CHECKPOINTS

Feature	Description
Relational Data Vectorization	You can use Oracle Machine Learning (OML) Feature Extraction algorithms with the `VECTOR_EMBEDDING()` SQL operator to vectorize sets of relational data, build similarity indexes, and perform similarity searches. These algorithms help in extracting the most informative features or columns from the data, making it easier to analyze correlations and redundancies in that data. Vectorize Relational Tables Using OML Feature Extraction Algorithms
Document Reranking	You can use the `RERANK` function, added to the existing `DBMS_VECTOR` and `DBMS_VECTOR_CHAIN` PL/SQL packages, to enhance the relevancy of your query results in Retrieval Augmented Generation (RAG) scenarios. Third-party reranking models are used to reassess and reorder an initial list of retrieved documents. • Use Reranking for Better RAG Results • DBMS_VECTOR.RERANK • DBMS_VECTOR_CHAIN.RERANK
`BLOB` Support for `UTL_TO_GENERATE_TEXT()`	In addition to the existing textual input, the `UTL_TO_GENERATE_TEXT()` PL/SQL function accepts `BLOB` as input. You can provide binary data, such as an image file, to generate a textual analysis or meaningful description of the contents of an image. Describe Images Using Public REST Providers
`BLOB` Support for `UTL_TO_EMBEDDING()`	In addition to the existing textual input, the `UTL_TO_EMBEDDING()` PL/SQL function accepts `BLOB` as input. This lets you directly generate a vector embedding from image files using REST API calls to third-party image embedding models. Convert Image to Embedding Using Public REST Providers
List of REST Endpoints	You can refer to the list of REST endpoints and corresponding REST calls supported for all third-party REST providers. Supported Third-Party Provider Operations and Endpoints

January 2025, Release Update 23.7

Included are some notable Oracle AI Vector Search updates with Oracle Database 23ai, Release Update 23.7.

> **Note:**
>
> If you are using RAC instances, certain features must be enabled using Oracle RAC two-stage rolling updates to ensure that any patches are enabled on all nodes. For more information, see *Oracle Real Application Clusters Administration and Deployment Guide*.

Feature	Description
External Table support	Columns of type `VECTOR` can be included in external tables, providing the option to store vector embeddings used for AI workloads outside of the database while still using the database as a semantic search engine. If using RAC instances, you must enable Patch 37244967 using Oracle RAC two-stage rolling updates. Vectors in External Tables

Feature	Description
Arithmetic and aggregate function support	The arithmetic operators for addition, subtraction, and multiplication along with the aggregate functions `SUM` and `AVG` can be used with vectors and columns of type `VECTOR`. Arithmetic operators are supported for use with vectors in both SQL and PL/SQL. If using RAC instances, you must enable Patch 37289468 using Oracle RAC two-stage rolling updates for support in PL/SQL. Arithmetic Operators Aggregate Functions
PL/SQL `BINARY` vector support	In addition to `FLOAT32` (the default format for comma-separated string representations of vectors), `FLOAT64`, and `INT8` dimension formats, you can also use the `BINARY` dimension format. A `BINARY` vector represents each dimension as a single bit (0 or 1). `BINARY` vectors can now be created and declared in PL/SQL, supported in the same way as they are in SQL. If using RAC instances, you must enable Patch 37289468 using Oracle RAC two-stage rolling updates. BINARY Vectors
PL/SQL `JACCARD` distance metric support	The distance metric `JACCARD` can be used in PL/SQL for similarity searches with `BINARY` vectors. In PL/SQL, there is no standalone shorthand `JACCARD_DISTANCE` function. Use `JACCARD` as the distance metric parameter of the `VECTOR_DISTANCE` function instead. If using RAC instances, you must enable Patch 37289468 using Oracle RAC two-stage rolling updates. VECTOR_DISTANCE For additional information about `VECTOR` operations supported by PL/SQL, see *Oracle Database PL/SQL Language Reference*
Globally Distributed Database support	Columns of type `VECTOR` can be included in sharded tables and duplicated tables in a distributed database, and vector indexes are supported on sharded tables in a distributed database, with some restrictions. Vectors in Distributed Database Tables Vector Indexes in a Globally Distributed Database "Restrictions" in Create Tables Using the VECTOR Data Type
In-Database Algorithms Support for `VECTOR` Data Type Predictors	The `VECTOR` data type is supported and can be used as an input to database machine learning algorithms such as classification, anomaly, regression, clustering and feature extraction. Overview of Oracle AI Vector Search More information on using `VECTOR` data type in machine learning can be found in Vector Data Type Support
Support for Image Transformer Models with AI Vector Search Using the In-Database ONNX Runtime	The Oracle Database ONNX runtime engine includes support of models for encoding images. The ONNX Pipeline models provide methods for: • Text Embedding - ONNX Pipeline Models : Text Embedding • Image Embedding - ONNX Pipeline Models : Image Embedding • Multi-modal Embedding - ONNX Pipeline Models: CLIP Multi-Modal Embedding • Text Classification - ONNX Pipeline Models: Text Classification • Reranking - ONNX Pipeline Models: Reranking Pipeline

Feature	Description
Hybrid Vector Index for JSON	New `PATHS` field is added to the `DBMS_VECTOR_CHAIN.VECTORIZER` that allows specification of an array of path objects : CREATE_PREFERENCE New syntax in the `DBMS_HYBRID_VECTOR.SEARCH` API to allow searches restricted to paths. The `VECTOR` field accepts an `INPATH` parameter which an array of valid JSON paths to restrict the search : SEARCH Support for the creation of a LOCAL index on a HVI when the underlying vector `index_type` is IVF : CREATE HYBRID VECTOR INDEX Enhancements on `ALTER INDEX` : • Support for replacing a vectorizer preference for an existing HVI index. • Support for replacing only the model and/or vector `index_type` without specifying full vectorizer preference for an existing HVI index. • Support for 'up-converting' an existing Text SEARCH or JSON SEARCH index to an HVI without a full rebuild of the textual/JSON parts of the index. ALTER INDEX Support for running optimization only on the `$VR` table for an HVI : Hybrid Vector Index Maintenance Operations
Included Columns in Neighbor Partition Vector Indexes	Included columns permit additional table (non-vector) columns to be stored in a Neighbor Partition vector index. The benefit is that query execution is optimized by removing the need to access the underlying base table to retrieve these columns. If using RAC instances, you must enable Patch 37306139 using Oracle RAC two-stage rolling updates. Included Columns

April 2025, Release Update 23.8

Included are some notable Oracle AI Vector Search updates with Oracle Database 23ai, Release Update 23.8.

> **Note:**
>
> If you are using RAC instances, certain features must be enabled using Oracle RAC two-stage rolling updates to ensure that any patches are enabled on all nodes. For more information, see *Oracle Real Application Clusters Administration and Deployment Guide*.

Feature	Description
Custom JavaScript Distance Function	In addition to the availability of built-in vector distance functions, you can now create your own custom distance function. The Multilingual Engine (MLE) is used to create a JavaScript function that defines the distance operation of your choice. The custom distance function can be used in similarity searches and in the creation of HNSW vector indexes. Custom Distance Function

Feature	Description
Automatic Vector Pool sizing with on-premises deployments	The vector memory pool can be configured to dynamically grow to meet the sizing needs for creating HNSW indexes. This is now supported for both Oracle Autonomous Databases and non-Autonomous databases. Size the Vector Pool
`SPARSE` support in PL/SQL	PL/SQL now natively supports the creation and declaration of `SPARSE` vectors. If using RAC instances, you must enable Patch 37535524 using Oracle RAC two-stage rolling updates. SPARSE Vectors *Oracle Database PL/SQL Language Reference*
New table function: `DBMS_HYBRID_VECTOR.SEARCHPIPELINE`	In addition to the existing `DBMS_HYBRID_VECTOR.SEARCH` API, the new table function `DBMS_HYBRID_VECTOR.SEARCHPIPELINE` returns a pipeline of row records. SEARCHPIPELINE
New function: `DBMS_HYBRID_VECTOR.GET_SQL`	`DBMS_HYBRID_VECTOR.GET_SQL` function displays the internal SQL query that is executed for `DBMS_HYBRID_VECTOR.SEARCH`. GET_SQL
`FILTER_BY` support in `DBMS_HYBRID_VECTOR.SEARCH`	A `FILTER_BY` field is added in the `SEARCH` API of the `DBMS_HYBRID_VECTOR` package. The `FILTER_BY` field allows you to narrow the search results using standard relational logical constraints. FILTER_BY
Terminable iteration for IVF vector indexes	Terminable iteration provides the ability to return the expected 'K' number of rows during a search using an IVF vector index. The underlying method extends the search to additional centroids ensuring that the needed K rows are returned. Terminable Iteration for IVF Index
Using JSON in Included Columns with IVF Indexes	Restrictions have been removed on JSON, `BLOB`, and `CLOB` data types, enabling broader and more flexible usage of these data types in included columns. Included Columns

Autonomous Database Updates

The following sections include new AI Vector Search features introduced in Oracle Autonomous Database as part of the listed update.

- June 2024
  Included are some notable Oracle AI Vector Search updates with the Autonomous Database, June 2024 release.
- July 2024
  Included are some notable Oracle AI Vector Search updates with the Autonomous Database, July 2024 release.
- August 2024
  No additional features were introduced for Oracle Autonomous Database in August 2024.
- September 2024
  No additional features were introduced for Oracle Autonomous Database in September 2024.

- October 2024
  New features are available for Oracle Database in October with Release Update 23.6.
- November 2024
  No additional features were introduced for Oracle Autonomous Database in November 2024.
- December 2024
  No additional features were introduced for Oracle Autonomous Database in December 2024.
- January 2025
  New features are available for Oracle Database in January with Release Update 23.7.
- February 2025
  No additional features were introduced for Oracle Autonomous Database in February 2025.
- March 2025
  No additional features were introduced for Oracle Autonomous Database in March 2025.
- April 2025
  New features are available for Oracle Database in April with Release Update 23.8.

June 2024

Included are some notable Oracle AI Vector Search updates with the Autonomous Database, June 2024 release.

Feature	Description
HNSW Index Support in RAC Environments	It is possible to use Hierarchical Navigable Small World (HNSW) indexes with `SELECT` statements in RAC environments. For more information, see Understand HNSW Index Population Mechanisms in Oracle RAC or Single Instance.
Sizing the Vector Pool with ADB-S	With Autonomous Database Serverless (ADB-S) services, the vector pool dynamically grows and shrinks. It cannot be explicitly set. For more information, see Size the Vector Pool.
SQL Quick Start with Vector Generator	A SQL scenario is provided to get you started using Oracle AI Vector Search. The quick start includes a PL/SQL program that creates a vector generator, offering a simple alternative to using a vector embedding model. For the full tutorial, see SQL Quick Start Using a FLOAT32 Vector Generator.
Similarity Search and Index Creation Syntax	• The `APPROX` and `APPROXIMATE` keywords are now optional. If omitted while connected to an ADB-S instance, an approximate search using a vector index is attempted if one exists. For more information about approximate search with indexes, see Approximate Search Using HNSW and Approximate Search Using IVF. • The `NEIGHBOR` keyword in the `ORGANIZATION` clause of `CREATE VECTOR INDEX` is now optional. For the full syntax to create HNSW and IVF indexes, see Hierarchical Navigable Small World Index Syntax and Parameters and Inverted File Flat Index Syntax and Parameters.

July 2024

Included are some notable Oracle AI Vector Search updates with the Autonomous Database, July 2024 release.

Feature	Description
New Vector Index Views	The following vector index views are now available with Autonomous Database Serverless (ADB-S) services: • V$VECTOR_INDEX • V$VECTOR_GRAPH_INDEX • V$VECTOR_PARTITIONS_INDEX

August 2024

No additional features were introduced for Oracle Autonomous Database in August 2024.

September 2024

No additional features were introduced for Oracle Autonomous Database in September 2024.

October 2024

New features are available for Oracle Database in October with Release Update 23.6.

For information, see October 2024, Release Update 23.6.

November 2024

No additional features were introduced for Oracle Autonomous Database in November 2024.

December 2024

No additional features were introduced for Oracle Autonomous Database in December 2024.

January 2025

New features are available for Oracle Database in January with Release Update 23.7.

For information, see January 2025, Release Update 23.7.

February 2025

No additional features were introduced for Oracle Autonomous Database in February 2025.

March 2025

No additional features were introduced for Oracle Autonomous Database in March 2025.

April 2025

New features are available for Oracle Database in April with Release Update 23.8.

For information, see April 2025, Release Update 23.8.

Deprecated Features

The following features are deprecated, and may be desupported in a future release.

Starting with Oracle Database 23ai, Release Update 23.7, the python packages EmbeddingModel and EmbeddingModelConfig are deprecated. These packages are replaced with ONNXPipeline and ONNXPipelineConfig respectively.

- Details and utility of the deprecated package can be found here : Python Classes to Convert Pretrained Models to ONNX Models (Deprecated)
- Oracle recommends that you use the latest version. You can find the details of the updated python classes in Import Pretrained Models in ONNX Format

2
Overview

Oracle AI Vector Search stores and indexes vector embeddings for fast retrieval and similarity search.

- Overview of Oracle AI Vector Search
  Oracle AI Vector Search is designed for Artificial Intelligence (AI) workloads and allows you to query data based on semantics, rather than keywords.
- Why Use Oracle AI Vector Search?
  One of the biggest benefits of Oracle AI Vector Search is that semantic search on unstructured data can be combined with relational search on business data in one single system.
- Oracle AI Vector Search Workflow
  A typical Oracle AI Vector Search workflow follows the included primary steps.

Overview of Oracle AI Vector Search

Oracle AI Vector Search is designed for Artificial Intelligence (AI) workloads and allows you to query data based on semantics, rather than keywords.

VECTOR Data Type

The `VECTOR` data type is introduced with the release of Oracle Database 23ai, providing the foundation to store vector embeddings alongside business data in the database. Using embedding models, you can transform unstructured data into vector embeddings that can then be used for semantic queries on business data. In order to use the `VECTOR` data type and its related features, the `COMPATIBLE` initialization parameter must be set to 23.4.0 or higher. For more information about the parameter and how to change it, see *Oracle Database Upgrade Guide*.

See the following basic example of using the `VECTOR` data type in a table definition:

```
CREATE TABLE docs (doc_id INT, doc_text CLOB, doc_vector VECTOR);
```

For more information about the `VECTOR` data type and how to use vectors in tables, see Create Tables Using the VECTOR Data Type.

Due to the numerical nature of the `VECTOR` data type, you can use it as an input to the machine learning algorithms such as classification, anomaly, regression, clustering and feature extraction. More details on using `VECTOR` data type in machine learning could be found in Vector Data Type Support.

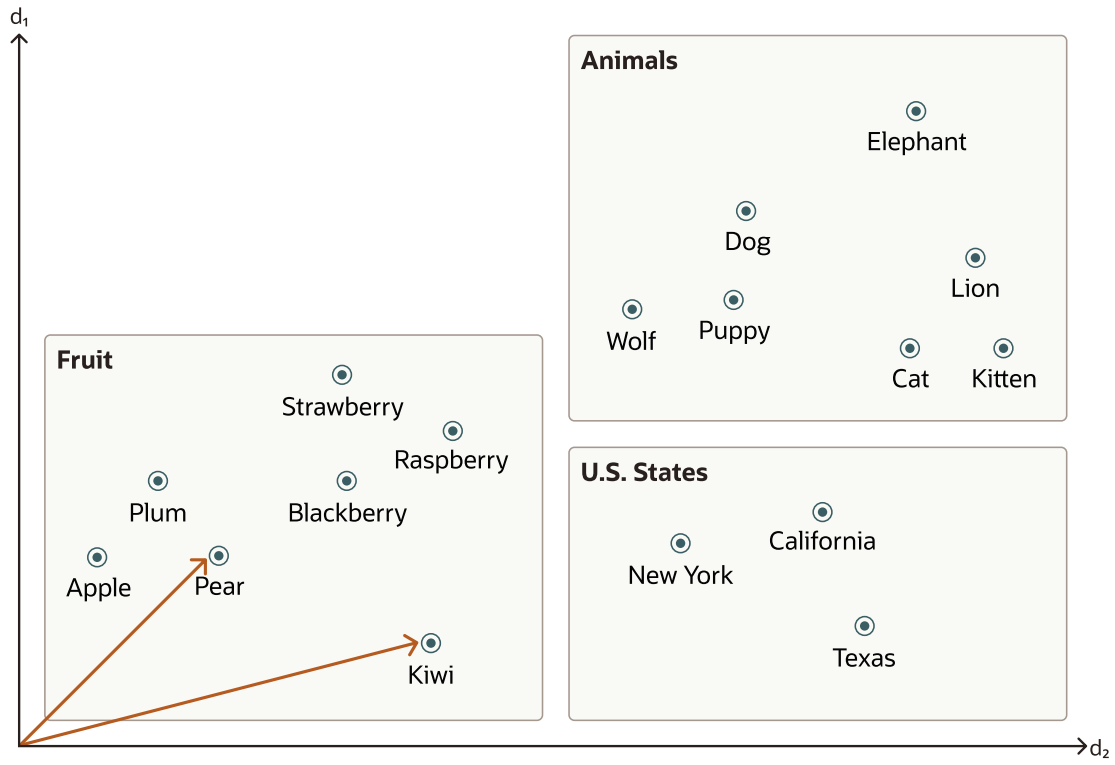
> **Note:**
>
> Support for `VECTOR` data type machine learning is available in all versions starting 23.7, but not in earlier versions.

Vector Embeddings

If you've ever used applications such as voice assistants, chatbots, language translators, recommendation systems, anomaly detection, or video search and recognition, you've implicitly used vector embeddings features.

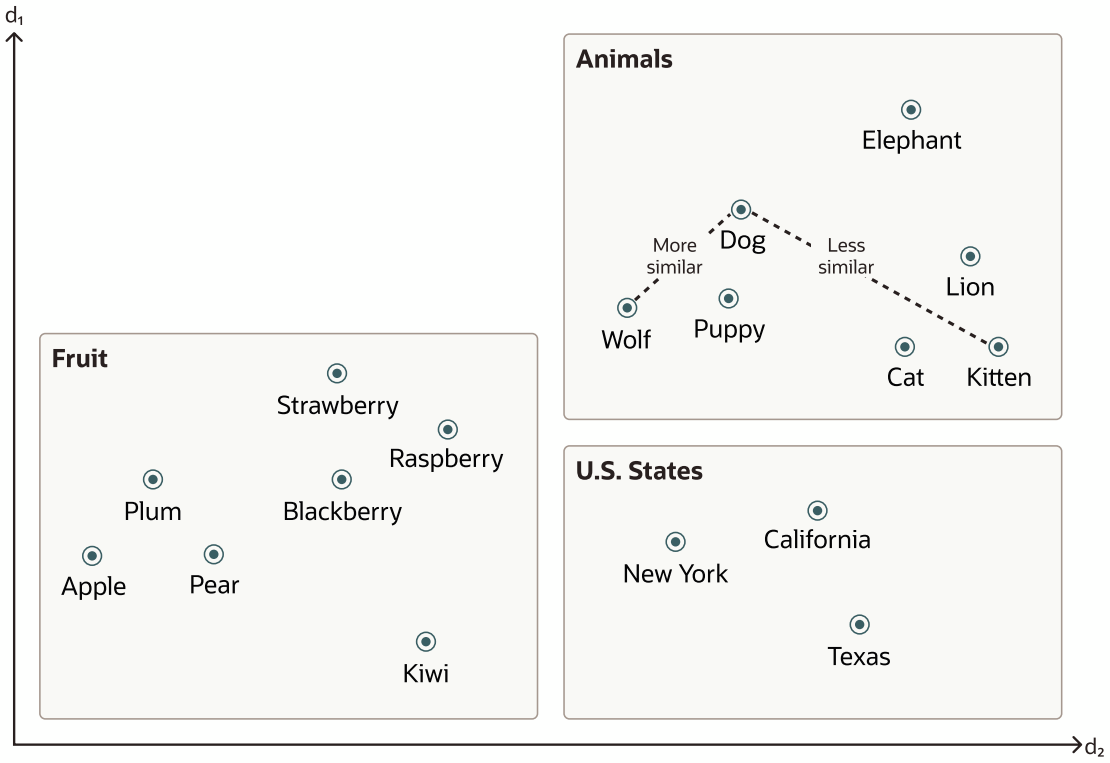
Oracle AI Vector Search stores vector embeddings, which are mathematical vector representations of data points. These vector embeddings describe the semantic meaning behind content such as words, documents, audio tracks, or images. As an example, while doing text based searches, vector search is often considered better than keyword search as vector search is based on the meaning and context behind the words and not the actual words themselves. This vector representation translates semantic similarity of objects, as perceived by humans, into proximity in a mathematical vector space. This vector space usually has multihundreds, if not thousands, of dimensions. Put differently, vector embeddings are a way of representing almost any kind of data, such as text, images, videos, users, or music as points in a multidimensional space where the locations of those points in space, and proximity to others, are semantically meaningful.

This simplified diagram illustrates a vector space where words are encoded as 2-dimensional vectors.



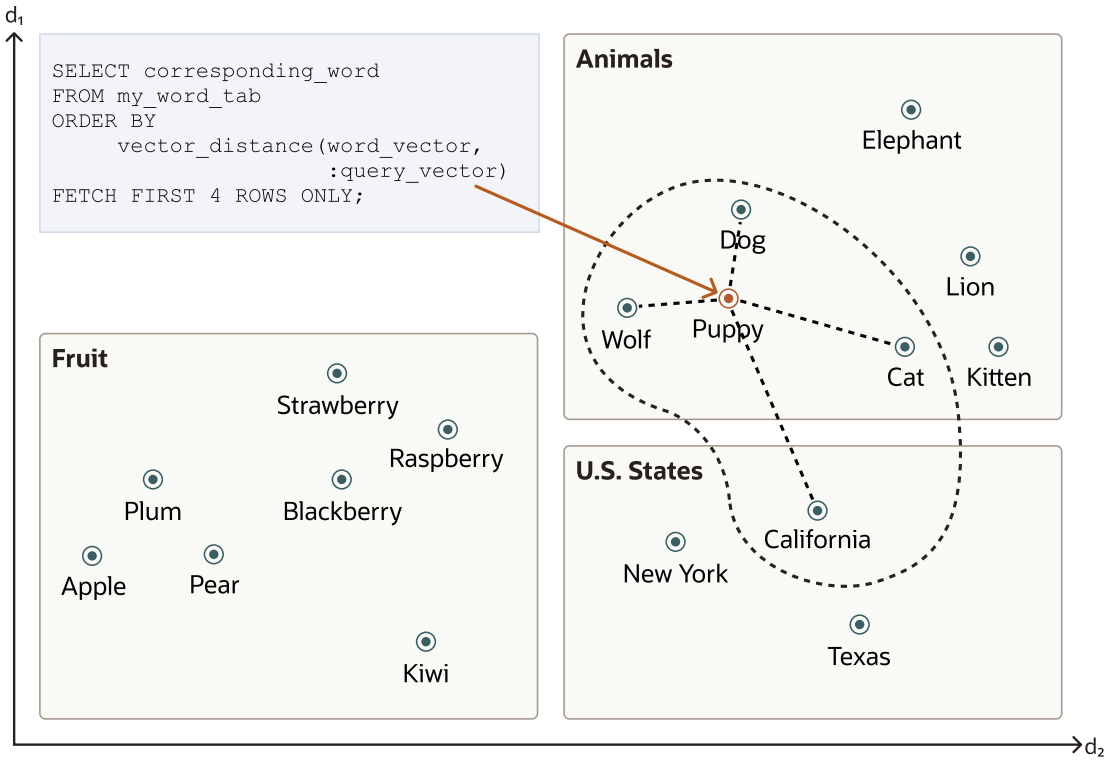
Similarity Search

Searching semantic similarity in a data set is now equivalent to searching nearest neighbors in a vector space instead of using traditional keyword searches using query predicates. As illustrated in the following diagram, the distance between *dog* and *wolf* in this vector space is shorter than the distance between *dog* and *kitten*. In this space, a dog is more similar to a wolf than it is to a kitten. See Perform Exact Similarity Search for more information.



Vector data tends to be unevenly distributed and clustered into groups that are semantically related. Doing a similarity search based on a given query vector is equivalent to retrieving the K-nearest vectors to your query vector in your vector space. Basically, you need to find an ordered list of vectors by ranking them, where the first row in the list is the closest or most similar vector to the query vector, the second row in the list is the second closest vector to the query vector, and so on. When doing a similarity search, the relative order of distances is what really matters rather than the actual distance.

Using the preceding vector space, here is an illustration of a semantic search where your query vector is the one corresponding to the word *Puppy* and you want to identify the four closest words:



Similarity searches tend to get data from one or more clusters depending on the value of the query vector and the fetch size.

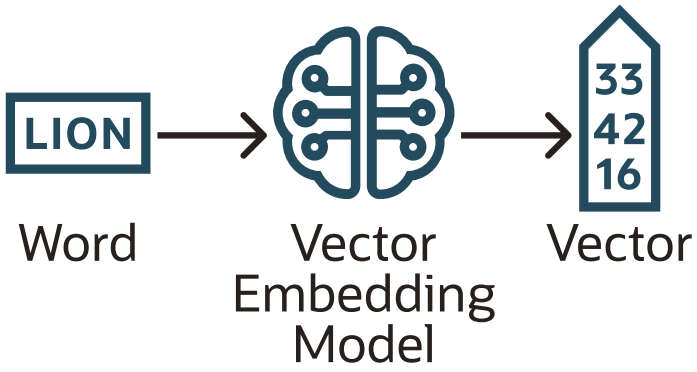
Approximate searches using *vector indexes* can limit the searches to specific clusters, whereas exact searches visit vectors across all clusters. See Use Vector Indexes for more information.

Vector Embedding Models

One way of creating such vector embeddings could be to use someone's domain expertise to quantify a predefined set of features or dimensions such as shape, texture, color, sentiment, and many others, depending on the object type with which you're dealing. However, the efficiency of this method depends on the use case and is not always cost effective.

Instead, vector embeddings are created via neural networks. Most modern vector embeddings use a transformer model, as illustrated by the following diagram, but convolutional neural networks can also be used.

Figure 2-1 Vector Embedding Model



Depending on the type of your data, you can use different pretrained, open-source models to create vector embeddings. For example:

- For textual data, sentence transformers transform words, sentences, or paragraphs into vector embeddings.
- For visual data, you can use Residual Network (ResNet) to generate vector embeddings.
- For audio data, you can use the visual spectrogram representation of the audio data to fall back into the visual data case.

Each model also determines the number of dimensions for your vectors. For example:

- Cohere's embedding model embed-english-v3.0 has 1024 dimensions.
- OpenAI's embedding model text-embedding-3-large has 3072 dimensions.
- Hugging Face's embedding model all-MiniLM-L6-v2 has 384 dimensions

Of course, you can always create your own model that is trained with your own data set.

Import Embedding Models into Oracle Database

Although you can generate vector embeddings outside the Oracle Database using pretrained open-source embeddings models or your own embeddings models, you also have the option to import those models directly into the Oracle Database if they are compatible with the Open Neural Network Exchange (ONNX) standard. Oracle Database implements an ONNX runtime directly within the database. This allows you to generate vector embeddings directly within the Oracle Database using SQL. See Generate Vector Embeddings for more information.

Why Use Oracle AI Vector Search?

One of the biggest benefits of Oracle AI Vector Search is that semantic search on unstructured data can be combined with relational search on business data in one single system.

This is not only powerful but also significantly more effective because you don't need to add a specialized vector database, eliminating the pain of data fragmentation between multiple systems.

For example, suppose you use an application that allows you to find a house that is similar to a picture you took of one you like that is located in your preferred area for a certain budget. Finding a good match in this case requires combining a semantic picture search with searches on business data.

With Oracle AI Vector Search, you can create the following table:

```
CREATE TABLE house_for_sale (house_id     NUMBER,
                             price        NUMBER,
                             city         VARCHAR2(400),
                             house_photo  BLOB,
                             house_vector VECTOR);
```

The following sections of this guide describe in detail the meaning of the `VECTOR` data type and how to load data in this column data type.

With that table, you can run the following query to answer your basic question:

```
SELECT house_photo, city, price
FROM   house_for_sale
WHERE  price <= :input_price AND
```

```
       city  = :input_city
ORDER BY VECTOR_DISTANCE(house_vector, :input_vector);
```

Later sections of this guide describe in detail the meaning of the `VECTOR_DISTANCE` function. This query is just to show you how simple it is to combine a vector embedding similarity search with relation predicates.

In conjunction with Oracle Database 23ai, Oracle Exadata System Software release 24.1.0 introduces AI Smart Scan, a collection of Exadata-specific optimizations capable of improving the performance of various AI vector query operations by orders of magnitude.

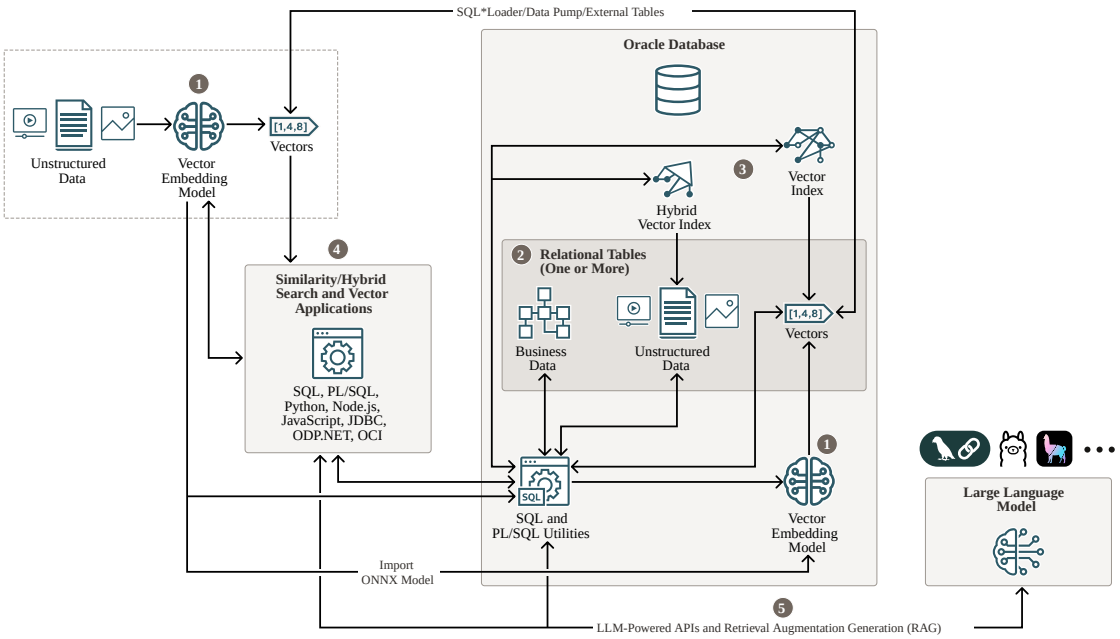
AI Smart Scan automatically accelerates Oracle Database 23ai AI Vector Search with optimizations that deliver low-latency parallelized scans across massive volumes of vector data. AI Smart Scan processes vector data at memory speed, leveraging ultra-fast Exadata RDMA Memory (XRMEM) and Exadata Smart Flash Cache in the Exadata storage servers, and performs vector distance computations and top-K filtering at the data source, avoiding unnecessary network data transfer and database server processing.

Oracle AI Vector Search Workflow

A typical Oracle AI Vector Search workflow follows the included primary steps.

This is illustrated in the following diagram:

Figure 2-2 Oracle AI Vector Search Use Case Flowchart



> **Note:**
>
> Find all of our Interactive Architecture Diagrams on the Oracle Help Center.

Oracle AI Vector Search is designed for Artificial Intelligence (AI) workloads. It allows you to query data based on semantics and image similarity rather than simply keywords. The

preceding diagram shows the possible steps you must take to manage vector embeddings with Oracle AI Vector Search.

Primary workflow steps:

1. **Generate Vector Embeddings from Your Unstructured Data**
   You can perform this step either outside or within Oracle Database. To perform this step inside Oracle Database, you must first import a vector embedding model using the ONNX standard. Your unstructured data can reside within or outside Oracle Database.

   For more information, see Generate Vector Embeddings.

2. **Store Vector Embeddings, Unstructured Data, and Relational Business Data in Oracle Database**
   After you have generated the vector embeddings, you can store them along with the corresponding unstructured and relational business data. If vector embeddings are stored outside Oracle Database, you can use SQL*Loader or Data Pump to load the vector embedding inside a relational table within Oracle Database. It is also possible to access vector embeddings stored outside the database through external tables.

   For more information, see Store Vector Embeddings.

3. **Create Vector Indexes and Hybrid Vector Indexes**
   Similar to how you create indexes on regular table columns, you can create vector indexes on vector embeddings, and you can create hybrid vector indexes (a combination of Oracle Text index and vector index) on your unstructured data. This is beneficial for running similarity searches over huge vector spaces.

   For more information, see Create Vector Indexes and Hybrid Vector Indexes.

4. **Query Data with Similarity and Hybrid Searches**
   You can then use Oracle AI Vector Search native SQL operations to combine similarity with traditional relational key searches. In addition, you can run hybrid searches, an advanced information retrieval technique that combines both the similarity and keyword searches to achieve highly relevant search results. SQL and PL/SQL provide powerful utilities to transform unstructured data, such as documents, into chunks before generating vector embeddings on each chunk.

   For more information, see Query Data With Similarity and Hybrid Searches and Supported Clients and Languages.

5. **Generate a Prompt and Send it to an LLM for a Full RAG Inference**
   You can use vector utility PL/SQL APIs for prompting large language models (LLMs) with textual prompts and images using LLM-powered interfaces. LLMs inherently lack the ability to access or incorporate new information after their training cutoff. By providing your LLM with up-to-date facts from your company, you can minimize the probability that an LLM will make up answers (hallucinate). Retrieval Augmented Generation (RAG) is an approach developed to address the limitations of LLMs. RAG combines the strengths of pretrained language models, including reranking ones, with the ability to retrieve information from a dataset or database in real time during the generation of responses. Oracle AI Vector Search enables RAG and LLM integration using popular frameworks like LangChain, Ollama, and LlamaIndex.

   For more information, see Work with LLM-Powered APIs and Retrieval Augmented Generation.

3
Get Started

To get started, review the steps for the different tasks that you can do with Oracle AI Vector Search.

- SQL Quick Start Using a Vector Embedding Model Uploaded into the Database
  A set of SQL commands is provided to run a particular scenario that will help you understand Oracle AI Vector Search capabilities.
- SQL Quick Start Using a FLOAT32 Vector Generator
  A PL/SQL program that creates a vector generator is included along with example queries and results, providing a simple way to get started with Oracle AI Vector Search without a vector embedding model.
- SQL Quick Start Using a BINARY Vector Generator
  A set of procedures generate `BINARY` vectors, providing a simple way to get started with Oracle AI Vector Search without a vector embedding model.
- Your Vector Documentation Map to GenAI Prompts
  Follow the included steps to increase your chance of getting better and more consistent answers about this document from your preferred Generative AI Chatbot with internet search capabilities.

SQL Quick Start Using a Vector Embedding Model Uploaded into the Database

A set of SQL commands is provided to run a particular scenario that will help you understand Oracle AI Vector Search capabilities.

This quick start scenario introduces you to the `VECTOR` data type, which represents the semantic meaning behind your unstructured data. You will also use Vector Search SQL operators, allowing you to perform a similarity search to find vectors (and thereby content) that are similar to each other. Vector indexes are also created to help you accelerate similarity searches in an approximate manner. In addition to pure similarity searches, you will create Hybrid Vector Indexes and use them to run hybrid searches.

See Overview of Oracle AI Vector Search for more introductory information if needed.

The script chunks two Oracle Database Documentation books, assigns them corresponding vector embeddings, and shows you some similarity searches using vector indexes.

To run this script you need three files similar to the following:

- `my_embedding_model.onnx`, which is an ONNX export of the corresponding embedding model. To create such a file, see Convert Pretrained Models to ONNX Model: End-to-End Instructions or download the provided Hugging Face all-MiniLM-L12-v2 model in ONNX format.
- `json-relational-duality-developers-guide.pdf`, which is the PDF file for JSON-Relational Duality Developer's Guide
- `oracle-database-23ai-new-features-guide.pdf`, which is the PDF file for Oracle Database New Features.

> **Note:**
>
> You can use other PDF files instead of the ones listed here. If you prefer, you can use another model of your choice as long as you can generate it as an `.onnx` file.

Let's start.

1. Copy the files to your local server directory or Oracle Cloud Infrastructure (OCI) Object Storage.

   If you have downloaded the Hugging Face all-MiniLM-L12-v2 model in ONNX format, run the following before completing the rest of this step:

   ```
   unzip all-MiniLM-L12-v2_augmented.zip
   ```

   Result:

   ```
   Archive:   all-MiniLM-L12-v2_augmented.zip
   inflating: all_MiniLM_L12_v2.onnx
   inflating: README-ALL_MINILM_L12_V2-augmented.txt
   ```

   For more information about the all-MiniLM-L12-v2 model, see the blog post Now Available! Pre-built Embedding Generation model for Oracle Database 23ai.

   You have the option to use the `scp` command for the local case and Oracle Command Line Interface or cURL for the Object Storage case. In the Object Storage case, the relevant command using Oracle Command Line Interface is `oci os object put`.

   If using Object Storage, you can also mount the bucket on your database server using the following steps (executed as a `root` user). This will allow you to easily copy files to Object Storage.

   a. Install the `s3fs-fuse` package.

      ```
      yum install -y s3fs-fuse
      ```

   b. On OCI, create a Customer Secret key. Make sure to save the `ACCESS_KEY` and `SECRET_KEY` in your notes.

   c. Create a folder that will be the mount point for the object storage bucket.

      ```
      mkdir /mnt/bucket
      chown oracle:oinstall /mnt/bucket
      ```

   d. Put your Customer Secret key in a file that will be used to authenticate to OCI Object Storage.

      ```
      echo $ACCESS_KEY:$SECRET_KEY > .passwd=s3fs
      chmod 400 .passwd-s3fs
      ```

   e. Mount your object storage bucket in the mount point folder (note this is a one line command).

      ```
      s3fs ${BUCKET} /mnt/bucket -o passwd_file=.passwd-s3fs
      -o url=https://${NAMESPACE}.compat.objectstorage.$
      ```

```
{REGION}.oraclecloud.com
-onomultipart -o use_path_request_style -o endpoint=${REGION} -ouid=$
{ORAUID},
gid=${ORAGID},allow_other,mp_umask=022
```

f. If you want to make the mount permanent after reboot, you can create a crontab entry (note this is a one line command).

```
echo "@reboot s3fs ${BUCKET} /mnt/bucket -o passwd_file=.passwd-s3fs -o
url=https://${NAMESPACE}.compat.objectstorage.${REGION}.oraclecloud.com
-onomultipart
-o use_path_request_style -o endpoint=${REGION} -ouid=${ORAUID},gid=$
{ORAGID},
allow_other,mp_umask=022" > crontab-fragment.txt
```

g. Add the crontab entry to your server crontab.

```
crontab -l | cat - crontab-fragment.txt >crontab.txt && crontab
crontab.txt

rm -f crontab.txt crontab-fragment.txt
```

As an alternative to the preceding Object Storage instructions, you also have the option to use the `LOAD_ONNX_MODEL_CLOUD` procedure of the `DBMS_VECTOR` package to load an ONNX embedding model from cloud Object Storage. For more information about the procedure, see *Oracle Database PL/SQL Packages and Types Reference*.

2. Create storage, user, and privileges.

   Here you create a new tablespace and a new user. You grant that user the `DB_DEVELOPER_ROLE` and create an Oracle directory to point to the PDF files. You grant the new user the possibility to read and write from/to that directory.

```
sqlplus / as sysdba

CREATE TABLESPACE tbs1
DATAFILE 'tbs5.dbf' SIZE 20G AUTOEXTEND ON
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;

drop user vector cascade;

create user vector identified by vector DEFAULT TABLESPACE tbs1 quota
unlimited on tbs1;

grant DB_DEVELOPER_ROLE to vector;

create or replace directory VEC_DUMP as '/my_local_dir/';

grant read, write on directory vec_dump to vector;
```

3. Load your embedding model into the Oracle Database using one of the following methods, depending where your ONNX file is stored.

   a. If your ONNX file is stored locally:

Using the `DBMS_VECTOR` package, load your embedding model into the Oracle Database. You must specify the directory where you stored your model in ONNX format as well as describe what type of model it is and how you want to use it.

For more information about downloading pretrained embedding models, converting them into ONNX format, and importing the ONNX file into Oracle Database, see Import Pretrained Models in ONNX Format.

i.
```
connect vector/<vector user password>@<pdb instance network name>

EXEC DBMS_VECTOR.DROP_ONNX_MODEL(model_name => 'doc_model', force
=> true);

EXECUTE DBMS_VECTOR.LOAD_ONNX_MODEL('VEC_DUMP',
                                    'my_embedding_model.onnx',
                                    'doc_model');
```

> **Note:**
>
> If you have downloaded the Oracle pre-built ONNX file for the Hugging Face all-MiniLM-L12-v2 model, you can use the following simplified command to load the file in the Database:
>
> ```
> BEGIN
>    DBMS_VECTOR.LOAD_ONNX_MODEL(
>         directory => 'DM_DUMP',
>         file_name => 'all_MiniLM_L12_v2.onnx',
>         model_name => 'ALL_MINILM_L12_V2');
> END;
> /
> ```

b. If your ONNX file is already stored on object store:

i. Create the credentials:

```
SET DEFINE OFF;
BEGIN
    DBMS_CLOUD.create_credential(
    credential_name => 'MY_CLOUD_CREDENTIAL',
    username => '<username>',
    password => '<user password>';
END;
/
```

ii. List the ONNX model saved on Object Storage. The `location_uri` should correspond to the location of the ONNX file:

```
SELECT mod.object_name,
       round(mod.bytes/1024/1024,2) size_mb,
       to_date(substr(mod.last_modified,1,18), 'DD-MON-RR
HH24.MI.SS') modified
  FROM table(dbms_cloud.list_objects(credential_name =>
'CLOUD_CREDENTIAL',
```

```
          location_uri => 'https://
objectstorage.<region>.oraclecloud.com/n/<namespace>/b/
<bucketName>/o/')) mod
  WHERE mod.object_name = 'my_embedding_model.onnx';
```

iii. Import the model into the database:

```
DECLARE
  model_source BLOB := NULL;
BEGIN
  model_source := DBMS_CLOUD.get_object(credential_name =>
'MY_CLOUD_CREDENTIAL',
    object_uri => 'https://objectstorage.<region>.oraclecloud.com/n/
<namespace>/b/<bucketName>/o/my_embedding_model.onnx');
  DBMS_VECTOR.load_onnx_model('doc_model',
                              model_source);
END;
/
```

iv. List the model inside the database:

```
SELECT MODEL_NAME, ATTRIBUTE_NAME, ATTRIBUTE_TYPE, DATA_TYPE,
VECTOR_INFO
  FROM USER_MINING_MODEL_ATTRIBUTES WHERE MODEL_NAME = 'doc_model';

SELECT MODEL_NAME, MINING_FUNCTION, ALGORITHM, ALGORITHM_TYPE,
MODEL_SIZE
  FROM USER_MINING_MODELS WHERE MODEL_NAME = 'doc_model';
```

4. (Optional) Verify that the loaded embedding model is working.

   You can call the `VECTOR_EMBEDDING` SQL function to test your embedding model by passing a sample text string (`hello`) as the input.

```
SELECT TO_VECTOR(VECTOR_EMBEDDING(doc_model USING 'hello' as data)) AS
embedding;
```

5. Create a relational table to store books in the PDF format.

   You now create a table containing all the books you want to chunk and vectorize. You associate each new book with an ID and a pointer to your local directory where the books are stored.

```
drop table documentation_tab purge;
create table documentation_tab (id number, data blob);
insert into documentation_tab values(1, to_blob(bfilename('VEC_DUMP',
'json-relational-duality-developers-guide.pdf')));
insert into documentation_tab values(2, to_blob(bfilename('VEC_DUMP',
'oracle-database-23ai-new-features-guide.pdf')));
commit;
select dbms_lob.getlength(data) from documentation_tab;
```

6. Create a relational table to store unstructured data chunks and associated vector embeddings using `my_embedding_model.onnx`.

You start by creating the table structure using the `VECTOR` data type. For more information about declaring a table's column as a `VECTOR` data type, see Create Tables Using the VECTOR Data Type .

The `INSERT` statement reads each PDF file from `DOCUMENTATION_TAB`, transforms each PDF file into text, chunks each resulting text, then finally generates corresponding vector embeddings on each chunk that is created. All that is done in one single `INSERT SELECT` statement.

Here you choose to use Vector Utility PL/SQL package `DBMS_VECTOR_CHAIN` to convert, chunk, and vectorize your unstructured data in one end-to-end pipeline. Vector Utility PL/SQL functions are intended to be a set of chainable stages (using table functions) through which you pass your input data to transform into a different representation. In this case, from PDF to text to chunks to vectors. For more information about using chainable utility functions in the `DBMS_VECTOR_CHAIN` package, see About Chainable Utility Functions and Common Use Cases.

```
drop table doc_chunks purge;
create table doc_chunks (doc_id number, chunk_id number, chunk_data
varchar2(4000), chunk_embedding vector);

insert into doc_chunks
select dt.id doc_id, et.embed_id chunk_id, et.embed_data chunk_data,
to_vector(et.embed_vector) chunk_embedding
from
    documentation_tab dt,
    dbms_vector_chain.utl_to_embeddings(

dbms_vector_chain.utl_to_chunks(dbms_vector_chain.utl_to_text(dt.data),
json('{"normalize":"all"}')),
       json('{"provider":"database", "model":"doc_model"}')) t,
    JSON_TABLE(t.column_value, '$[*]' COLUMNS (embed_id NUMBER PATH
'$.embed_id', embed_data VARCHAR2(4000) PATH '$.embed_data', embed_vector
CLOB PATH '$.embed_vector')) et;

commit;
```

> **See Also:**
>
> - *Oracle Database JSON Developer's Guide* for information about the `JSON_TABLE` function, which supports the `VECTOR` data type

7. Generate a query vector for use in a similarity search.

   For a similarity search you will need query vectors. Here you enter your query text and generate an associated vector embedding.

   For example, you can use the following text: 'different methods of backup and recovery'. You use the `VECTOR_EMBEDDING` SQL function to generate the vector embeddings from the input text. The function takes an embedding model name and a text string to generate the corresponding vector. Note that you can generate vector embeddings outside of the database using your favorite tools. For more information about using the `VECTOR_EMBEDDING` SQL function, see About SQL Functions to Generate Embeddings.

In SQL*Plus, use the following code:

```
ACCEPT text_input CHAR PROMPT 'Enter text: '
VARIABLE text_variable VARCHAR2(1000)
VARIABLE query_vector VECTOR
BEGIN
  :text_variable := '&text_input';
  SELECT vector_embedding(doc_model using :text_variable as data)
into :query_vector;
END;
/

PRINT query_vector
```

In SQLCL, use the following code:

```
DEFINE text_input = '&text'

SELECT '&text_input';

VARIABLE text_variable VARCHAR2(1000)
VARIABLE query_vector CLOB
BEGIN
  :text_variable := '&text_input';
  SELECT vector_embedding(doc_model using :text_variable as data)
into :query_vector;
END;
/

PRINT query_vector
```

8. Run a similarity search to find, within your books, the first four most relevant chunks that talk about backup and recovery.

   Using the generated query vector, you search similar chunks in the `DOC_CHUNKS` table. For this, you use the `VECTOR_DISTANCE` SQL function and the `FETCH` SQL clause to retrieve the most similar chunks.

   For more information about the `VECTOR_DISTANCE` SQL function, see Vector Distance Functions and Operators.

   For more information about exact similarity search, see Perform Exact Similarity Search.

   ```
   SELECT doc_id, chunk_id, chunk_data
   FROM doc_chunks
   ORDER BY vector_distance(chunk_embedding , :query_vector, COSINE)
   FETCH FIRST 4 ROWS ONLY;
   ```

   You can also add a `WHERE` clause to further filter your search, for instance if you only want to look at one particular book.

   ```
   SELECT doc_id, chunk_id, chunk_data
   FROM doc_chunks
   WHERE doc_id=1
   ```

```
ORDER BY vector_distance(chunk_embedding , :query_vector, COSINE)
FETCH FIRST 4 ROWS ONLY;
```

9. Use the `EXPLAIN PLAN` command to determine how the optimizer resolves this query.

```
EXPLAIN PLAN FOR
SELECT doc_id, chunk_id, chunk_data
FROM doc_chunks
ORDER BY vector_distance(chunk_embedding , :query_vector, COSINE)
FETCH FIRST 4 ROWS ONLY;

select plan_table_output from
table(dbms_xplan.display('plan_table',null,'all'));


PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
---
Plan hash value: 1651750914
--------------------------------------------------------------------------------
----------------------
| Id  | Operation               | Name          | Rows  | Bytes |TempSpc|
Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
----------------------
|   0 | SELECT STATEMENT        |               |     4 |   104 |
|   549   (3)| 00:00:01 |
|*  1 |  COUNT STOPKEY          |               |       |       |
|            |          |
|   2 |   VIEW                  |               |  5014 |   127K|
|   549   (3)| 00:00:01 |
|*  3 |    SORT ORDER BY STOPKEY|               |  5014 |   156K|
232K|   549   (3)| 00:00:01 |
|   4 |     TABLE ACCESS FULL   | DOC_CHUNKS    |  5014 |   156K|
|   480   (3)| 00:00:01 |
--------------------------------------------------------------------------------
----------------------
```

10. Run a multi-vector similarity search to find, within your books, the first four most relevant chunks in the first two most relevant books.

    Here you keep using the same query vector as previously used.

    For more information about performing multi-vector similarity search, see Perform Multi-Vector Similarity Search.

```
SELECT doc_id, chunk_id, chunk_data
FROM doc_chunks
ORDER BY vector_distance(chunk_embedding , :query_vector, COSINE)
FETCH FIRST 2 PARTITIONS BY doc_id, 4 ROWS ONLY;
```

11. Create an In-Memory Neighbor Graph Vector Index on the vector embeddings that you created.

    When dealing with huge vector embedding spaces, you may want to create vector indexes to accelerate your similarity searches. Instead of scanning each and every vector

embedding in your table, a vector index uses heuristics to reduce the search space to accelerate the similarity search. This is called approximate similarity search.

For more information about creating vector indexes, see Create Vector Indexes and Hybrid Vector Indexes.

> **Note:**
>
> You must have explicit `SELECT` privilege to select from the `VECSYS.VECTOR$INDEX` table, which gives you detailed information about your vector indexes.

```
create vector index docs_hnsw_idx on doc_chunks(chunk_embedding)
organization inmemory neighbor graph
distance COSINE
with target accuracy 95;

SELECT INDEX_NAME, INDEX_TYPE, INDEX_SUBTYPE
FROM USER_INDEXES;
```

Result:

```
INDEX_NAME     INDEX_TYPE  INDEX_SUBTYPE
-------------- ----------- ----------------------------
DOCS_HNSW_IDX  VECTOR      INMEMORY_NEIGHBOR_GRAPH_HNSW


 SELECT JSON_SERIALIZE(IDX_PARAMS returning varchar2 PRETTY)
FROM VECSYS.VECTOR$INDEX where IDX_NAME = 'DOCS_HNSW_IDX';
```

Result:

```
JSON_SERIALIZE(IDX_PARAMSRETURNINGVARCHAR2PRETTY)
_____________________________________________________________________
{
  "type" : "HNSW",
  "num_neighbors" : 32,
  "efConstruction" : 300,
  "distance" : "COSINE",
  "accuracy" : 95,
  "vector_type" : "FLOAT32",
  "vector_dimension" : 384,
  "degree_of_parallelism" : 1,
  "pdb_id" : 3,
  "indexed_col" : "CHUNK_EMBEDDING"
}
```

12. Determine the memory allocation in the vector memory area.

    To get an idea about the size of your In-Memory Neighbor Graph Vector Index in memory, you can use the `V$VECTOR_MEMORY_POOL` view. See Size the Vector Pool for more information about sizing the vector pool to allow for vector index creation and maintenance.

> **Note:**
>
> You must have explicit `SELECT` privilege to select from the `V$VECTOR_MEMORY_POOL` view, which gives you detailed information about the vector pool.

```
select CON_ID, POOL, ALLOC_BYTES/1024/1024 as ALLOC_BYTES_MB,
USED_BYTES/1024/1024 as USED_BYTES_MB
from V$VECTOR_MEMORY_POOL order by 1,2;
```

13. Run an approximate similarity search to identify, within your books, the first four most relevant chunks.

    Using the previously generated query vector, you search chunks in the `DOC_CHUNKS` table that are similar to your query vector. For this, you use the `VECTOR_DISTANCE` function and the `FETCH APPROX` SQL clause to retrieve the most similar chunks using your vector index.

    For more information about approximate similarity search, see Perform Approximate Similarity Search Using Vector Indexes.

    ```
    SELECT doc_id, chunk_id, chunk_data
    FROM doc_chunks
    ORDER BY vector_distance(chunk_embedding , :query_vector, COSINE)
    FETCH APPROX FIRST 4 ROWS ONLY WITH TARGET ACCURACY 80;
    ```

    You can also add a `WHERE` clause to further filter your search, for instance if you only want to look at one particular book.

    ```
    SELECT doc_id, chunk_id, chunk_data
    FROM doc_chunks
    WHERE doc_id=1
    ORDER BY vector_distance(chunk_embedding , :query_vector, COSINE)
    FETCH APPROX FIRST 4 ROWS ONLY WITH TARGET ACCURACY 80;
    ```

14. Use the `EXPLAIN PLAN` command to determine how the optimizer resolves this query.

    See **Optimizer Plans for Vector Indexes** for more information about how the Oracle Database optimizer uses vector indexes to run your approximate similarity searches.

    ```
    EXPLAIN PLAN FOR
    SELECT doc_id, chunk_id, chunk_data
    FROM doc_chunks
    ORDER BY vector_distance(chunk_embedding , :query_vector, COSINE)
    FETCH APPROX FIRST 4 ROWS ONLY WITH TARGET ACCURACY 80;

    select plan_table_output from
    table(dbms_xplan.display('plan_table',null,'all'));


    PLAN_TABLE_OUTPUT
    ------------------------------------------------------------------------------
    -----
    Plan hash value: 2946813851
    ------------------------------------------------------------------------------
    ----------------------------
    ```

```
| Id  | Operation                     | Name          | Rows  | Bytes |
TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
-----------------------------
|   0 | SELECT STATEMENT              |               |     4 |   104
|       |   12083 (2)| 00:00:01 |
|*  1 |  COUNT STOPKEY                |               |       |
|       |            |          |
|   2 |   VIEW                        |               |  5014 |
127K|       |   12083 (2)| 00:00:01 |
|*  3 |    SORT ORDER BY STOPKEY      |               |  5014 |
19M|    39M|   12083 (2)| 00:00:01 |
|   4 |     TABLE ACCESS BY INDEX ROWID| DOC_CHUNKS    |  5014 |
19M|       |    1    (0)| 00:00:01 |
|   5 |      VECTOR INDEX HNSW SCAN    | DOCS_HNSW_IDX |  5014 |
19M|       |    1    (0)| 00:00:01 |
-------------------------------------------------------------------------------
-----------------------------
```

15. Determine your vector index performance for your approximate similarity searches.

    The index accuracy reporting feature allows you to determine the accuracy of your vector indexes. After a vector index is created, you may be interested to know how accurate your approximate vector searches are.

    The `DBMS_VECTOR.INDEX_ACCURACY_QUERY` PL/SQL procedure provides an accuracy report for a top-K index search for a specific query vector and a specific target accuracy. In this case you keep using the query vector generated previously. For more information about index accuracy reporting, see Index Accuracy Report.

    ```
    SET SERVEROUTPUT ON
    DECLARE
        report VARCHAR2(128);
    BEGIN
        report := dbms_vector.index_accuracy_query(
            OWNER_NAME => 'VECTOR',
            INDEX_NAME => 'DOCS_HNSW_IDX',
            qv => :query_vector,
            top_K => 10,
            target_accuracy => 90 );
        dbms_output.put_line(report);
    END;
    /
    ```

    The report looks like the following: Accuracy achieved (100%) is 10% higher than the Target Accuracy requested (90%).

16. You are now going to create a new table that will be used to run hybrid searches.

    ```
    DROP TABLE documentation_tab2 PURGE;

    CREATE TABLE documentation_tab2 (id NUMBER, file_name VARCHAR2(200));

    INSERT INTO documentation_tab2 VALUES(1, 'json-relational-duality-
    developers-guide.pdf');

    INSERT INTO documentation_tab2 VALUES(2, 'oracle-database-23ai-new-
    ```

```
features-guide.pdf');

COMMIT;
```

17. To create a hybrid vector index, you need to specify your data store where your files reside. Here we are using the `VEC_DUMP` directory where the PDF files are stored:

```
BEGIN
  ctx_ddl.create_preference('DS', 'DIRECTORY_DATASTORE');
  ctx_ddl.set_attribute('DS', 'DIRECTORY', 'VEC_DUMP');
END;
/
```

18. You can now create a hybrid vector index on the `documentation_tab2` table. Here, you indicate to the hybrid vector index where to find the files that need to be indexed and the types that will be filtered, as well as which embedding model to use and which type of vector index. For more information, see Understand Hybrid Vector Indexes.

```
CREATE HYBRID VECTOR INDEX my_hybrid_vector_idx ON
documentation_tab2(file_name)
  PARAMETERS ('
                DATASTORE      DS
                FILTER         CTXSYS.AUTO_FILTER
                MODEL          doc_model
                VECTOR_IDXTYPE ivf
');
```

19. Once the hybrid vector index has been created, check the following internal tables that were created:

```
SELECT COUNT(*) FROM DR$my_hybrid_vector_idx$VR;
```

Result:

```
   COUNT(*)
___________
       9581
```

or

```
SELECT COUNT(*) FROM my_hybrid_vector_idx$VECTORS;
```

Result:

```
   COUNT(*)
___________
       9581
```

```
DESC my_hybrid_vector_idx$VECTORS
```

Result:

```
Name                    Null?       Type
_______________________ ___________ ___________________
DOC_ROWID               NOT NULL    ROWID
DOC_CHUNK_ID            NOT NULL    NUMBER
DOC_CHUNK_COUNT         NOT NULL    NUMBER
DOC_CHUNK_OFFSET        NOT NULL    NUMBER
DOC_CHUNK_LENGTH        NOT NULL    NUMBER
DOC_CHUNK_TEXT                      VARCHAR2(4000)
DOC_CHUNK_EMBEDDING                 VECTOR
```

```
SELECT COUNT(*) FROM DR$MY_HYBRID_VECTOR_IDX$I;
```

Result:

```
   COUNT(*)
___________
       4927
```

20. You can run your first hybrid search by specifying the following parameters:
    - The hybrid vector index name
    - The search scorer you want to use (this scoring function is used after merging results from keyword search and similarity search)
    - The fusion function to use to merge the results from both searches
    - The search text you want for your similarity search
    - The search mode to use to produce the vector search results
    - The aggregation function to use to calculate the vector score for each document identified by your similarity search
    - The score weight for your vector score
    - The `CONTAINS` string for your keyword search
    - The score weight for your keyword search
    - The returned max values you want to see
    - The maximum number of documents and chunks you want to see in the result

    For complete `DBMS_HYBRID_VECTOR.SEARCH` syntax information, see SEARCH.

```
SET LONG 10000

SELECT json_serialize(
    DBMS_HYBRID_VECTOR.SEARCH(
      JSON(
           '{
              "hybrid_index_name" : "my_hybrid_vector_idx",
              "search_scorer"     : "rsf",
              "search_fusion"     : "UNION",
              "vector":
                        {
```

```
                          "search_text"   : How to insert data in json
format to a table?",
                          "search_mode"   : "DOCUMENT",
                          "aggregator"    : "MAX",
                          "score_weight"  : 1,
                         },
              "text":
                         {
                          "contains"      : "data AND json",
                          "score_weight"  : 1,
                         },
              "return":
                         {
                          "values"        : [ "rowid", "score",
"vector_score", "text_score" ],
                          "topN"          : 10
                         }
            }'
      )
   ) RETURNING CLOB pretty);
```

Result:

```
JSON_SERIALIZE(DBMS_HYBRID_VECTOR.SEARCH(JSON('{"HYBRID_INDEX_NAME":"MY_HYB
RID_VECTOR_IDX","SEARCH_SCORER":"RSF","SEARCH_FUSION":"UNION","VECTOR":
{"SEARCH_TEXT":"HOWTOSEARCHUSINGVECTORINDEXES?","SEARCH_MODE":"DOCUMENT","A
GGREGATOR":"MAX","SCORE_WEIGHT
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
_____________________________
[
  {
    "rowid" : "AAASd4AALAAAFjYAAB",
    "score" : 88.77,
    "vector_score" : 77.53,
    "text_score" : 100
  }
]
```

> **Note:**
>
> Before executing the following statement, replace the rowid value that you got from running the previous statement.

```
SELECT file_name FROM documentation_tab2 WHERE rowid='AAASd4AALAAAFjYAAB';
```

Result:

```
FILE_NAME

_____________________________________________
json-relational-duality-developers-guide.pdf
```

> **Note:**
>
> In this step, a lot of parameters were explicitly indicated. However, you can stick to the defaults and run the following statement instead.

```
SELECT json_serialize(
     DBMS_HYBRID_VECTOR.SEARCH(
        JSON(
             '{
               "hybrid_index_name" : "my_hybrid_vector_idx",
               "search_text"       : "How to insert data in json format to
a table?"
             }'
            )
    ) RETURNING CLOB pretty);
```

21. Hybrid vector index maintenance is done automatically in the background at two levels. Every 3 seconds, an index synchronization is run to maintain the index with ongoing DMLs. The index maintenance can take much more time depending on the number of DMLs and files added. So, you may not see your modifications right away as you will experience in this scenario. A scheduler job is also run every night at midnight to optimize the index. As you run DMLs on your index, the index gets fragmented. The optimize job defragments your index for better query performance.

    To experiment with hybrid vector index maintenance, insert a new row in your base table and run the same hybrid search query as you did in the previous step:

```
INSERT INTO documentation_tab2 VALUES(3, 'json-relational-duality-
developers-guide.pdf');

COMMIT;


SELECT json_serialize(
    DBMS_HYBRID_VECTOR.SEARCH(
      JSON(
           '{
              "hybrid_index_name" : "my_hybrid_vector_idx",
              "search_scorer"     : "rsf",
              "search_fusion"     : "UNION",
              "vector":
                       {
                         "search_text"   : "How to insert data in json
format to a table?",
                         "search_mode"   : "DOCUMENT",
                         "aggregator"    : "MAX",
                         "score_weight"  : 1,
```

```
                        },
                "text":
                        {
                         "contains"      : "data AND json",
                         "score_weight"  : 1,
                        },
                "return":
                        {
                          "values"        : [ "rowid", "score",
"vector_score", "text_score" ],
                          "topN"          : 10
                        }
             }'
      )
   ) RETURNING CLOB pretty);
```

You should observe no changes in the result:

```
JSON_SERIALIZE(DBMS_HYBRID_VECTOR.SEARCH(JSON('{"HYBRID_INDEX_NAME":"MY_HYB
RID_VECTOR_IDX","SEARCH_SCORER":"RSF","SEARCH_FUSION":"UNION","VECTOR":
{"SEARCH_TEXT":"HOWTOSEARCHUSINGVECTORINDEXES?","SEARCH_MODE":"DOCUMENT","A
GGREGATOR":"MAX","SCORE_WEIGHT
_____
_____
_____
_____
[
  {
    "rowid" : "AAASd4AALAAAFjYAAB",
    "score" : 88.77,
    "vector_score" : 77.53,
    "text_score" : 100
  }
]
```

22. Wait several minutes and rerun your hybrid search.

```
SELECT json_serialize(
    DBMS_HYBRID_VECTOR.SEARCH(
      JSON(
           '{
              "hybrid_index_name" : "my_hybrid_vector_idx",
              "search_scorer"     : "rsf",
              "search_fusion"     : "UNION",
              "vector":
                        {
                          "search_text"   : "How to insert data in json
format to a table?",
                          "search_mode"   : "DOCUMENT",
                          "aggregator"    : "MAX",
                          "score_weight"  : 1,
                        },
              "text":
                        {
                         "contains"      : "data AND json",
```

```
                          "score_weight"  : 1,
                         },
               "return":
                         {
                           "values"        : [ "rowid", "score",
"vector_score", "text_score" ],
                           "topN"          : 10
                         }
            }'
      )
   ) RETURNING CLOB pretty);
```

By this point, the result has been updated:

```
JSON_SERIALIZE(DBMS_HYBRID_VECTOR.SEARCH(JSON('{"HYBRID_INDEX_NAME":"MY_HYB
RID_VECTOR_IDX","SEARCH_SCORER":"RSF","SEARCH_FUSION":"UNION","VECTOR":
{"SEARCH_TEXT":"HOWTOSEARCHUSINGVECTORINDEXES?","SEARCH_MODE":"DOCUMENT","A
GGREGATOR":"MAX","SCORE_WEIGHT
_____
_____
_____
_____
[
  {
    "rowid" : "AAASd4AALAAAFjYAAB",
    "score" : 88.77,
    "vector_score" : 77.53,
    "text_score" : 100
  },
  {
    "rowid" : "AAASd4AALAAAFjYAAC",
    "score" : 88.77,
    "vector_score" : 77.53,
    "text_score" : 100
  }
]
```

SQL Quick Start Using a FLOAT32 Vector Generator

A PL/SQL program that creates a vector generator is included along with example queries and results, providing a simple way to get started with Oracle AI Vector Search without a vector embedding model.

The generator is a PL/SQL program that allows you to randomly generate vectors with a specified number of dimensions and clusters. Each dimension value is generated using specified minimum and maximum possible values. The output of the generation process is the population of a table called `genvec` that you can then use, for example, to experiment with similarity searches.

The following instructions assume you already have access to a database account with sufficient privileges (minimally the `DB_DEVELOPER_ROLE` role).

> **Note:**
>
> Do not use the vector generator on production databases. The program is made available for testing and demo purposes.

1.

2. Create the `genvec` table.

```
DROP TABLE genvec PURGE;

CREATE TABLE genvec (
  id number,           -- id of the generated vector
  v VECTOR,            -- generated vector
  name VARCHAR2(500),  -- name for the generated vector: C1 to Cn are
centroids, Cx_y is vector number y in cluster number x
  nv VECTOR,           -- normalized version of the generated vector
  ly number            -- random number you can use to filter out rows in
addition to similarity search on vectors
);
```

3. Create the package `vector_gen_pkg` and its associated package body.

   Here you create the vector generator package:

```
CREATE OR REPLACE PACKAGE vector_gen_pkg AS

  TYPE t_vectors IS TABLE OF vector INDEX BY PLS_INTEGER;

  FUNCTION get_coordinate(
    input_string CLOB,
    i PLS_INTEGER
  ) RETURN NUMBER;

  PROCEDURE generate_vectors(
    num_vectors IN PLS_INTEGER,  -- Number of vectors to generate
    dimensions IN PLS_INTEGER,   -- Number of dimensions of each vector
    num_clusters IN PLS_INTEGER, -- Number of clusters to create
    cluster_spread IN NUMBER,    -- Relative closeness of each vector in
each cluster (using standard deviation)
    min_value IN NUMBER,         -- Minimum value for a vector coordinate
    max_value IN NUMBER          -- Maximum value for a vector coordinate
  );

END vector_gen_pkg;
/
```

   And the package body:

```
CREATE OR REPLACE PACKAGE BODY vector_gen_pkg AS

  ----------------------------------------
  ----------------------------------------
  ---- V E C T O R    G E N E R A T O R -----
```

```
  ------------------------------------------
  ------------------------------------------
  -- Version 1.0                   ---------
  ------------------------------------------
  ------------------------------------------
  ---- DO NOT USE ON PRODUCTION DATABASES ---
  ---- ONLY FOR TESTING AND DEMO PURPOSES ---
  ------------------------------------------


  FUNCTION get_coordinate(
    input_string CLOB,
    i PLS_INTEGER
  ) RETURN NUMBER IS
    start_pos NUMBER;
    end_pos NUMBER;
    comma_pos NUMBER;
    coord VARCHAR2(100);
    comma_count NUMBER := 0;
    commas NUMBER;
    working_string CLOB;
  BEGIN
    -- Remove leading and trailing brackets
    working_string := input_string;
    working_string := TRIM(BOTH '[]' FROM working_string);
    commas := LENGTH(working_string) - LENGTH(REPLACE(working_string, ',',
''));

    -- Initialize positions
    start_pos := 1;
    end_pos := INSTR(working_string, ',', start_pos);

    IF i<=0 OR i>commas+1 THEN RETURN NULL;
    END IF;

    -- Loop through the string to find the i-th coordinate
    LOOP
      IF comma_count + 1 = i THEN
        IF end_pos = 0 THEN
          -- If there's no more comma, the coordinate is the rest of the
string
          coord := SUBSTR(working_string, start_pos);
        ELSE
          coord := SUBSTR(working_string, start_pos, end_pos - start_pos);
        END IF;
        RETURN coord;
      END IF;

      -- Move to the next coordinate
      comma_count := comma_count + 1;
      start_pos := end_pos + 1;
      end_pos := INSTR(working_string, ',', start_pos);

      -- Exit loop if no more coordinates
      EXIT WHEN start_pos > LENGTH(working_string);
    END LOOP;
```

```
    -- If the function hasn't returned yet, the index was out of bounds
    RETURN NULL;

  END;



  PROCEDURE generate_random_vector(
    dimensions IN PLS_INTEGER,
    min_value IN NUMBER,
    max_value IN NUMBER,
    vec OUT vector
    ) IS
    e CLOB;
  BEGIN
    e := '[';
    FOR i IN 1..dimensions-1 LOOP
      e := e || DBMS_RANDOM.VALUE(min_value, max_value) ||',';
    END LOOP;
    e := e || DBMS_RANDOM.VALUE(min_value, max_value) ||']';
    vec := VECTOR(e);
  END generate_random_vector;



  PROCEDURE generate_clustered_vector(
    centroid IN vector,
    cluster_spread IN NUMBER,
    vec OUT vector
  ) IS
    e CLOB;
    d number;
    BEGIN
      d := VECTOR_DIMENSION_COUNT(centroid);
      e := '[';
      FOR i IN 1 .. d-1 LOOP
        e := e || (get_coordinate(to_clob(centroid),i) +
(DBMS_RANDOM.NORMAL * cluster_spread)) ||',';
      END LOOP;
      e := e ||
(get_coordinate(to_clob(centroid),VECTOR_DIMENSION_COUNT(centroid)) +
(DBMS_RANDOM.NORMAL * cluster_spread)) || ']';
      vec := VECTOR(e);
  END generate_clustered_vector;



  FUNCTION normalize_vector(vec IN vector) RETURN vector IS
    e CLOB;
    v CLOB;
    n number;
    d number;
  BEGIN
    n := VECTOR_NORM(vec);
    v := to_clob(vec);
```

```
    d := VECTOR_DIMENSION_COUNT(vec);
    e := '[';
    FOR i IN 1 .. d-1 LOOP
      e := e || (get_coordinate(v,i)/n) ||',';
    END LOOP;
    e := e || (get_coordinate(v,d)/n) || ']';
    RETURN VECTOR(e);
  END normalize_vector;



  PROCEDURE generate_vectors(
    num_vectors IN PLS_INTEGER,  -- Must be 1 or above
    dimensions IN PLS_INTEGER,   -- Must be above 1 but less than 500
    num_clusters IN PLS_INTEGER, -- Must be 1 or above
    cluster_spread IN NUMBER,    -- Must be grather than 0
    min_value IN NUMBER,
    max_value IN NUMBER
  ) IS
    centroids t_vectors;
    vectors_per_cluster PLS_INTEGER;
    remaining_vectors PLS_INTEGER;
    vec vector;
    idx PLS_INTEGER := 1;
    max_id NUMBER;
    working_vector VECTOR;

  BEGIN
    IF (num_vectors) <=0 OR (num_clusters < 1) OR (num_vectors <
num_clusters) OR (dimensions <= 0) OR (dimensions > 500) OR
(cluster_spread <= 0) OR (min_value >= max_value) THEN RETURN;
    END IF;

    SELECT MAX(id) INTO max_id FROM genvec;
    IF max_id IS NULL THEN max_id := 0;
    END IF;

    -- Generate cluster centroids
    FOR i IN 1..num_clusters LOOP

      generate_random_vector(dimensions, min_value, max_value,
centroids(i));
      working_vector := normalize_vector(centroids(i));
      INSERT INTO genvec VALUES (max_id + idx, centroids(i), 'C'||i,
working_vector, DBMS_RANDOM.VALUE(3,600000000));
      idx := idx + 1;

    END LOOP;

    -- Calculate vectors per cluster
    vectors_per_cluster := TRUNC(num_vectors / num_clusters);
    remaining_vectors := num_vectors MOD num_clusters;

    -- Generate vectors for each cluster
    IF vectors_per_cluster > 1 THEN
      FOR i IN 1..num_clusters LOOP
```

```
        FOR j IN 1..(vectors_per_cluster - 1) LOOP
          generate_clustered_vector(centroids(i), cluster_spread, vec);
          working_vector := normalize_vector(vec);
          INSERT INTO genvec VALUES (max_id + idx, vec, 'C'||i||'-'||j,
working_vector, DBMS_RANDOM.VALUE(3,600000000));
          idx := idx + 1;
        END LOOP;
      END LOOP;
    END IF;

    -- Handle remaining vectors: all associated with cluster 1
    IF remaining_vectors > 0 THEN
      FOR j IN 1..remaining_vectors LOOP
        generate_clustered_vector(centroids(1), cluster_spread, vec);
        working_vector := normalize_vector(vec);
        INSERT INTO genvec VALUES (max_id + idx, vec, 'C1-'||idx,
working_vector, DBMS_RANDOM.VALUE(3,600000000));
        idx := idx + 1;
      END LOOP;
    END IF;
    COMMIT;
  END generate_vectors;

END vector_gen_pkg;
/
```

4. After you have your vector generator set up, you can run this and the following steps to understand how to use it.

   Run the `generate_vectors` procedure of the `vector_gen_pkg` package with sample values:

```
BEGIN
  vector_gen_pkg.generate_vectors(
    num_vectors => 100,    -- Number of vectors to generate. Must be 1 or
above
    dimensions => 3,       -- Number of dimensions of each vector. Must be
above 1 but less than 500
    num_clusters => 6,     -- Number of clusters to create. Must be 1 or
above
    cluster_spread => 1,   -- Relative closeness of each vector in each
cluster (using standard deviation). Must be grather than 0
    min_value => 0,        -- Minimum value for a vector coordinate
    max_value => 100       -- Maximum value for a vector coordinate. Min
value must be smaller than max value
  );
END;
/
```

5. Run a `SELECT` statement to view the newly generated vectors.

```
SELECT name, v FROM genvec;
```

Example output:

```
NAME     V

________ ____________________________________________________________
C1       [6.35792809E+001,5.28954163E+001,5.16500435E+001]
C2       [5.67991257E+001,5.00640755E+001,2.3642437E+001]
C3       [2.42510891E+001,5.36970367E+001,6.88145638E+000]
C4       [8.13146515E+001,2.88190498E+001,4.09245186E+001]
C5       [6.70646744E+001,2.53395119E+001,6.14522667E+001]
C6       [5.60192604E+001,8.31662598E+001,2.93592377E+001]
C1-1     [6.4469986E+001,5.25044632E+001,5.22250557E+001]
C1-2     [6.31295433E+001,5.21443062E+001,5.02242126E+001]
C1-3     [6.25154915E+001,5.25730362E+001,5.2617794E+001]
C1-4     [6.3491375E+001,5.21440697E+001,5.06722069E+001]
C1-5     [6.21516266E+001,5.32161064E+001,5.23233032E+001]
C1-6     [6.2913269E+001,5.16970291E+001,5.16683655E+001]
C1-7     [6.22267456E+001,5.40408363E+001,5.0272541E+001]
C1-8     [6.1414093E+001,5.28870888E+001,5.27458E+001]

NAME      V

_________ ____________________________________________________________
C1-9      [6.29652252E+001,5.32767754E+001,5.27030106E+001]
C1-10     [6.35940704E+001,5.27265244E+001,5.23180656E+001]
C1-11     [6.34133224E+001,5.39401283E+001,5.29368248E+001]
C1-12     [6.18856697E+001,5.31113129E+001,5.18861504E+001]
C1-13     [6.32378883E+001,5.30308647E+001,5.04571724E+001]
C1-14     [6.18148689E+001,5.33705482E+001,5.29123802E+001]
C1-15     [6.43224258E+001,5.23124084E+001,5.21299057E+001]
C2-1      [5.59053535E+001,5.20054626E+001,2.28595486E+001]
C2-2      [5.71644516E+001,5.13243408E+001,2.31167526E+001]
C2-3      [5.66626244E+001,5.00615959E+001,2.27138176E+001]
C2-4      [5.73383865E+001,5.04509125E+001,2.36539135E+001]
C2-5      [5.6621357E+001,5.01576767E+001,2.38867531E+001]
C2-6      [5.59768562E+001,5.17590942E+001,2.49088764E+001]
C2-7      [5.64437904E+001,4.71531525E+001,2.23245487E+001]

NAME      V

_________ ____________________________________________________________
C2-8      [5.81449051E+001,5.09049644E+001,2.29072056E+001]
C2-9      [5.37190132E+001,4.87386665E+001,2.28188381E+001]
C2-10     [5.77416382E+001,4.93461685E+001,2.32014389E+001]
C2-11     [5.68353958E+001,5.11093979E+001,2.43693123E+001]
C2-12     [5.79631157E+001,5.0297657E+001,2.28039799E+001]
C2-13     [5.57930183E+001,5.11965866E+001,2.35887661E+001]
C2-14     [5.57345848E+001,5.03228951E+001,2.30780907E+001]
C2-15     [5.69435997E+001,4.8590435E+001,2.58747597E+001]
C3-1      [2.40239315E+001,5.2352993E+001,5.63517284E+000]
C3-2      [2.39717846E+001,5.30635986E+001,5.86633539E+000]
C3-3      [2.70314407E+001,5.48788643E+001,7.96345377E+000]
C3-4      [2.39875908E+001,5.39634552E+001,5.87654877E+000]
C3-5      [2.47772026E+001,5.2187336E+001,6.83652115E+000]
C3-6      [2.32920208E+001,5.41494293E+001,6.40737772E+000]

NAME      V

_________ ____________________________________________________________
C3-7      [2.46129742E+001,5.32308769E+001,6.29999685E+000]
```

```
C3-8     [2.51000671E+001,5.33271561E+001,8.86797047E+000]
C3-9     [2.4337059E+001,5.26281281E+001,6.9616766E+000]
C3-10    [2.39770508E+001,5.42386856E+001,5.63018417E+000]
C3-11    [2.59837551E+001,5.34013176E+001,6.97773361E+000]
C3-12    [2.40400314E+001,5.25649719E+001,7.2636981E+000]
C3-13    [2.13184013E+001,5.28633308E+001,8.3834734E+000]
C3-14    [2.50075855E+001,5.21548729E+001,6.88196087E+000]
C3-15    [2.53695087E+001,5.60495186E+001,6.76059389E+000]
C4-1     [8.28819885E+001,2.95163822E+001,4.03809738E+001]
C4-2     [8.18269348E+001,2.95735188E+001,3.99435768E+001]
C4-3     [8.2709259E+001,2.90755043E+001,4.07345886E+001]
C4-4     [8.18622665E+001,2.88013916E+001,4.1822567E+001]
C4-5     [7.99165421E+001,2.89941139E+001,4.09653854E+001]

NAME     V
_______  ____________________________________________________
C4-6     [8.12936707E+001,2.98655643E+001,4.00380211E+001]
C4-7     [8.21705704E+001,2.90163479E+001,3.94858704E+001]
C4-8     [8.20081329E+001,2.89751148E+001,4.1045887E+001]
C4-9     [8.25486298E+001,2.84143009E+001,4.15654945E+001]
C4-10    [8.22034149E+001,2.92223415E+001,4.20033302E+001]
C4-11    [8.2048996E+001,2.98751106E+001,4.09612732E+001]
C4-12    [8.09316025E+001,2.7799057E+001,4.12611198E+001]
C4-13    [8.04624023E+001,2.88711109E+001,4.07331085E+001]
C4-14    [8.13773575E+001,2.97510109E+001,4.09169846E+001]
C4-15    [8.35310364E+001,2.971031E+001,4.16878052E+001]
C5-1     [6.87114258E+001,2.53504581E+001,6.11055298E+001]
C5-2     [6.73569031E+001,2.35163498E+001,6.01617432E+001]
C5-3     [6.78224869E+001,2.61236534E+001,6.0729248E+001]
C5-4     [6.76432266E+001,2.56426888E+001,6.35400085E+001]

NAME     V
_______  ____________________________________________________
C5-5     [6.75377045E+001,2.60873699E+001,6.35584145E+001]
C5-6     [6.84944687E+001,2.51576214E+001,6.24934502E+001]
C5-7     [6.79246216E+001,2.53722992E+001,6.32098122E+001]
C5-8     [6.84075165E+001,2.63778133E+001,6.10950584E+001]
C5-9     [6.73214798E+001,2.70551453E+001,6.27835197E+001]
C5-10    [6.50006485E+001,2.67408028E+001,6.07828026E+001]
C5-11    [6.68869705E+001,2.3982399E+001,6.13440819E+001]
C5-12    [6.55524521E+001,2.42231808E+001,6.07235756E+001]
C5-13    [6.72140808E+001,2.42842178E+001,6.21546478E+001]
C5-14    [6.89587936E+001,2.67715569E+001,6.08621559E+001]
C5-15    [6.68405685E+001,2.44039059E+001,6.12652893E+001]
C6-1     [5.4925251E+001,8.28179474E+001,3.0869236E+001]
C6-2     [5.52922363E+001,8.23375549E+001,2.94804363E+001]
C6-3     [5.60466652E+001,8.18454132E+001,2.99774895E+001]

NAME     V
_______  ____________________________________________________
C6-4     [5.74460373E+001,8.26830368E+001,2.86887722E+001]
C6-5     [5.57439041E+001,8.14622726E+001,2.94924259E+001]
C6-6     [5.4913372E+001,8.48766251E+001,2.92711105E+001]
C6-7     [5.66876144E+001,8.25907898E+001,2.84199276E+001]
C6-8     [5.6253479E+001,8.3280838E+001,2.69524212E+001]
C6-9     [5.50792351E+001,8.37676392E+001,3.08755417E+001]
```

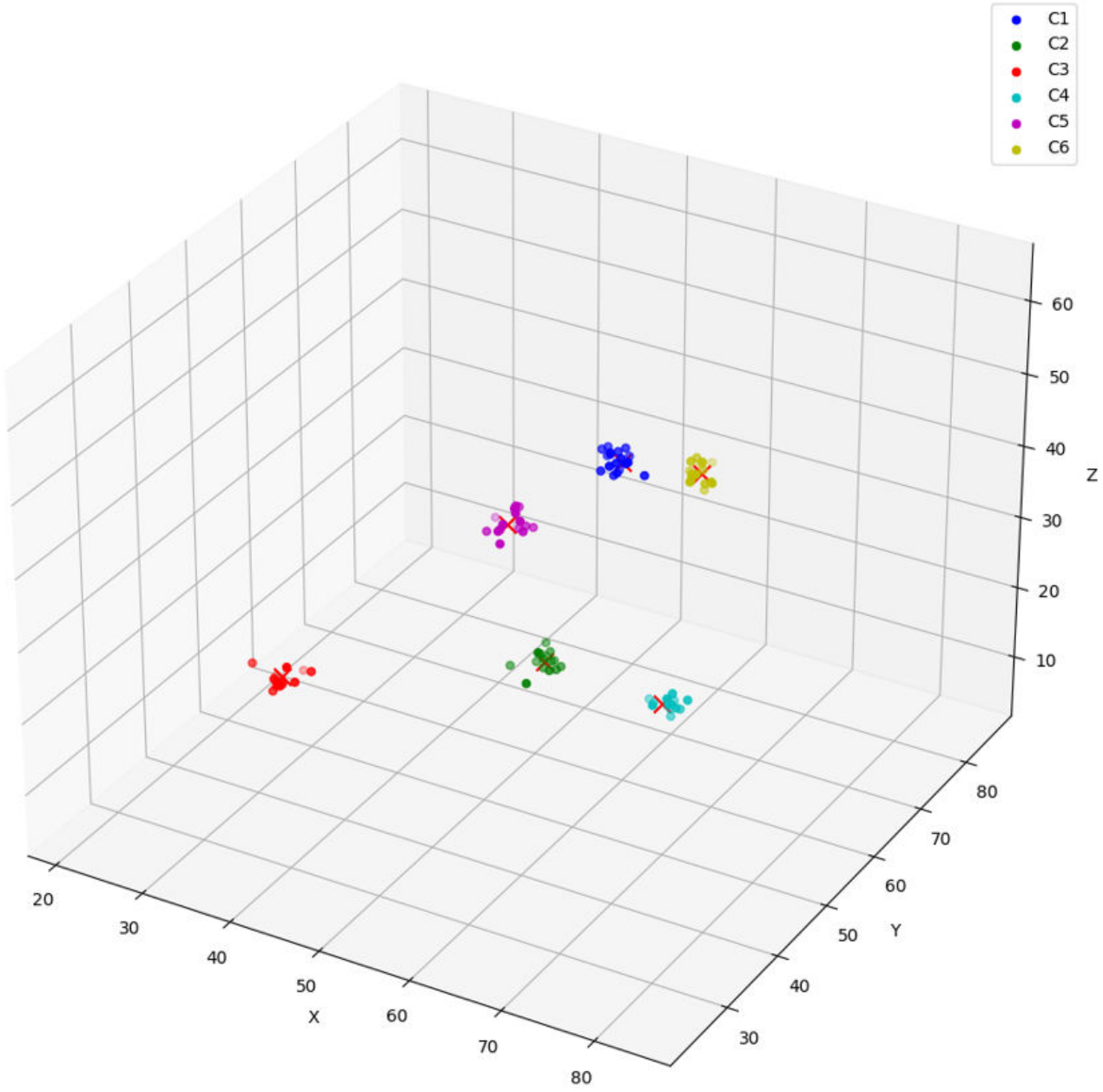
```
C6-10    [5.57719955E+001,8.11036758E+001,2.92569256E+001]
C6-11    [5.60834808E+001,8.3103096E+001,3.09748001E+001]
C6-12    [5.58962059E+001,8.3612648E+001,2.95026093E+001]
C6-13    [5.73348083E+001,8.26950226E+001,2.88242455E+001]
C6-14    [5.45099411E+001,8.33315659E+001,2.90559101E+001]
C6-15    [5.57930641E+001,8.5720871E+001,2.92863998E+001]
C1-97    [6.3716671E+001,5.41518326E+001,5.18371048E+001]
C1-98    [6.58774261E+001,5.32223206E+001,5.05089798E+001]

NAME      V
_________ ____________________________________________________
C1-99     [6.31867676E+001,5.25712204E+001,5.16621819E+001]
C1-100    [6.20503845E+001,5.15550919E+001,5.08155479E+001]

100 rows selected.
```

To visualize the resulting vectors in space, consider the following graph:

Figure 3-1 Vector Clusters



6. Run a similarity search on the generated vectors in the `genvec` table.

   First define a variable called `cluster_number` to be used to form the name of the vector in the query.

   ```
   DEFINE cluster_number = '&clusterid'
   ```

   You will be prompted to enter a value for `clusterid`. In this example, we use `5`:

   ```
   Enter value for clusterid: 5
   ```

Run the following query to perform a similarity search on the generated vectors:

```
SELECT name
  FROM genvec
  ORDER BY VECTOR_DISTANCE(v,(SELECT v FROM genvec WHERE
name='C'||'&cluster_number'),EUCLIDEAN)
  FETCH EXACT FIRST 20 ROWS ONLY;
```

Example output:

```
NAME

________
C5
C5-15
C5-13
C5-3
C5-11
C5-1
C5-8
C5-6
C5-7
C5-12
C5-9
C5-4
C5-2
C5-5

NAME

________
C5-14
C5-10
C4-5
C4-12
C4-4
C4-13

20 rows selected.
```

7. Here is another example of running a similarity search on the generated vectors, this time using the Cosine distance metric.

```
SELECT name
  FROM genvec
  ORDER BY VECTOR_DISTANCE(v,(SELECT v FROM genvec WHERE
name='C'||'&cluster_number'),COSINE)
  FETCH EXACT FIRST 20 ROWS ONLY;
```

Example output:

```
NAME

________
C5
C5-6
C5-12
```

```
C5-15
C5-7
C5-4
C5-5
C5-13
C5-11
C5-3
C5-1
C5-8
C5-9
C5-2

NAME

________
C5-14
C5-10
C4-5
C4-4
C4-10
C4-12

20 rows selected.
```

8. Create a variable called `query_vector` and then use `SELECT INTO` to store a vector value in the variable.

```
VARIABLE query_vector CLOB

BEGIN
  SELECT v INTO :query_vector
  FROM genvec
  WHERE name='C'||'&cluster_number';
END;
/

PRINT query_vector;
```

Example output:

```
QUERY_VECTOR
--------------------------------------------------------------------------------
[6.70646744E+001,2.53395119E+001,6.14522667E+001]
```

9. Create an explain plan for a similarity search using the query vector created in the previous step.

```
EXPLAIN PLAN FOR
  SELECT name
  FROM genvec
  ORDER BY VECTOR_DISTANCE(v, :query_vector, EUCLIDEAN)
  FETCH EXACT FIRST 20 ROWS ONLY;

SELECT plan_table_output
  FROM table(dbms_xplan.display('plan_table',null,'all'));
```

Example output:

```
PLAN_TABLE_OUTPUT
_____________________________________________________________________________
__________
Plan hash value: 1549136425

-------------------------------------------------------------------------------
-------
| Id  | Operation               | Name    | Rows  | Bytes | Cost (%CPU)|
Time     |
-------------------------------------------------------------------------------
-------
|   0 | SELECT STATEMENT        |         |    20 |  5040 |     4  (25)|
00:00:01 |
|*  1 |  COUNT STOPKEY          |         |       |       |
|          |
|   2 |   VIEW                  |         |   100 | 25200 |     4  (25)|
00:00:01 |
|*  3 |    SORT ORDER BY STOPKEY|         |   100 | 25800 |     4  (25)|
00:00:01 |
|   4 |     TABLE ACCESS FULL   | GENVEC  |   100 | 25800 |     3   (0)|
00:00:01 |
-------------------------------------------------------------------------------
-------

Query Block Name / Object Alias (identified by operation id):
-------------------------------------------------------------

PLAN_TABLE_OUTPUT
_____________________________________________________________

   1 - SEL$2
   2 - SEL$1 / "from$_subquery$_002"@"SEL$2"
   3 - SEL$1
   4 - SEL$1 / "GENVEC"@"SEL$1"

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter(ROWNUM<=20)
   3 - filter(ROWNUM<=20)

Column Projection Information (identified by operation id):
-----------------------------------------------------------

PLAN_TABLE_OUTPUT
_____________________________________________________________________________

   1 - "from$_subquery$_002"."NAME"[VARCHAR2,500]
   2 - "from$_subquery$_002"."NAME"[VARCHAR2,500]
   3 - (#keys=1) VECTOR_DISTANCE("V" /*+ LOB_BY_VALUE */ ,
       VECTOR(:QUERY_VECTOR, *, * /*+  USEBLOBPCW_QVCGMD */ ),
       EUCLIDEAN)[BINARY_DOUBLE,8], "NAME"[VARCHAR2,500]
   4 - "NAME"[VARCHAR2,500], VECTOR_DISTANCE("V" /*+ LOB_BY_VALUE */ ,
       VECTOR(:QUERY_VECTOR, *, * /*+  USEBLOBPCW_QVCGMD */ ),
```

```
       EUCLIDEAN)[BINARY_DOUBLE,8]

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)

41 rows selected.
```

10. Create an Hierarchical Navigable Small World (HNSW) index.

```
CREATE VECTOR INDEX genvec_hnsw_idx ON genvec(v)
  ORGANIZATION INMEMORY NEIGHBOR GRAPH
  DISTANCE EUCLIDEAN
  WITH TARGET ACCURACY 95;

SELECT INDEX_NAME, INDEX_TYPE, INDEX_SUBTYPE FROM USER_INDEXES;
```

Example output:

```
INDEX_NAME                     INDEX_TYPE     INDEX_SUBTYPE
_____________________________ _____________ _____________________________________
DM$CEDOC_MODEL                 NORMAL
SYS_IL0000073592C00002$$       LOB
GENVEC_HNSW_IDX                VECTOR         INMEMORY_NEIGHBOR_GRAPH_HNSW
SYS_C008694                    NORMAL

4 rows selected.
```

11. Query information about the HNSW index from the `VECSYS.VECTOR$INDEX` view.

```
SELECT JSON_SERIALIZE(IDX_PARAMS RETURNING VARCHAR2 PRETTY)
  FROM VECSYS.VECTOR$INDEX
  WHERE IDX_NAME = 'GENVEC_HNSW_IDX';
```

Example output:

```
JSON_SERIALIZE(IDX_PARAMSRETURNINGVARCHAR2PRETTY)
_____________________________________________________________________________
_____________________________________________________________________________
_____________________________________________________________________________
_________________________
{
  "type" : "HNSW",
  "num_neighbors" : 32,
  "efConstruction" : 300,
  "distance" : "EUCLIDEAN",
  "accuracy" : 95,
  "vector_type" : "FLOAT32",
  "vector_dimension" : 3,
  "degree_of_parallelism" : 1,
  "pdb_id" : 3,
  "indexed_col" : "V"
}
```

For information about the columns of the `VECSYS.VECTOR$INDEX` view, see VECSYS.VECTOR$INDEX.

12. With the HNSW index created, create another explain plan for a similarity search on the `genvec` table.

```
EXPLAIN PLAN FOR
  SELECT name
  FROM genvec
  ORDER BY VECTOR_DISTANCE(v, :query_vector, EUCLIDEAN)
  FETCH APPROX FIRST 20 rows only;

SELECT plan_table_output
FROM table(dbms_xplan.display('plan_table',null,'all'));
```

Example output:

```
PLAN_TABLE_OUTPUT
_____________________________________________________________________________
_____________________________
Plan hash value: 1202819565

-------------------------------------------------------------------------------
-------------------------------
| Id  | Operation                       | Name            | Rows  | Bytes |
TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
-------------------------------
|   0 | SELECT STATEMENT                |                 |    20 |  5040
|       |   165   (2)| 00:00:01 |
|*  1 |  COUNT STOPKEY                  |                 |       |
|       |            |          |
|   2 |   VIEW                          |                 |   100 | 25200
|       |   165   (2)| 00:00:01 |
|*  3 |    SORT ORDER BY STOPKEY        |                 |   100 |
425K|   808K|   165   (2)| 00:00:01 |
|   4 |     TABLE ACCESS BY INDEX ROWID| GENVEC          |   100 |
425K|       |     1   (0)| 00:00:01 |
|   5 |      VECTOR INDEX HNSW SCAN     | GENVEC_HNSW_IDX |   100 |
425K|       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------
-------------------------------

Query Block Name / Object Alias (identified by operation id):

PLAN_TABLE_OUTPUT
_____________________________________________________________
-------------------------------------------------------------


   1 - SEL$2
   2 - SEL$1 / "from$_subquery$_002"@"SEL$2"
   3 - SEL$1
   4 - SEL$1 / "GENVEC"@"SEL$1"
   5 - SEL$1 / "GENVEC"@"SEL$1"

Predicate Information (identified by operation id):
```

```
-------------------------------------------------

   1 - filter(ROWNUM<=20)
   3 - filter(ROWNUM<=20)


PLAN_TABLE_OUTPUT

_____________________________
Column Projection Information (identified by operation id):
-----------------------------------------------------------

   1 - "from$_subquery$_002"."NAME"[VARCHAR2,500]
   2 - "from$_subquery$_002"."NAME"[VARCHAR2,500]
   3 - (#keys=1) VECTOR_DISTANCE("V" /*+ LOB_BY_VALUE */ ,
VECTOR(:QUERY_VECTOR, *, * /*+
       USEBLOBPCW_QVCGMD */ ), EUCLIDEAN)[8], "NAME"[VARCHAR2,500]
   4 - "NAME"[VARCHAR2,500], VECTOR_DISTANCE("V" /*+ LOB_BY_VALUE */ ,
VECTOR(:QUERY_VECTOR, *, *
       /*+  USEBLOBPCW_QVCGMD */ ), EUCLIDEAN)[8]
   5 - "GENVEC".ROWID[ROWID,10], VECTOR_DISTANCE("V" /*+ LOB_BY_VALUE */ ,
VECTOR(:QUERY_VECTOR,
       *, * /*+  USEBLOBPCW_QVCGMD */ ), EUCLIDEAN)[8]

Note
-----

PLAN_TABLE_OUTPUT

_____________________________________________________________
   - dynamic statistics used: dynamic sampling (level=2)

43 rows selected.
```

As you can see, the explain plan now includes information about the HNSW index.

13. Again perform a similarity search on the vectors in the `genvec` table. Note that it is possible for query results to vary based on the indexing technique used. The results included in this scenario are simply an example.

```
SELECT name
  FROM genvec
  ORDER BY vector_distance(v, :query_vector, EUCLIDEAN)
  FETCH APPROX FIRST 20 ROWS ONLY;
```

Example output:

```
NAME

________
C5
C5-15
C5-13
C5-3
C5-11
C5-1
C5-8
```

```
C5-6
C5-7
C5-12
C5-9
C5-4
C5-2
C5-5

NAME

________
C5-14
C5-10
C4-5
C4-12
C4-4
C4-13

20 rows selected.
```

14. Drop the HNSW index and create an Inverted File Flat (IVF) vector index.

```
DROP INDEX genvec_hnsw_idx;

CREATE VECTOR INDEX genvec_ivf_idx ON genvec(v)
  ORGANIZATION NEIGHBOR PARTITIONS
  DISTANCE EUCLIDEAN
  WITH TARGET ACCURACY 95;

SELECT JSON_SERIALIZE(IDX_PARAMS RETURNING VARCHAR2 PRETTY)
  FROM VECSYS.VECTOR$INDEX WHERE IDX_NAME = 'GENVEC_IVF_IDX';
```

Example output:

```
JSON_SERIALIZE(IDX_PARAMSRETURNINGVARCHAR2PRETTY)
_____________________________________________________________________________
_____________________________________________________________________________
_____________________________________________________________________________
_____________________________________________________________________________
_____________________
{
  "target_centroids" : 40,
  "pdb_id" : 3,
  "vector_type" : "FLOAT32",
  "type" : "IVF_FLAT",
  "vector_dimension" : 3,
  "distance" : "EUCLIDEAN",
  "indexed_col" : "V",
  "min_vectors_per_partition" : 10,
  "degree_of_parallelism" : 1,
  "accuracy" : 95,
  "num_centroids" : 24,
  "samples_per_partition" : 256
}
```

15. Again create an explain plan, which now includes information about the newly created IVF index.

```
EXPLAIN PLAN FOR
  SELECT name
  FROM genvec
  ORDER BY VECTOR_DISTANCE(v, :query_vector, EUCLIDEAN)
  FETCH APPROX FIRST 20 ROWS ONLY;

SELECT plan_table_output
  FROM table(dbms_xplan.display('plan_table',null,'all'));
```

Example output:

```
PLAN_TABLE_OUTPUT
_____________________________________________________________________________
_____________________________________________________________________________
____________________
Plan hash value: 2965029064

-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------
| Id  | Operation                           |
Name                                                                | Rows  |
Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------
|   0 | SELECT STATEMENT
|                                                                       |     5
|  1260 |    29  (14)| 00:00:01 |       |       |
|   1 |  VIEW
|                                                                       |     5
|  1260 |    29  (14)| 00:00:01 |       |       |
|   2 |   SORT ORDER BY
|                                                                       |     5
| 21910 |    29  (14)| 00:00:01 |       |       |
|*  3 |    HASH JOIN
|                                                                       |     5
| 21910 |    28  (11)| 00:00:01 |       |       |
|   4 |     VIEW                            |
VW_IVPSR_11E7D7DE                                                     |    20
|   320 |    24   (9)| 00:00:01 |       |       |
|*  5 |      COUNT STOPKEY
|                                                                       |
|       |           |          |       |       |
|   6 |       VIEW                          |
VW_IVPSJ_578B79F1                                                     |    25
|   450 |    24   (9)| 00:00:01 |       |       |
|*  7 |        SORT ORDER BY STOPKEY
|                                                                       |    25
|   550 |    24   (9)| 00:00:01 |       |       |
|*  8 |         HASH JOIN
|                                                                       |    25
```

```
|   550 |    23   (5)| 00:00:01 |       |       |

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------
|   9 |          PART JOIN FILTER CREATE
| :BF0000                                                          |     6
|    18 |     4  (25)| 00:00:01 |       |       |
|  10 |           VIEW                      |
VW_IVCR_B5B87E67                                                   |     6
|    18 |     4  (25)| 00:00:01 |       |       |
|* 11 |            COUNT STOPKEY
|                                                                  |
|       |            |          |       |       |
|  12 |             VIEW                    |
VW_IVCN_9A1D2119                                                   |    24
|   312 |     4  (25)| 00:00:01 |       |       |
|* 13 |              SORT ORDER BY STOPKEY
|                                                                  |    24
|   216 |     4  (25)| 00:00:01 |       |       |
|  14 |               TABLE ACCESS FULL     |
VECTOR$GENVEC_IVF_IDX$87355_87370_0$IVF_FLAT_CENTROIDS             |    24
|   216 |     3   (0)| 00:00:01 |       |       |
|  15 |          PARTITION LIST JOIN-
FILTER|
|   100 |  1900 |     3   (0)| 00:00:01 |:BF0000|:BF0000|
|  16 |           TABLE ACCESS FULL         |
VECTOR$GENVEC_IVF_IDX$87355_87370_0$IVF_FLAT_CENTROID_PARTITIONS |   100
|  1900 |     3   (0)| 00:00:01 |:BF0000|:BF0000|
|  17 |     TABLE ACCESS FULL               |
GENVEC                                                             |   100
|   426K|     3   (0)| 00:00:01 |       |       |
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
------------------

Query Block Name / Object Alias (identified by operation id):
-------------------------------------------------------------


PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
   1 - SEL$94C0F189 / "from$_subquery$_002"@"SEL$2"
   2 - SEL$94C0F189
   4 - SEL$E731354C / "VW_IVPSR_11E7D7DE"@"SEL$1"
   5 - SEL$E731354C
   6 - SEL$0C00A749 / "VW_IVPSJ_578B79F1"@"SEL$E731354C"
   7 - SEL$0C00A749
  10 - SEL$700CE8F1 / "VW_IVCR_B5B87E67"@"SEL$0C00A749"
  11 - SEL$700CE8F1
  12 - SEL$E5326247 / "VW_IVCN_9A1D2119"@"SEL$700CE8F1"
  13 - SEL$E5326247
  14 - SEL$E5326247 / "VTIX_CENTRD"@"SEL$E5326247"
  16 - SEL$0C00A749 / "VTIX_CNPART"@"SEL$0C00A749"
  17 - SEL$94C0F189 / "GENVEC"@"SEL$1"
```

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
---
Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("GENVEC".ROWID="VW_IVPSR_11E7D7DE"."BASE_TABLE_ROWID")
   5 - filter(ROWNUM<=20)
   7 - filter(ROWNUM<=20)
   8 - access("VW_IVCR_B5B87E67"."CENTROID_ID"="VTIX_CNPART"."CENTROID_ID")
  11 - filter(ROWNUM<=6)
  13 - filter(ROWNUM<=6)

Column Projection Information (identified by operation id):
-----------------------------------------------------------

   1 - "from$_subquery$_002"."NAME"[VARCHAR2,500]

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
------------
   2 - (#keys=1) "VEC_DIST"[BINARY_DOUBLE,8], "GENVEC"."NAME"[VARCHAR2,500]
   3 - (#keys=1) "GENVEC"."NAME"[VARCHAR2,500],
"VEC_DIST"[BINARY_DOUBLE,8], "GENVEC"."NAME"[VARCHAR2,500]
   4 - "BASE_TABLE_ROWID"[ROWID,10], "VEC_DIST"[BINARY_DOUBLE,8]
   5 - "VW_IVPSJ_578B79F1"."BASE_TABLE_ROWID"[ROWID,10],
"VW_IVPSJ_578B79F1"."VEC_DIST"[BINARY_DOUBLE,8]
   6 - "VW_IVPSJ_578B79F1"."BASE_TABLE_ROWID"[ROWID,10],
"VW_IVPSJ_578B79F1"."VEC_DIST"[BINARY_DOUBLE,8]
   7 - (#keys=1) VECTOR_DISTANCE("VTIX_CNPART"."DATA_VECTOR" /*+
LOB_BY_VALUE */ , VECTOR(:QUERY_VECTOR, *, * /*+  USEBLOBPCW_QVCGMD */ ),
       EUCLIDEAN)[BINARY_DOUBLE,8],
"VTIX_CNPART"."BASE_TABLE_ROWID"[ROWID,10]
   8 - (#keys=1) "VTIX_CNPART"."BASE_TABLE_ROWID"[ROWID,10],
VECTOR_DISTANCE("VTIX_CNPART"."DATA_VECTOR" /*+ LOB_BY_VALUE */ ,
VECTOR(:QUERY_VECTOR, *, * /*+
       USEBLOBPCW_QVCGMD */ ), EUCLIDEAN)[BINARY_DOUBLE,8]
   9 - "VW_IVCR_B5B87E67"."CENTROID_ID"[NUMBER,22],
"VW_IVCR_B5B87E67"."CENTROID_ID"[NUMBER,22]
  10 - "CENTROID_ID"[NUMBER,22]
  11 - "VW_IVCN_9A1D2119"."CENTROID_ID"[NUMBER,22]
  12 - "VW_IVCN_9A1D2119"."CENTROID_ID"[NUMBER,22]
  13 - (#keys=1)
VECTOR_DISTANCE("VECTOR$GENVEC_IVF_IDX$87355_87370_0$IVF_FLAT_CENTROIDS"."C
ENTROID_VECTOR" /*+ LOB_BY_VALUE */ , VECTOR(:QUERY_VECTOR, *, *

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-----------
       /*+  USEBLOBPCW_QVCGMD */ ), EUCLIDEAN)[BINARY_DOUBLE,8],
"VTIX_CENTRD"."CENTROID_ID"[NUMBER,22]
  14 - "VTIX_CENTRD"."CENTROID_ID"[NUMBER,22],
```

```
VECTOR_DISTANCE("VECTOR$GENVEC_IVF_IDX$87355_87370_0$IVF_FLAT_CENTROIDS"."C
ENTROID_VECTOR" /*+ LOB_BY_VALUE */
       , VECTOR(:QUERY_VECTOR, *, * /*+  USEBLOBPCW_QVCGMD */ ), EUCLIDEAN)
[BINARY_DOUBLE,8]
  15 - "VTIX_CNPART"."BASE_TABLE_ROWID"[ROWID,10],
"VTIX_CNPART"."CENTROID_ID"[NUMBER,22],
VECTOR_DISTANCE("VTIX_CNPART"."DATA_VECTOR" /*+ LOB_BY_VALUE */ ,
       VECTOR(:QUERY_VECTOR, *, * /*+  USEBLOBPCW_QVCGMD */ ), EUCLIDEAN)
[BINARY_DOUBLE,8]
  16 - "VTIX_CNPART"."BASE_TABLE_ROWID"[ROWID,10],
"VTIX_CNPART"."CENTROID_ID"[NUMBER,22],
VECTOR_DISTANCE("VTIX_CNPART"."DATA_VECTOR" /*+ LOB_BY_VALUE */ ,
       VECTOR(:QUERY_VECTOR, *, * /*+  USEBLOBPCW_QVCGMD */ ), EUCLIDEAN)
[BINARY_DOUBLE,8]
  17 - "GENVEC".ROWID[ROWID,10], "GENVEC"."NAME"[VARCHAR2,500]

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - this is an adaptive plan

83 rows selected.
```

16. Finally, run the similarity search once again.

```
SELECT name
  FROM genvec
  ORDER BY VECTOR_DISTANCE(v, :query_vector, EUCLIDEAN)
  FETCH APPROX FIRST 20 ROWS ONLY;
```

Example output:

```
NAME

________
C5
C5-15
C5-13
C5-3
C5-11
C5-1
C5-8
C5-6
C5-7
C5-12
C5-9
C5-4
C5-2
C5-5

NAME

________
C5-14
C5-10
C4-5
C4-12
```

```
C4-4
C4-13

20 rows selected.
```

SQL Quick Start Using a BINARY Vector Generator

A set of procedures generate `BINARY` vectors, providing a simple way to get started with Oracle AI Vector Search without a vector embedding model.

The included procedures allow you to randomly generate binary vectors with a specified number of dimensions and clusters. The output of the generation process is the population of a table called `genbvec` that you can then use, for example, to experiment with similarity searches.

The following instructions assume you already have access to a database account with sufficient privileges (minimally the `DB_DEVELOPER_ROLE` role).

> **Note:**
>
> Do not use the `BINARY` vector generator on production databases. This tutorial is made available for testing and demo purposes.

> **Note:**
>
> If you already have access to a third-party `BINARY` vector embedding model, you can perform a real text-to-`BINARY`-embedding transformation by calling third-party REST APIs using the Vector Utility PL/SQL package `DBMS_VECTOR`. For more information, refer to the example in Convert Text String to BINARY Embedding Outside Oracle Database.

1. Create the `genbvec` table.

```
DROP TABLE genbvec PURGE;

CREATE TABLE genbvec (
  id NUMBER,            -- id of the generated vector
  v VECTOR(*, BINARY),  -- generated vector
  name VARCHAR2(500),   -- name for the generated vector: C1 to Cn are
centroids, Cx_y is vector number y in cluster number x
  bv VARCHAR2(40),      -- bit version of the generated vector
  ly NUMBER             -- random number you can use to filter out rows in
addiion to similarity search on vectors
);
```

2. Create the procedures used to generate `BINARY` vectors:

```
CREATE OR REPLACE PROCEDURE generate_random_binary_vector(
dimensions IN NUMBER,
result_int OUT VECTOR,
result_binary OUT VARCHAR2
) IS
```

```
    binary_vector VARCHAR2(40);
    int8_value NUMBER;
    number_of_bits NUMBER;
    char_vector VARCHAR2(40);
BEGIN
  -- Validate dimension is a multiple of 8
  IF MOD(dimensions, 8) != 0 THEN
    RAISE_APPLICATION_ERROR(-20001, 'Number of dimensions must be a
multiple of 8');
  END IF;

  -- Generate the random binary vector
  binary_vector := '';
  FOR i IN 1 .. dimensions LOOP
    IF DBMS_RANDOM.VALUE(0, 1) < 0.5 THEN
      binary_vector := binary_vector || '0';
    ELSE
      binary_vector := binary_vector || '1';
    END IF;
  END LOOP;

  -- Convert 8-bit packets to their int8 values and build the result string
  number_of_bits := dimensions/8;
  char_vector := '[';
  FOR i IN 0 .. number_of_bits - 1 LOOP
    int8_value := 0;
    FOR j IN 0 .. 7 LOOP
      int8_value := int8_value + TO_NUMBER(SUBSTR(binary_vector, i*8+j+1,
1)) * POWER(2, j);
    END LOOP;
    char_vector := char_vector || int8_value;
    IF i < number_of_bits - 1 THEN
      char_vector := char_vector || ',';
    END IF;
  END LOOP;
  char_vector := char_vector || ']';

  -- Return the generated vector value
  result_int := to_vector(char_vector, dimensions, BINARY);
  result_binary := binary_vector;
END generate_random_binary_vector;
/


CREATE OR REPLACE PROCEDURE generate_binary_cluster(
  centroid IN VARCHAR2,            -- a string of 1 and 0
  spread IN NUMBER,                -- Maximum Hamming distance between
centroid and other vectors in the same cluster
  cluster_size IN NUMBER,          -- Number of vectors to generate in
addition to the centroid
  result_binary OUT SYS_REFCURSOR,
  result_int8 OUT SYS_REFCURSOR
) IS
    dimension NUMBER;
    max_spread NUMBER;
    vector VARCHAR2(40);
```

```
    char_vector VARCHAR2(40);
    flip_positions DBMS_SQL.VARCHAR2_TABLE;
    random_position NUMBER;
    tresult_binary DBMS_SQL.VARCHAR2_TABLE;
    tresult_int8 DBMS_SQL.VARCHAR2_TABLE;
    binary_vector VARCHAR2(40);
    cluster_index NUMBER := 1;
    number_of_bits NUMBER;
    int8_value NUMBER;
BEGIN
  -- Determine the dimension of the centroid vector
  dimension := LENGTH(centroid);

  -- Ensure dimension is a multiple of 8
  IF MOD(dimension, 8) != 0 THEN
    RAISE_APPLICATION_ERROR(-20001, 'Number of dimensions must be a
multiple of 8');
  END IF;

  -- Generate the cluster of binary vectors
  WHILE cluster_index <= cluster_size LOOP
    binary_vector := centroid;

    -- Randomly flip bits in the centroid vector with a max of spread bits
    max_spread := TRUNC(DBMS_RANDOM.VALUE(1, spread+1));
    flip_positions.DELETE;
    FOR i IN 1 .. max_spread LOOP
      random_position := TRUNC(DBMS_RANDOM.VALUE(1, dimension+1));
      -- Ensure no duplicates
      WHILE flip_positions.EXISTS(random_position) LOOP
        random_position := TRUNC(DBMS_RANDOM.VALUE(1, dimension+1));
      END LOOP;
      flip_positions(random_position) := '1';
    END LOOP;

    -- Apply flips to binary vector
    FOR i IN 1 .. dimension LOOP
      IF flip_positions.EXISTS(i) THEN
        IF SUBSTR(binary_vector, i, 1) = '0' THEN
          binary_vector := SUBSTR(binary_vector, 1, i-1) || '1' ||
SUBSTR(binary_vector, i+1);
        ELSE
          binary_vector := SUBSTR(binary_vector, 1, i-1) || '0' ||
SUBSTR(binary_vector, i+1);
        END IF;
      END IF;
    END LOOP;

    -- Convert binary vector to int8 values
    number_of_bits := dimension/8;
    char_vector := '[';
    FOR i IN 0 .. number_of_bits-1 LOOP
      int8_value := 0;
      FOR j IN 0 .. 7 LOOP
        int8_value := int8_value + TO_NUMBER(SUBSTR(binary_vector,
i*8+j+1, 1)) * POWER(2, j);
```

```
      END LOOP;
      char_vector := char_vector || int8_value;
      IF i < number_of_bits-1 THEN
        char_vector := char_vector || ',';
      END IF;
    END LOOP;
    char_vector := char_vector || ']';

    -- Add generated vectors to result tables
    tresult_binary(cluster_index) := binary_vector;
    tresult_int8(cluster_index) := char_vector;

    cluster_index := cluster_index + 1;
    END LOOP;

    -- Open cursor for binary result set
    OPEN result_binary FOR
      SELECT COLUMN_VALUE AS binary_vector
      FROM TABLE(tresult_binary);

    -- Open cursor for int8 result set
    OPEN result_int8 FOR
      SELECT COLUMN_VALUE AS int8_vector
      FROM TABLE(tresult_int8);

END generate_binary_cluster;
/


CREATE OR REPLACE PROCEDURE generate_binary_vectors(
  num_vectors NUMBER,    -- If numbers of vector is not a multiple of
num_clusters, remaining vectors are not generated
  num_clusters NUMBER,  -- Must be greater than 0
  dimensions NUMBER,    -- Must be a multiple of 8
  cluster_spread NUMBER -- Maximum Hamming distance between centroid and
other vectors in the same cluster: max number of bits flipped
) IS
    vectors_per_cluster NUMBER;
    remaining_vectors NUMBER;
    i NUMBER := 1;
    j NUMBER := 1;
    idx NUMBER := 1;
    max_id NUMBER;
    ri VECTOR(*, BINARY);
    rb VARCHAR2(40);
    result_binary SYS_REFCURSOR;
    result_int8 SYS_REFCURSOR;
    vb VARCHAR2(40);
    vi VARCHAR2(40);
BEGIN

  IF (num_vectors) <=0 OR (num_clusters < 1) OR (num_vectors <
num_clusters)
        OR (dimensions <= 0) OR (dimensions > 504) OR (cluster_spread <=
0) THEN
    RAISE_APPLICATION_ERROR(-20001, 'Issues with arguments provided');
```

```
  END IF;

  SELECT MAX(id) INTO max_id FROM genbvec;
  IF max_id IS NULL THEN max_id := 0;
  END IF;

  -- Calculate vectors per cluster
  vectors_per_cluster := TRUNC(num_vectors / num_clusters);
  remaining_vectors := num_vectors MOD num_clusters; -- remaining vectors
are not generated

  -- Generate cluster centroids
  FOR i IN 1..num_clusters LOOP
    generate_random_binary_vector(dimensions, ri, rb);
    INSERT INTO genbvec VALUES (max_id + idx, ri, 'C'||i, rb,
DBMS_RANDOM.VALUE(3,600000000));
    idx := idx + 1;

    -- Generate vectors for each cluster
    IF vectors_per_cluster > 1 THEN
      generate_binary_cluster(rb, cluster_spread, vectors_per_cluster,
result_binary, result_int8);

      -- Output the binary result
      j:= 1;
      LOOP
        FETCH result_binary INTO vb;
        FETCH result_int8 INTO vi;
        EXIT WHEN result_binary%NOTFOUND;
        ri := TO_VECTOR(vi, dimensions, BINARY);
        INSERT INTO genbvec VALUES (max_id + idx, ri, 'C'||i||'-'||j, vb,
DBMS_RANDOM.VALUE(3,600000000));
        j := j+1;
        idx := idx + 1;
      END LOOP;
      CLOSE result_binary;
      CLOSE result_int8;
    END IF;
  END LOOP;
  COMMIT;

END generate_binary_vectors;
/
```

3. After you have your vector generator procedures set up, you can run the commands in this step to get started experimenting with `BINARY` vectors in the database.

   This example generates two clusters, each having twenty-one 32-dimension vectors (including the centroid) with a maximum spread of 3 from the centroid:

   a. Start out by generating some `BINARY` vectors using the `generate_binary_vectors` procedure. The results of the generation are inserted into the table, `genbvec`.

```
BEGIN
  generate_binary_vectors(
    num_vectors  =>   40,   -- If numbers of vector is not a multiple
of num_clusters, remaining vectors are not generated
```

```
    num_clusters =>    2,   -- Must be grather than 0
    dimensions   =>   32,   -- Must be a multiple of 8 and less than 504
    cluster_spread =>  3    -- Maximum Hamming distance between
centroid and other vectors in the same cluster: max number of bits
flipped
  );
END;
/
```

b. Run a `SELECT` statement to view the generated `BINARY` vectors.

```
SET SERVEROUTPUT ON;

SELECT id, v, name, VECTOR_DIMENSION_COUNT(v) DIMS,
VECTOR_DIMENSION_FORMAT(v) FORMAT, bv, ly FROM genbvec;
```

Example output:

```
     ID  V                    NAME       DIMS      FORMAT
BV                                                    LY
-------  -------------------  ---------  --------  ----------
----------------------------------  --------------
      1  [24,153,161,63]      C1         32        BINARY
0001100010011001100010111111100     99789021.1
      2  [24,153,165,63]      C1-1       32        BINARY
0001100010011001101010010111111100     60221003.5
      3  [26,152,165,63]      C1-2       32        BINARY
0101100000011001101010010111111100     387124796
      4  [24,201,161,62]      C1-3       32        BINARY
0001100010010011100010101111100     291263868
      5  [24,187,161,63]      C1-4       32        BINARY
0001100011011101100010111111100     583827824
      6  [24,153,161,61]      C1-5       32        BINARY
0001100010011001100010110111100     144826451
      7  [24,153,171,55]      C1-6       32        BINARY
0001100010011001110101011101100     113684378
      8  [88,153,161,61]      C1-7       32        BINARY
0001101010011001100010110111100     312081799
      9  [152,217,161,47]     C1-8       32        BINARY
0001100110011011100010111110100     173971628
     10  [24,153,163,59]      C1-9       32        BINARY
0001100010011001110010111011100     500775192
     11  [24,153,160,61]      C1-10      32        BINARY
0001100010011001000010110111100     137309652
     12  [25,185,161,47]      C1-11      32        BINARY
1001100010011101100010111110100     483392712
     13  [89,153,161,63]      C1-12      32        BINARY
1001101010011001100010111111100     458730494
     14  [24,153,229,31]      C1-13      32        BINARY
0001100010011001101001111111000     325738354

     ID  V                    NAME       DIMS      FORMAT
BV                                                    LY
-------  -------------------  ---------  --------  ----------
```

```
----------------------------------  --------------
     15  [24,152,161,63]      C1-14      32        BINARY
00011000000110011000010111111100     260267242
     16  [24,153,165,63]      C1-15      32        BINARY
00011000100110011010010111111100     153663322
     17  [24,137,169,63]      C1-16      32        BINARY
00011000100100011001010111111100     411918780
     18  [24,185,161,63]      C1-17      32        BINARY
00011000100111011000010111111100     53885587.1
     19  [152,137,161,63]     C1-18      32        BINARY
00011001100100011000010111111100     321305137
     20  [25,153,161,63]      C1-19      32        BINARY
10011000100110011000010111111100     180742593
     21  [16,153,161,63]      C1-20      32        BINARY
00001000100110011000010111111100     511768659
     22  [183,107,24,190]     C2         32        BINARY
11101101110101100001100001111101     529205377
     23  [181,251,24,190]     C2-1       32        BINARY
10101101110111110001100001111101     391560729
     24  [191,107,25,186]     C2-2       32        BINARY
11111101110101101001100001011101     191852938
     25  [182,106,24,190]     C2-3       32        BINARY
01101101010101100001100001111101     164088550
     26  [183,107,56,187]     C2-4       32        BINARY
11101101110101100001110011011101     20400437.6
     27  [183,106,16,190]     C2-5       32        BINARY
11101101010101100000100001111101     363725396
     28  [183,107,40,190]     C2-6       32        BINARY
11101101110101100001010001111101     144549103

     ID  V                    NAME       DIMS      FORMAT
BV                                                    LY
-------  -------------------  ---------  --------  ----------
----------------------------------  --------------
     29  [183,107,26,190]     C2-7       32        BINARY
11101101110101100101100001111101     318036129
     30  [183,123,24,188]     C2-8       32        BINARY
11101101110111100001100000111101     309460286
     31  [179,35,24,190]      C2-9       32        BINARY
11001101110001000001100001111101     25042254.7
     32  [182,251,24,190]     C2-10      32        BINARY
01101101110111110001100001111101     355499793
     33  [183,251,24,190]     C2-11      32        BINARY
11101101110111110001100001111101     483002129
     34  [183,107,24,254]     C2-12      32        BINARY
11101101110101100001100001111111     497697267
     35  [183,42,24,158]      C2-13      32        BINARY
11101101010101000001100001111001     64446273.5
     36  [151,107,28,186]     C2-14      32        BINARY
11101001110101100011100001011101     248483969
     37  [167,43,16,190]      C2-15      32        BINARY
11100101110101000000100001111101     513880134
     38  [183,106,24,190]     C2-16      32        BINARY
11101101010101100001100001111101     558247180
     39  [183,123,24,190]     C2-17      32        BINARY
11101101110111100001100001111101     287706546
```

```
     40  [151,107,24,190]     C2-18      32        BINARY
11101001110101100001100001111101     309138884
     41  [167,107,28,186]     C2-19      32        BINARY
11100101110101100011100001011101     433932877
     42  [63,106,24,190]      C2-20      32        BINARY
11111100010101100001100001111101     84539416.7
```

c. Perform a similarity search on the `BINARY` vectors.

```
SELECT name, v, bv, VECTOR_DISTANCE(v,
  (SELECT v FROM genbvec WHERE name='C1'), HAMMING) DISTANCE
FROM genbvec
ORDER BY VECTOR_DISTANCE(v, (SELECT v FROM genbvec WHERE name='C1'),
HAMMING);
```

Example output:

```
   NAME  V                  BV                                 DISTANCE
-------  -----------------  ---------------------------------
----------
     C1  [24,153,161,63]    00011000100110011000010111111100   0
   C1-1  [24,153,165,63]    00011000100110011010010111111100   1.0
  C1-20  [16,153,161,63]    00001000100110011000010111111100   1.0
  C1-19  [25,153,161,63]    10011000100110011000010111111100   1.0
  C1-17  [24,185,161,63]    00011000100111011000010111111100   1.0
  C1-15  [24,153,165,63]    00011000100110011010010111111100   1.0
  C1-14  [24,152,161,63]    00011000000110011000010111111100   1.0
   C1-5  [24,153,161,61]    00011000100110011000010110111100   1.0
   C1-4  [24,187,161,63]    00011000110111011000010111111100   2.0
  C1-18  [152,137,161,63]   00011001100100011000010111111100   2.0
  C1-16  [24,137,169,63]    00011000100100011001010111111100   2.0
  C1-12  [89,153,161,63]    10011010100110011000010111111100   2.0
  C1-10  [24,153,160,61]    00011000100110010000010110111100   2.0
   C1-9  [24,153,163,59]    00011000100110011100010111011100   2.0

   NAME  V                  BV                                 DISTANCE
-------  -----------------  ---------------------------------
----------
   C1-7  [88,153,161,61]    00011010100110011000010110111100   2.0
   C1-2  [26,152,165,63]    01011000000110011010010111111100   3.0
   C1-3  [24,201,161,62]    00011000100100111000010101111100   3.0
   C1-6  [24,153,171,55]    00011000100110011101010111101100   3.0
   C1-8  [152,217,161,47]   00011001100110111000010111110100   3.0
  C1-11  [25,185,161,47]    10011000100111011000010111110100   3.0
  C1-13  [24,153,229,31]    00011000100110011010011111111000   3.0
   C2-1  [181,251,24,190]   10101101110111110001100001111101   15.0
  C2-10  [182,251,24,190]   01101101110111110001100001111101   15.0
   C2-6  [183,107,40,190]   11101101110101100001010001111101   16.0
  C2-11  [183,251,24,190]   11101101110111110001100001111101   16.0
   C2-2  [191,107,25,186]   11111101110101101001100001011101   17.0
  C2-20  [63,106,24,190]    11111100010101100001100001111101   17.0
  C2-18  [151,107,24,190]   11101001110101100001100001111101   17.0

   NAME  V                  BV                                 DISTANCE
```

```
-------  -----------------  --------------------------------
----------
  C2-17  [183,123,24,190]   11101101110111100001100001111101   17.0
  C2-15  [167,43,16,190]    11100101110101000000100001111101   17.0
   C2-9  [179,35,24,190]    11001101110001000001100001111101   17.0
   C2-4  [183,107,56,187]   11101101110101100001110011011101   17.0
     C2  [183,107,24,190]   11101101110101100001100001111101   18.0
   C2-8  [183,123,24,188]   11101101110111100001100000111101   18.0
   C2-3  [182,106,24,190]   01101101010101100001100001111101   18.0
   C2-5  [183,106,16,190]   11101101010101100000100001111101   18.0
   C2-7  [183,107,26,190]   11101101110101100101100001111101   19.0
  C2-12  [183,107,24,254]   11101101110101100001100001111111   19.0
  C2-13  [183,42,24,158]    11101101010101000001100001111001   19.0
  C2-14  [151,107,28,186]   11101001110101100011100001011101   19.0
  C2-16  [183,106,24,190]   11101101010101100001100001111101   19.0
  C2-19  [167,107,28,186]   11100101110101100011100001011101   21.0

42 rows selected.
```

d. Run another similarity search, this time limiting the results to the first 21 rows. In this example, this means the results include `BINARY` vectors only from cluster 1.

```
SELECT name, v, bv, VECTOR_DISTANCE(v,
    (SELECT v FROM genbvec WHERE name='C1'), HAMMING) DISTANCE
FROM genbvec
ORDER BY VECTOR_DISTANCE(v, (SELECT v FROM genbvec WHERE name='C1'),
HAMMING)
FETCH EXACT FIRST 21 ROWS ONLY;
```

Example output:

```
   NAME  V                  BV                                DISTANCE
-------  -----------------  --------------------------------
----------
     C1  [24,153,161,63]    00011000100110011000010111111100   0
   C1-1  [24,153,165,63]    00011000100110011010010111111100   1.0
  C1-20  [16,153,161,63]    00001000100110011000010111111100   1.0
  C1-19  [25,153,161,63]    10011000100110011000010111111100   1.0
  C1-17  [24,185,161,63]    00011000100111011000010111111100   1.0
  C1-15  [24,153,165,63]    00011000100110011010010111111100   1.0
  C1-14  [24,152,161,63]    00011000000110011000010111111100   1.0
   C1-5  [24,153,161,61]    00011000100110011000010110111100   1.0
   C1-4  [24,187,161,63]    00011000110111011000010111111100   2.0
  C1-18  [152,137,161,63]   00011001100100011000010111111100   2.0
  C1-16  [24,137,169,63]    00011000100100011001010111111100   2.0
  C1-12  [89,153,161,63]    10011010100110011000010111111100   2.0
  C1-10  [24,153,160,61]    00011000100110010000010110111100   2.0
   C1-9  [24,153,163,59]    00011000100110011100010111011100   2.0

   NAME  V                  BV                                DISTANCE
-------  -----------------  --------------------------------
----------
   C1-7  [88,153,161,61]    00011010100110011000010110111100   2.0
   C1-2  [26,152,165,63]    01011000000110011010010111111100   3.0
```

```
   C1-3  [24,201,161,62]     00011000100100111000010101111100   3.0
   C1-6  [24,153,171,55]     00011000100110011110101011101100   3.0
   C1-8  [152,217,161,47]    00011001100110111000010111110100   3.0
  C1-11  [25,185,161,47]     10011000100111011000010111110100   3.0
  C1-13  [24,153,229,31]     00011000100110011010011111111000   3.0

21 rows selected.
```

e. In this iteration, the similarity search omits the `HAMMING` distance metric. However, because `HAMMING` is the default metric used with `BINARY` vectors, the results are the same as the previous query.

```
SELECT name, v, bv, VECTOR_DISTANCE(v,
    (SELECT v FROM genbvec WHERE name='C1')) DISTANCE
FROM genbvec
ORDER BY VECTOR_DISTANCE(v, (SELECT v FROM genbvec WHERE name='C1'))
FETCH EXACT FIRST 21 ROWS ONLY;
```

Example output:

```
   NAME  V                  BV                                 DISTANCE
-------  -----------------  ---------------------------------
----------
     C1  [24,153,161,63]    00011000100110011000010111111100   0
   C1-1  [24,153,165,63]    00011000100110011010010111111100   1.0
  C1-20  [16,153,161,63]    00001000100110011000010111111100   1.0
  C1-19  [25,153,161,63]    10011000100110011000010111111100   1.0
  C1-17  [24,185,161,63]    00011000100111011000010111111100   1.0
  C1-15  [24,153,165,63]    00011000100110011010010111111100   1.0
  C1-14  [24,152,161,63]    00011000000110011000010111111100   1.0
   C1-5  [24,153,161,61]    00011000100110011000010110111100   1.0
   C1-4  [24,187,161,63]    00011000110111011000010111111100   2.0
  C1-18  [152,137,161,63]   00011001100100011000010111111100   2.0
  C1-16  [24,137,169,63]    00011000100100011001010111111100   2.0
  C1-12  [89,153,161,63]    10011010100110011000010111111100   2.0
  C1-10  [24,153,160,61]    00011000100110010000010110111100   2.0
   C1-9  [24,153,163,59]    00011000100110011100010111011100   2.0

   NAME  V                  BV                                 DISTANCE
-------  -----------------  ---------------------------------
----------
   C1-7  [88,153,161,61]    00011010100110011000010110111100   2.0
   C1-2  [26,152,165,63]    01011000000110011010010111111100   3.0
   C1-3  [24,201,161,62]    00011000100100111000010101111100   3.0
   C1-6  [24,153,171,55]    00011000100110011110101011101100   3.0
   C1-8  [152,217,161,47]   00011001100110111000010111110100   3.0
  C1-11  [25,185,161,47]    10011000100111011000010111110100   3.0
  C1-13  [24,153,229,31]    00011000100110011010011111111000   3.0

21 rows selected.
```

Your Vector Documentation Map to GenAI Prompts

Follow the included steps to increase your chance of getting better and more consistent answers about this document from your preferred Generative AI Chatbot with internet search capabilities.

1. Click on a topic of interest in the AI Vector Search Topic Map section of this page.
2. Copy and paste the corresponding prompt in your preferred Chatbot UI.
3. Enter your question at the end of the prompt.
4. Run your prompt.

> **WARNING:**
>
> Your use of Generative AI Chatbots is solely at your own risk, and you are solely responsible for complying with any terms and conditions related to the use of any such chatbots. Oracle provides these information map prompts only for convenience and educational purposes. Notwithstanding any other terms and conditions related to the use of Generative AI Chatbots, your use of these prompts constitutes your acceptance of that risk and expresses exclusion of Oracle's responsibility or liability for any damages resulting from such use.

AI Vector Search Topic Map

- AI Vector Search Overview
- Get Started
- Generate Vector Embeddings
  - Understand Vector Embeddings Generation
  - Convert Pretrained Models to ONNX Format
  - Import ONNX Models into Oracle Database
  - Access Third-Party Models for Vector Generation
  - Generate Embedding Functions and Examples
  - Perform Chunking with Embedding Examples
  - Configure Chunking Parameters and Chunking Functions
  - Store Vector Embeddings in Relational Tables
- Vector Indexes
  - Size the Vector Pool
  - Hierarchical Navigable Small World Indexes
  - Inverted File Flat Vector Indexes
  - Guidelines for Vector Indexes
  - Create and Alter Hybrid Vector Index
  - Search Hybrid Vector Index
- Vector Operations

  - Distance Metrics
  - Vector Distance Functions and Operators
  - Vector Constructors, Converters, Descriptors, Arithmetic Operators and Aggregate Functions
  - Query Data with Similarity Searches
  - Work with LLM-Powered APIs
  - Retrieval Augmented Generation
- Vector Diagnostics
  - Oracle AI Vector Search Views
    * Text Processing Views
    * Views Related to Vector Indexes and Hybrid Vector Indexes
  - Oracle AI Vector Search Statistics and Initialization Parameters
  - Vector Search PL/SQL Packages

AI Vector Search Topic Map Prompts

AI Vector Search Overview

```
Provided URLs:
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/overview-
ai-vector-search.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/why-use-
vector-search-instead-other-vector-databases.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/oracle-ai-
vector-search-workflow.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
glossary.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/supported-
clients-and-languages.html

Instructions:
  1. Set temperature to 0 to answer question.
  2. Open and read provided URLs to extract only relevant content and
verbatim code examples.
  3. Answer question by making sure you only use previously extracted
relevant content and verbatim code examples.

Question:
<enter your question here>
```

Get Started

```
Provided URLs:
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/sql-quick-
start-using-vector-embedding-model-uploaded-database.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/sql-quick-
start-using-float32-vector-generator.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/sql-quick-
start-using-binary-vector-generator.html
```

```
Instructions:
  1. Set temperature to 0 to answer question.
  2. Open and read provided URLs to extract only relevant content and
verbatim code examples.
  3. Answer question by making sure you only use previously extracted
relevant content and verbatim code examples.

Question:
<enter your question here>
```

Understand Vector Embeddings Generation

```
Provided URLs:
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
understand-stages-data-transformations.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/sql-
functions-generate-embeddings.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/chainable-
utility-functions-and-common-use-cases.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/vector-
helper-procedures.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/supplied-
vector-utility-pl-sql-packages.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/supported-
third-party-provider-operations-and-endpoints.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/terms-
using-vector-utility-pl-sql-packages.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/validate-
json-input-parameters.html

Instructions:
  1. Set temperature to 0 to answer question.
  2. Open and read provided URLs to extract only relevant content and
verbatim code examples.
  3. Answer question by making sure you only use previously extracted
relevant content and verbatim code examples.

Question:
<enter your question here>
```

Convert Pretrained Models to ONNX Format

```
Provided URLs:
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/onnx-
pipeline-models-text-embedding.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/onnx-
pipeline-models-image-embedding.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/onnx-
pipeline-models-multi-modal-embedding.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/onnx-
pipeline-models-text-classification.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/onnx-
pipeline-models-reranking.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/convert-
pretrained-models-onnx-model-end-end-instructions.html
```

```
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/python-
classes-convert-pretrained-models-onnx-models.html

Instructions:
  1. Set temperature to 0 to answer question.
  2. Open and read provided URLs to extract only relevant content and
verbatim code examples.
  3. Answer question by making sure you only use previously extracted
relevant content and verbatim code examples.

Question:
<enter your question here>
```

Import ONNX Models into Oracle Database

```
Provided URLs:
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/import-
onnx-models-oracle-database-end-end-example.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/alternate-
method-import-onnx-models.html

Instructions:
  1. Set temperature to 0 to answer question.
  2. Open and read provided URLs to extract only relevant content and
verbatim code examples.
  3. Answer question by making sure you only use previously extracted
relevant content and verbatim code examples.

Question:
<enter your question here>
```

Access Third-Party Models for Vector Generation

```
Provided URLs:
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/access-
third-party-models-vector-generation-leveraging-third-party-rest-apis.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
dbms_vector-vecse.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
dbms_vector_chain-vecse.html

Instructions:
  1. Set temperature to 0 to answer question.
  2. Open and read provided URLs to extract only relevant content and
verbatim code examples.
  3. Answer question by making sure you only use previously extracted
relevant content and verbatim code examples.

Question:
<enter your question here>
```

Generate Embedding Functions and Examples

```
Provided URLs:
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
vector_embedding.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
vector_chunks.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/convert-
text-string-embedding-oracle-database.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/convert-
text-string-binary-embedding-oracle-database.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/convert-
text-string-embedding-using-public-third-party-apis.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/convert-
text-string-embedding-locally-ollama.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/convert-
image-embedding-using-public-third-party-apis.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/vectorize-
relational-tables-using-oml-feature-extraction-algorithms.html

Instructions:
  1. Set temperature to 0 to answer question.
  2. Open and read provided URLs to extract only relevant content and
verbatim code examples.
  3. Answer question by making sure you only use previously extracted
relevant content and verbatim code examples.

Question:
<enter your question here>
```

Perform Chunking With Embedding Examples

```
Provided URLs:
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/convert-
text-chunks-custom-chunking-specifications.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/convert-
file-text-chunks-embeddings-oracle-database.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/convert-
file-embeddings-oracle-database.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/generate-
and-use-embeddings-end-end-search.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
vector_chunks.html

Instructions:
  1. Set temperature to 0 to answer question.
  2. Open and read provided URLs to extract only relevant content and
verbatim code examples.
  3. Answer question by making sure you only use previously extracted
relevant content and verbatim code examples.

Question:
<enter your question here>
```

Configure Chunking Parameters and Chunking Functions

```
Provided URLs:
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/explore-
chunking-techniques-and-examples.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/create-
and-use-custom-vocabulary.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/create-
and-use-custom-language-data.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
vector_chunks.html

Instructions:
  1. Set temperature to 0 to answer question.
  2. Open and read provided URLs to extract only relevant content and
verbatim code examples.
  3. Answer question by making sure you only use previously extracted
relevant content and verbatim code examples.

Question:
<enter your question here>
```

Store Vector Embeddings in Relational Tables

```
Provided URLs:
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/create-
tables-using-vector-data-type.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/vectors-
external-tables.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/querying-
inline-external-table.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
performing-semantic-similarity-search-using-external-table.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/vectors-
distributed-database-tables.html#GUID-F75FB9CA-E7D1-46BC-847A-7324EF21D829
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/binary-
vectors.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/sparse-
vectors.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/insert-
vectors-database-table-using-insert-statement.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/load-
character-vector-data-using-sqlloader-example.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/load-
binary-vector-data-using-sqlloader-example.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/unload-
and-load-vectors-using-oracle-data-pump.html

Instructions:
  1. Set temperature to 0 to answer question.
  2. Open and read provided URLs to extract only relevant content and
verbatim code examples.
  3. Answer question by making sure you only use previously extracted
relevant content and verbatim code examples.
```

```
Question:
<enter your question here>
```

Size the Vector Pool

```
Provided URLs:
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/size-
vector-pool.html

Instructions:
  1. Set temperature to 0 to answer question.
  2. Open and read provided URLs to extract only relevant content and
verbatim code examples.
  3. Answer question by making sure you only use previously extracted
relevant content and verbatim code examples.

Question:
<enter your question here>
```

Hierarchical Navigable Small World Indexes

```
Provided URLs:
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
understand-hierarchical-navigable-small-world-indexes.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
hierarchical-navigable-small-world-index-syntax-and-parameters.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
understand-transaction-support-tables-hnsw-indexes.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
understand-hnsw-index-population-mechanisms-oracle-rac-and-single-
instance.html

Instructions:
  1. Set temperature to 0 to answer question.
  2. Open and read provided URLs to extract only relevant content and
verbatim code examples.
  3. Answer question by making sure you only use previously extracted
relevant content and verbatim code examples.

Question:
<enter your question here>
```

Inverted File Flat Vector Indexes

```
Provided URLs:
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
understand-inverted-file-flat-vector-indexes.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/inverted-
file-flat-index-syntax-and-parameters.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/inverted-
file-flat-vector-indexes-partitioning-schemes.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/included-
columns.html#GUID-641E6D17-6B59-4860-A5BD-DF3D33793D43
```

```
Instructions:
  1. Set temperature to 0 to answer question.
  2. Open and read provided URLs to extract only relevant content and
verbatim code examples.
  3. Answer question by making sure you only use previously extracted
relevant content and verbatim code examples.

Question:
<enter your question here>
```

Guidelines for Vector Indexes

```
Provided URLs:
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
guidelines-using-vector-indexes.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/index-
accuracy-report.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/vector-
index-status-checkpoint-and-advisor-procedures.html

Instructions:
  1. Set temperature to 0 to answer question.
  2. Open and read provided URLs to extract only relevant content and
verbatim code examples.
  3. Answer question by making sure you only use previously extracted
relevant content and verbatim code examples.

Question:
<enter your question here>
```

Create and Alter Hybrid Vector Index

```
Provided URLs:
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/create-
vector-indexes-and-hybrid-vector-indexes.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
understand-hybrid-vector-indexes.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/create-
hybrid-vector-index.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/alter-
index.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
guidelines-and-restrictions-hybrid-vector-indexes.html

Instructions:
  1. Set temperature to 0 to answer question.
  2. Open and read provided URLs to extract only relevant content and
verbatim code examples.
  3. Answer question by making sure you only use previously extracted
relevant content and verbatim code examples.

Question:
<enter your question here>
```

Search Hybrid Vector Index

```
Provided URLs:
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
understand-hybrid-search.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/query-
hybrid-vector-indexes-end-end-example.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
search.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
searchpipeline.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
get_sql.html

Instructions:
  1. Set temperature to 0 to answer question.
  2. Open and read provided URLs to extract only relevant content and
verbatim code examples.
  3. Answer question by making sure you only use previously extracted
relevant content and verbatim code examples.

Question:
<enter your question here>
```

Distance Metrics

```
Provided URLs:
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/euclidean-
and-squared-euclidean-distances.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/cosine-
similarity.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/dot-
product-similarity.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/manhattan-
distance.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/hamming-
distance.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/jaccard-
similarity.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/custom-
distance-function.html

Instructions:
  1. Set temperature to 0 to answer question.
  2. Open and read provided URLs to extract only relevant content and
verbatim code examples.
  3. Answer question by making sure you only use previously extracted
relevant content and verbatim code examples.

Question:
<enter your question here>
```

Vector Distance Functions and Operators

```
Provided URLs:
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
vector_distance.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
l1_distance.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
l2_distance.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
cosine_distance.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
inner_product.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
hamming_distance-vecse.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
jaccard_distance-vecse.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/custom-
distance-function.html

Instructions:
  1. Set temperature to 0 to answer question.
  2. Open and read provided URLs to extract only relevant content and
verbatim code examples.
  3. Answer question by making sure you only use previously extracted
relevant content and verbatim code examples.

Question:
<enter your question here>
```

Vector Constructors, Converters, Descriptors, Arithmetic Operators and Aggregate Functions

```
Provided URLs:
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/to_vector-
vecse.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/vector-
vecse.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
from_vector-vecse.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
vector_serialize-vecse.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
vector_norm-vecse.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
vector_dimension_count-vecse.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
vector_dims-vecse.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
vector_dimension_format-vecse.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
arithmetic-operators.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
avg_vecse.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
```

```
sum_vecse.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/json-
compatibility-vector-data-type.html

Instructions:
  1. Set temperature to 0 to answer question.
  2. Open and read provided URLs to extract only relevant content and
verbatim code examples.
  3. Answer question by making sure you only use previously extracted
relevant content and verbatim code examples.

Question:
<enter your question here>
```

Query Data with Similarity Searches

```
Provided URLs:
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/perform-
exact-similarity-search.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
understand-approximate-similarity-search-using-vector-indexes.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/optimizer-
plans-hnsw-vector-indexes.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/optimizer-
plans-ivf-vector-indexes.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
approximate-search-using-hnsw.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
approximate-search-using-ivf.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/vector-
index-hints.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/perform-
multi-vector-similarity-search.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
terminable-iteration-ivf.html#VECSE-GUID-26BA0FE6-59A7-48B9-A39C-FFF49E5349A1

Instructions:
  1. Set temperature to 0 to answer question.
  2. Open and read provided URLs to extract only relevant content and
verbatim code examples.
  3. Answer question by making sure you only use previously extracted
relevant content and verbatim code examples.

Question:
<enter your question here>
```

Work with LLM-Powered APIs

```
Provided URLs:
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/generate-
summary-using-public-third-party-apis.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/generate-
summary-using-ollama.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/generate-
text-using-public-third-party-apis.html
```

```
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/generate-
text-locally-ollama.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/describe-
images-using-public-third-party-apis.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/describe-
images-using-ollama.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/supported-
third-party-provider-operations-and-endpoints.html

Instructions:
  1. Set temperature to 0 to answer question.
  2. Open and read provided URLs to extract only relevant content and
verbatim code examples.
  3. Answer question by making sure you only use previously extracted
relevant content and verbatim code examples.

Question:
<enter your question here>
```

Retrieval Augmented Generation

```
Provided URLs:
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/retrieval-
augmented-generation1.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/sql-rag-
example.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/oracle-ai-
vector-search-integration-langchain.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/oracle-ai-
vector-search-integration-llamaindex.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/use-
reranking-better-rag-results.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/supported-
third-party-provider-operations-and-endpoints.html

Instructions:
  1. Set temperature to 0 to answer question.
  2. Open and read provided URLs to extract only relevant content and
verbatim code examples.
  3. Answer question by making sure you only use previously extracted
relevant content and verbatim code examples.

Question:
<enter your question here>
```

Text Processing Views

```
Provided URLs:
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
all_vector_abbrev_tokens.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
all_vector_lang.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
dba_vector_hitcounts.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
```

```
user_vector_hitcounts.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
user_vector_abbrev_tokens.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
user_vector_lang.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
user_vector_vocab.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
user_vector_vocab_tokens.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
all_vector_vocab.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
all_vector_vocab_tokens.html

Instructions:
  1. Set temperature to 0 to answer question.
  2. Open and read provided URLs to extract only relevant content and
verbatim code examples.
  3. Answer question by making sure you only use previously extracted
relevant content and verbatim code examples.

Question:
<enter your question here>
```

Views Related to Vector Indexes and Hybrid Vector Indexes

```
Provided URLs:
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
vvector_memory_pool.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/vecsys-
vectorindex.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
vvector_index.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
vvector_graph_index.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
vvector_partitions_index.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/vecsys-
vectorindexcheckpoints.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
indexnamevectors.html

Instructions:
  1. Set temperature to 0 to answer question.
  2. Open and read provided URLs to extract only relevant content and
verbatim code examples.
  3. Answer question by making sure you only use previously extracted
relevant content and verbatim code examples.

Question:
<enter your question here>
```

Oracle AI Vector Search Statistics and Initialization Parameters

```
Provided URLs:
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/oracle-ai-
vector-search-dictionary-statistics.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/oracle-
machine-learning-static-dictionary-views.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/oracle-ai-
vector-search-parameters.html

Instructions:
  1. Set temperature to 0 to answer question.
  2. Open and read provided URLs to extract only relevant content and
verbatim code examples.
  3. Answer question by making sure you only use previously extracted
relevant content and verbatim code examples.

Question:
<enter your question here>
```

Vector Search PL/SQL Packages

```
Provided URLs:
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
dbms_vector-vecse.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
create_credential-dbms_vector.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
create_index.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
disable_checkpoint.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
drop_credential-dbms_vector.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
drop_onnx_model-procedure-dbms_vector.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
enable_checkpoint.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
get_index_status.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
index_accuracy_query.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
index_accuracy_report.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
index_vector_memory_advisor.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
load_onnx_model-procedure.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/json-
metadata-parameters-onnx-models.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
load_onnx_model_cloud.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/query.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
rebuild_index.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/rerank-
```

```
dbms_vector.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
utl_to_chunks-dbms_vector.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
utl_to_embedding-and-utl_to_embeddings-dbms_vector.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
utl_to_generate_text-dbms_vector.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
dbms_vector_chain-vecse.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
create_credential-dbms_vector_chain.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
create_lang_data.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
create_preference.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
create_vocabulary.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
drop_credential-dbms_vector_chain.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
drop_lang_data.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
drop_preference.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
drop_vocabulary.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/rerank-
dbms_vector_chain.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
utl_to_chunks-dbms_vector_chain.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
utl_to_embedding-and-utl_to_embeddings-dbms_vector_chain.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
utl_to_generate_text-dbms_vector_chain.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
utl_to_summary.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
utl_to_text.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/supported-
languages-and-data-file-locations.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
dbms_hybrid_vector-vecse.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
search.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
searchpipeline.html
https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/
get_sql.html

Instructions:
  1. Set temperature to 0 to answer question.
  2. Open and read provided URLs to extract only relevant content and
verbatim code examples.
  3. Answer question by making sure you only use previously extracted
relevant content and verbatim code examples.
```

```
Question:
<enter your question here>
```

4

Generate Vector Embeddings

Oracle AI Vector Search offers Vector Utilities (SQL and PL/SQL tools) to automatically generate vector embeddings from your unstructured data, either within or outside Oracle Database.

Embeddings are vector representations that capture the semantic meaning of your data, rather than the actual words in a document or pixels in an image (as explained in Overview of Oracle AI Vector Search). A vector embedding model creates these embeddings by assigning numerical values to each element of your data (such as a word, sentence, or paragraph).

To generate embeddings within the database, you can import and use vector embedding models in ONNX format. To generate embeddings outside the database, you can access third-party vector embedding models (remotely or locally) by calling third-party REST APIs.

- About Vector Generation
  Learn about *Vector Utility SQL functions* and *Vector Utility PL/SQL packages* that help you transform unstructured data into vector embeddings.
- Import Pretrained Models in ONNX Format
  Oracle Database 23ai includes an ONNX runtime engine for running embedding models directly inside the database. This section covers the process of importing existing pretrained embedding models into Oracle database, including converting those models into the ONNX format if they are not already converted.
- Access Third-Party Models for Vector Generation Leveraging Third-Party REST APIs
  You can access third-party vector embedding models to generate vector embeddings from your data, outside the database by calling third-party REST APIs.
- Vector Generation Examples
  Run these end-to-end examples to see how you can generate vector embeddings, both within and outside the database.

About Vector Generation

Learn about *Vector Utility SQL functions* and *Vector Utility PL/SQL packages* that help you transform unstructured data into vector embeddings.

- Understand the Stages of Data Transformations
  Your input data may travel through different stages before turning into a vector.
- About SQL Functions to Generate Embeddings
  Choose to implement Vector Utility SQL functions to perform parallel or on-the-fly chunking and embedding operations, within the database. The supplied SQL functions for vector generation are `VECTOR_CHUNKS` and `VECTOR_EMBEDDING`.
- About PL/SQL Packages to Generate Embeddings
  Choose to implement Vector Utility PL/SQL packages to perform chunking, embedding, and text generation operations along with text processing and similarity search, both within and outside the database. You can schedule these operations as end-to-end pipelines. The supplied PL/SQL packages for vector generation are `DBMS_VECTOR` and `DBMS_VECTOR_CHAIN`.

Understand the Stages of Data Transformations

Your input data may travel through different stages before turning into a vector.

Depending on the size of your input data, which can range from small strings to very large documents, the data passes through a pipeline of optional transformation stages from *Plain Text* to *Chunks* to *Tokens* to *Vectors*, with *Vector Index* as the endpoint.

Prepare: Plain Text and Chunks

This stage prepares your unstructured data to ensure that it is in a format that can be processed by vector embedding models.

To prepare large unstructured textual data (for example, a PDF or Word document), you may first transform the document into plain text and then pass the resulting document through **Chunker**. The chunker then splits the plain text document into multiple appropriate-sized segments through a splitting process known as **Chunking**. A single document may be split into many **chunks**, each transformed into a vector. A chunk can be a set of words (to capture specific words or word pieces), sentences (to capture a specific context), or paragraphs (to capture broader themes).

Later, you will learn about several chunking parameters and techniques that help you define your own chunking specifications and strategies, so that you can generate relevant and meaningful chunks according to your use case.

Splitting your data into chunks is not needed for small documents, phrases, text strings, or short summaries. Embedding models can directly process such content into a vector representation. This is explained in the following section.

Embed: Tokens and Vectors

You now pass the text or extracted chunks as input through a declared vector embedding model for generating vector embeddings on each text string or chunk. Embeddings are vector representations that capture the semantic meaning or context of your data. A vector embedding model creates these embeddings by assigning numerical values to each element of your data, that is, to each word, sentence, or paragraph.

The chunks are first passed through **Tokenizer** associated with your embedding model. The tokenizer further splits the chunks into individual words or word pieces known as **tokens**. An embedding model then embeds each token into a vector representation.

Tokenizers used by embedding models usually have limits on the size of the input text (number of tokens) they can deal with, so it is important that you chunk your data beforehand into appropriate-sized segments to avoid loss of text when generating embeddings. If the number of tokens are larger than the maximum input limit imposed by the model, then some tokens get truncated into a defined input length. Note that chunking is not needed if your text meets this maximum input limit.

The chunker must select a text size that approximates the maximum input limit of your model. The actual number of tokens depends on the specified tokenizer for an embedding model, which typically uses a **vocabulary** list of words, numbers, punctuations, and pieces of tokens. A vocabulary contains a set of tokens that are collected during a statistical training process. Each tokenizer uses a different vocabulary format to process text into tokens.

For example, the BERT multilingual model uses Word-Piece encoding or the GPT model uses Byte-Pair encoding.

The BERT model may tokenize the following sentence (containing four words):

```
Embedding usecase for chunking
```

into the following eight tokens and also include `##` (number signs) to indicate non-initial pieces of words:

```
Em ##bedd ##ing use ##case for chunk ##ing
```

Vocabulary files are included as part of a model's distribution. You can supply a vocabulary file (recognized by your model's tokenizer) to the chunker beforehand, so that it can correctly estimate the token count of your input data at the time of chunking.

Populate and Query: Vector Indexes

Finally, you store the extracted vectors in a vector index to implement combined similarity and relational searches on those vectors.

Instead of creating a vector index alone, you can also choose to create a hybrid vector index, which is a combination of both vector index and Oracle Text search index. This lets you implement hybrid searches to retrieve more relevant search results by performing both vector-based similarity searches and text-based keyword searches on the same data, simultaneously.

About SQL Functions to Generate Embeddings

Choose to implement Vector Utility SQL functions to perform parallel or on-the-fly chunking and embedding operations, within the database. The supplied SQL functions for vector generation are `VECTOR_CHUNKS` and `VECTOR_EMBEDDING`.

Vector Utility SQL functions are intended for a direct and quick interaction with data, within pure SQL.

VECTOR_CHUNKS

Use the `VECTOR_CHUNKS` SQL function if you want to split plain text into chunks (pieces of words, sentences, or paragraphs) in preparation for the generation of embeddings, to be used with a vector index.

For example, you can use this function to build a standalone Text Chunking system that lets you break down a large PDF document into smaller yet semantically meaningful chunk texts. You can experiment with your chunks by running parallel chunking operations, where you can inspect each chunk text, accordingly amend the chunking results, and then proceed further with other data transformation stages.

To generate chunks, this function uses the in-house implementation with Oracle Database.

For detailed information on this function, see VECTOR_CHUNKS.

VECTOR_EMBEDDING

Use the `VECTOR_EMBEDDING` function if you want to generate a single vector embedding for different data types.

For example, you can use this function in information-retrieval applications or chatbots, where you want to generate a query vector on the fly from a user's natural language text input string. You can then query a vector field with this query vector for a fast similarity search.

To generate an embedding, this function uses a vector embedding model (in ONNX format) that you load into the database.

> **Note:**
>
> If you want to generate embeddings by using third-party vector embedding models, then use Vector Utility PL/SQL packages. These packages let you work with both embedding models (in ONNX format) stored in the database and third-party embedding models (by calling third-party REST APIs).

For detailed information on this function, see VECTOR_EMBEDDING.

Related Topics

- Import Pretrained Models in ONNX Format
  Oracle Database 23ai includes an ONNX runtime engine for running embedding models directly inside the database. This section covers the process of importing existing pretrained embedding models into Oracle database, including converting those models into the ONNX format if they are not already converted.
- Generate Embeddings
  In these examples, you can see how to use the `VECTOR_EMBEDDING` SQL function or the `UTL_TO_EMBEDDING` PL/SQL function to generate a vector embedding from input text strings and images.

About PL/SQL Packages to Generate Embeddings

Choose to implement Vector Utility PL/SQL packages to perform chunking, embedding, and text generation operations along with text processing and similarity search, both within and outside the database. You can schedule these operations as end-to-end pipelines. The supplied PL/SQL packages for vector generation are `DBMS_VECTOR` and `DBMS_VECTOR_CHAIN`.

These packages can work with both vector embedding models in ONNX format (by importing these models into the database) and third-party vector embedding models (by calling third-party REST APIs). Each package is made up of subprograms, such as *chainable utility functions* and *vector helper procedures*.

- About Chainable Utility Functions and Common Use Cases
  These are intended to be a set of chainable and flexible "stages" through which you pass your input data to transform into a different representation, including vectors.
- About Vector Helper Procedures
  Vector helper procedures let you configure authentication credentials, preferences, and language-specific data for use in chainable utility functions.
- Supplied Vector Utility PL/SQL Packages
  Use either a lightweight `DBMS_VECTOR` package or a more advanced `DBMS_VECTOR_CHAIN` package with full capabilities.
- Terms of Using Vector Utility PL/SQL Packages
  You must understand the terms of using REST APIs that are part of Vector Utility PL/SQL packages.
- Validate JSON Input Parameters
  You can optionally validate the structure of your JSON input to the `DBMS_VECTOR.UTL` and `DBMS_VECTOR_CHAIN.UTL` functions, which use JSON to define their input parameters.

Related Topics

- Vector Generation Examples
  Run these end-to-end examples to see how you can generate vector embeddings, both within and outside the database.

About Chainable Utility Functions and Common Use Cases

These are intended to be a set of chainable and flexible "stages" through which you pass your input data to transform into a different representation, including vectors.

Supplied Chainable Utility Functions

You can combine a set of chainable utility (`UTL`) functions together in an end-to-end pipeline.

Each pipeline or **transformation chain** can include a single function or a combination of functions, which are applied to source documents as they are transformed into other representations (text, chunks, summary, or vector). These functions are chained together, such that the output from one function is used as an input for the next.

Each chainable utility function performs a specific task of transforming data into other representations, such as converting data to text, converting text to chunks, or converting the extracted chunks to embeddings.

At a high level, the supplied chainable utility functions include:

Function	Description	Input and Return Value
`UTL_TO_TEXT()`	Converts data (for example, Word, HTML, or PDF documents) to plain text.	Accepts the input as `CLOB` or `BLOB`. Returns a plain text version of the document as `CLOB`.
`UTL_TO_CHUNKS()`	Converts data to chunks.	Accepts the input as plain text (`CLOB` or `VARCHAR2`). Splits the data to return an array of chunks (`CLOB`).
`UTL_TO_EMBEDDING()`	Converts data to a single embedding.	Accepts the input as plain text (`CLOB`) or image (`BLOB`). Returns a single embedding (`VECTOR`).
`UTL_TO_EMBEDDINGS()`	Converts an array of chunks to an array of embeddings.	Accepts the input as an array of chunks (`VECTOR_ARRAY_T`). Returns an array of embeddings (`VECTOR_ARRAY_T`).
`UTL_TO_SUMMARY()`	Generates a concise summary for data, such as large or complex documents.	Accepts the input as plain text (`CLOB`). Returns a summary in plain text as `CLOB`.
`UTL_TO_GENERATE_TEXT()`	Generates text for prompts and images.	Accepts the input as text data (`CLOB`) for prompts, or as media data (`BLOB`) for media files such as images. Processes this information to return `CLOB` containing the generated text.
`RERANK()`	Reassesses and reorders an initial list of documents based on their similarity score.	Accepts the input as a query (`CLOB`) and a list of documents (`JSON`). Processes this information to return a `JSON` object containing a reranked list of documents, sorted by score.

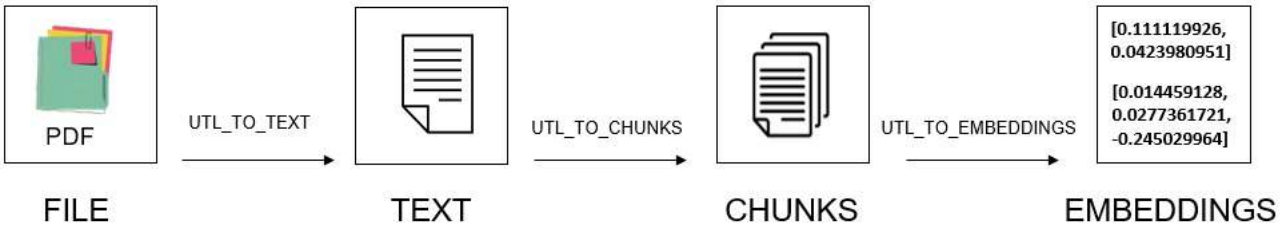
Sequence of Chains

Chainable utility functions are designed to be flexible and modular. You can create transformation chains in various sequences, depending on your use case.

For example, you can directly extract vectors from a large PDF file by creating a chain of the `UTL_TO_TEXT`, `UTL_TO_CHUNKS`, and `UTL_TO_EMBEDDINGS` chainable utility functions.

As shown in the following diagram, a *file-to-text-to-chunks-to-embeddings* chain performs a set of operations in this order:

1. Converts a PDF file to a plain text file by calling `UTL_TO_TEXT`.
2. Splits the resulting text into many appropriate-sized chunks by calling `UTL_TO_CHUNKS`.
3. Generates vector embeddings on each chunk by calling `UTL_TO_EMBEDDINGS`.



Common Use Cases

Let us look at some common use cases to understand how you can customize and apply these transformation chains:

Single-Step or Direct Transformation:

- **Document to vectors**:

  As discussed earlier, a common use case is to automatically generate vector embeddings from a set of documents. Embeddings are created by vector embedding models, and are numerical representations that capture the semantic meaning of your input data (as explained in Overview of Oracle AI Vector Search). The resulting embeddings are high-dimensional vectors, which you can use for text classification, question-answering, information retrieval, or text processing applications.

  – You can convert a set of documents to plain text, split the resulting text into smaller chunks to finally generate embeddings on each chunk, in a single *file-to-text-to-chunks-to-embeddings* chain. See Perform Chunking With Embedding.

  – For smaller documents or text strings, you can eliminate chunking (as explained in Understand the Stages of Data Transformations) and directly generate embeddings in a single *text-embeddings* chain. See Generate Embeddings.

- **Document to vectors, with chunking and summarization**:

  Another use case might be to generate a short summary of a document and then automatically extract vectors from that summary.

  After generating the summary, you can either generate a single vector (using `UTL_TO_EMBEDDING`) or chunk it and then generate multiple vectors (using `UTL_TO_EMBEDDINGS`).

  - You can convert the document to plain text, summarize the text into a concise gist to finally create a single embedding on the summarized text, in a *file-to-text-to-summary-to-embedding* chain.
  - You can convert the document to plain text, summarize the text into a gist, split the gist into chunks to finally create multiple embeddings on each summarized chunk, in a *file-to-text-to-summary-to-chunks-to-embeddings* chain.

  While both the chunking and summarization techniques make text smaller, they do so in different ways. Chunking just breaks the text into smaller pieces, whereas summarization extracts a salient meaning and context of that text into free-form paragraphs or bullet points.

  By summarizing the entire document and appending the summary to each chunk, you get the best of both worlds, that is, an individual piece that also has a high-level understanding of the overall document.

- **Image to vectors**:

  You can directly generate a vector embedding based on an image, which can be used for image classification, object detection, comparison of large datasets of images, or for a more effective similarity search on documents that include images. Embeddings are created by image embedding models or multimodal embedding models that extract each visual element of an image (such as shape, color, pattern, texture, action, or object) and accordingly assign a numerical value to each element.

  See Convert Image to Embedding Using Public REST Providers.

Step-by-Step or Parallel Transformation:

- **Text to vector**:

  A common use case might be information retrieval applications or chatbots, where you can convert a user's natural language text query string to a query vector on the fly. You can then compare that query vector against the existing vectors for a fast similarity search.

  You can vectorize a query vector and then run that query vector against your index vectors, in a *text-to-embedding* chain.

  See Convert Text String to Embedding Within Oracle Database, Convert Text String to Embedding Using Public REST Providers, and Convert Text String to Embedding Using the Local REST Provider Ollama.

- **Text to chunks**:

  Another use case might be to build a standalone Text Chunking system to break down a large amount of text into smaller yet semantically meaningful pieces, in a *text-to-chunks* chain.

  This method also gives you more flexibility to experiment with your chunks, where you can create, inspect, and accordingly amend the chunking results and then proceed further.

  See Convert Text to Chunks With Custom Chunking Specifications and Convert File to Text to Chunks to Embeddings Within Oracle Database.

- **Prompt or media file to text**:

  You can communicate with Large Language Models (LLMs) to perform several language-related tasks, such as text generation, translation, summarization, or Q&A. You can input text data in natural language (for prompts) or media data (for images) to an LLM-powered chat interface. LLM then processes this information to generate a text response.

  - **Prompt to text**:

A prompt can be an input text string, such as a question that you ask an LLM. For example, "`What is Oracle Text?`". A prompt can also be a set of instructions or a command, such as "`Summarize the following ...`", "`Draft an email asking for ...`", or "`Rewrite the following ...`". The LLM responds with a textual answer, description, or summary based on the specified task in the prompt.

See Generate Text Using Public REST Providers and Generate Text Using the Local REST Provider Ollama.

- **Image to text**:

  You can also prompt with a media file (such as an image) to generate a textual analysis or description of the contents of the image, which can then be used for image classification, object detection, or similarity search.

  Here, you additionally supply a text question as the prompt along with the image. For example, the prompt can be "`What is this image about?`" or "`How many birds are there in this image?`".

  See Describe Images Using Public REST Providers.

- **Text to summary**:

  You can build a standalone Text Summarization system to generate a summary of large or complex documents, in a *text-to-summary* chain.

  This method also gives you more flexibility to experiment with your summaries, where you can create, inspect, and accordingly amend the summarization results and then proceed further.

  See Generate Summary Using Public REST Providers.

  > **Note:**
  >
  > In addition to using external LLMs for summarization, you can use Oracle Text to generate a gist (summary) within the database. See UTL_TO_SUMMARY.

- **RAG implementation**

  A prompt can include results from a search, through the implementation of Retrieval Augmented Generation (RAG). Using RAG can mitigate inaccuracies and hallucinations, which may occur in generated responses when using LLMs.

  See SQL RAG Example.

  You can additionally utilize reranking to enhance the quality of information ingested into an LLM. See Use Reranking for Better RAG Results.

Schedule Vector Utility Packages

Some of the transformation chains may take a long time depending on your workload and implementation, thus you can schedule to run Vector Utility PL/SQL packages in the background.

The `DBMS_SCHEDULER` PL/SQL package helps you effectively schedule these packages, without manual intervention.

For information on how to create, run, and manage jobs with Oracle Scheduler, see *Oracle Database Administrator's Guide*.

About Vector Helper Procedures

Vector helper procedures let you configure authentication credentials, preferences, and language-specific data for use in chainable utility functions.

At a high level, the supplied vector helper procedures include:

- **Credential helper procedures** to securely manage authentication credentials, which are used to access third-party providers when making REST API calls.

Function	Description
`CREATE_CREDENTIAL`	Creates a credential name for securely storing user authentication credentials in Oracle Database.
`DROP_CREDENTIAL`	Drops an existing credential name.

- **Preference helper procedures** to manage vectorizer preferences, which are used when creating or altering hybrid vector indexes.

Function	Description
`CREATE_PREFERENCE`	Creates a vectorizer preference in the database.
`DROP_PREFERENCE`	Drops an existing vectorizer preference from the database.

- **Chunker helper procedures** to manage custom vocabulary and language data, which are used when chunking user data.

Function	Description
`CREATE_VOCABULARY`	Loads your own vocabulary file into the database.
`DROP_VOCABULARY`	Removes the specified vocabulary data from the database.
`CREATE_LANG_DATA`	Loads your own language data file (abbreviation tokens) into the database.
`DROP_LANG_DATA`	Removes abbreviation data for a given language from the database.

Related Topics

- Vector Search PL/SQL Packages
  The `DBMS_VECTOR`, `DBMS_VECTOR_CHAIN`, and `DBMS_HYBRID_VECTOR` PL/SQL APIs are available to support Oracle AI Vector Search capabilities.
- Text Processing Views
  These views display language-specific data (abbreviation token details) and vocabulary data related to the Oracle AI Vector Search SQL and PL/SQL utilities.

Supplied Vector Utility PL/SQL Packages

Use either a lightweight `DBMS_VECTOR` package or a more advanced `DBMS_VECTOR_CHAIN` package with full capabilities.

- `DBMS_VECTOR`:

  This package simplifies common operations with Oracle AI Vector Search, such as chunking text into smaller segments, extracting vector embeddings from user data, or generating text for a given prompt.

Subprogram	Operation	Provider	Implementation
Chainable Utility Functions	`UTL_TO_CHUNKS` to perform chunking	Oracle Database	Calls the `VECTOR_CHUNKS` SQL function under the hood
	`UTL_TO_EMBEDDING` and `UTL_TO_EMBEDDINGS` to generate one or more embeddings	Oracle Database	Calls the embedding model in ONNX format stored in the database
		Third-party REST providers	Calls the specified third-party embedding model
	`UTL_TO_GENERATE_TEXT` to generate text for prompts and images	Third-party REST providers	Calls the specified third-party text generation model
	`RERANK` to retrieve more relevant search output	Third-party REST providers	Calls the specified third-party embedding model
Credential Helper Procedures	`CREATE_CREDENTIAL` and `DROP_CREDENTIAL` to manage credentials for third-party service providers	Oracle Database	Stores credentials securely for use in Chainable Utility Functions

For detailed information on this package, see DBMS_VECTOR.

- `DBMS_VECTOR_CHAIN`:

  This package provides chunking and embedding functions along with some text generation and summarization capabilities. It is more suitable for text processing with similarity search and hybrid search, using functionality that can be pipelined together for an end-to-end search.

Subprogram	Operation	Provider	Implementation
Chainable Utility Functions	`UTL_TO_TEXT` to extract plain text data from documents	Oracle Database	Uses the Oracle Text component (`CONTEXT`) of Oracle Database
	`UTL_TO_CHUNKS` to perform chunking	Oracle Database	Calls the `VECTOR_CHUNKS` SQL function under the hood
	`UTL_TO_EMBEDDING` and `UTL_TO_EMBEDDINGS` to generate one or more embeddings	Oracle Database	Calls the embedding model in ONNX format stored in the database
		Third-party REST providers	Calls the specified third-party embedding model
	`UTL_TO_SUMMARY` to generate summaries	Oracle Database	Uses Oracle Text
		Third-party REST providers	Calls the specified third-party text summarization model
	`UTL_TO_GENERATE_TEXT` to generate text for prompts and images	Third-party REST providers	Calls the specified third-party text generation model
	`RERANK` to retrieve more relevant search output	Third-party REST providers	Calls the specified third-party embedding model
Credential Helper Procedures	`CREATE_CREDENTIAL` and `DROP_CREDENTIAL` to manage credentials for third-party service providers	Oracle Database	Stores credentials securely for use in Chainable Utility Functions

Subprogram	Operation	Provider	Implementation
Preference Helper Procedures	`CREATE_PREFERENCE` and `DROP_PREFERENCE` to manage preferences for hybrid vector indexes	Oracle Database	Creates vectorizer preferences for use in hybrid vector indexing pipelines
Chunker Helper Procedures	`CREATE_VOCABULARY` and `DROP_VOCABULARY` to manage custom token vocabularies	Oracle Database	Uses Oracle Text
	`CREATE_LANG_DATA` and `DROP_LANG_DATA` to manage language-specific data (abbreviation tokens)	Oracle Database	Uses Oracle Text

> **Note:**
>
> The `DBMS_VECTOR_CHAIN` package requires you to install the `CONTEXT` component of Oracle Text, an Oracle Database technology that provides indexing, term extraction, text analysis, text summarization, word and theme searching, and other utilities.
>
> Due to underlying dependance on the text processing capabilities of Oracle Text, note that both the `UTL_TO_TEXT` and `UTL_TO_SUMMARY` chainable utility functions and all the chunker helper procedures are available only in this package through Oracle Text.

For detailed information on this package, see DBMS_VECTOR_CHAIN.

Related Topics

- Supported Third-Party Provider Operations and Endpoints
  Review a list of third-party REST providers and REST endpoints that are supported for various vector generation, summarization, text generation, and reranking operations.

Terms of Using Vector Utility PL/SQL Packages

You must understand the terms of using REST APIs that are part of Vector Utility PL/SQL packages.

Some of the Vector Utility PL/SQL APIs enable you to perform embedding, summarization, and text generation operations outside Oracle Database, by using third-party REST providers (such as Cohere, Google AI, Hugging Face, Generative AI, OpenAI, or Vertex AI).

> **WARNING:**
>
> Certain features of the database may allow you to access services offered separately by third-parties, for example, through the use of JSON specifications that facilitate your access to REST APIs.
>
> Your use of these features is solely at your own risk, and you are solely responsible for complying with any terms and conditions related to use of any such third-party services. Notwithstanding any other terms and conditions related to the third-party services, your use of such database features constitutes your acceptance of that risk and express exclusion of Oracle's responsibility or liability for any damages resulting from such access.

Validate JSON Input Parameters

You can optionally validate the structure of your JSON input to the `DBMS_VECTOR.UTL` and `DBMS_VECTOR_CHAIN.UTL` functions, which use JSON to define their input parameters.

The JSON data is schemaless, so the amount of validation that Vector Utility package APIs do at runtime is minimal for better performance. The APIs validate only the mandatory JSON parameters, that is, the parameters that you supply for the APIs to run (not optional JSON parameters and attributes).

Before calling an API, you can use subprograms in the `DBMS_JSON_SCHEMA` package to test whether the input data to be specified in the `PARAMS` clause is valid with respect to a given JSON schema. This offers more flexibility and also ensures that only schema-valid data is inserted in a JSON column.

Validate JSON input parameters for the `DBMS_VECTOR.UTL` and `DBMS_VECTOR_CHAIN.UTL` functions against the following schema:

- **For the Database Provider**:
  - `SCHEMA_CHUNK`

```
{ "title" : "utl_to_chunks",
  "description" : "Chunk parameters",
  "type" : "object",
  "properties" : {
    "by"           : {"type" : "string", "enum" : [ "chars",
"characters", "words", "vocabulary" ]   },
    "max"          : {"type" : "string", "pattern" : "^[1-9][0-9]*$" },
    "overlap"      : {"type" : "string", "pattern" : "^[0-9]+$" },
    "split"        : {"type" : "string", "enum" : [ "none", "newline",
"blankline", "space", "recursively", "custom" ]  },
    "vocabulary"   : {"type" : "string"  },
    "language"     : {"type" : "string"  },
    "normalize"    : {"type" : "string", "enum" : [ "all", "none",
"options" ]  },
      "norm_options" : {"type" : "array",  "items": {  { "type":
"string", "enum": ["widechar", "whitespace", "punctuation"] }  },
    "custom_list"  : {"type" : "array",  "items": { "type":
"string" }  },
    "extended"     : {"type" : "boolean" } },
```

```
  "additionalProperties" : false
}
```

– SCHEMA_VOCAB

```
{ "title" : "create_vocabulary",
  "description" : "Create vocabulary parameters",
  "type" : "object",
  "properties" : {
    "table_name"      : {"type" : "string" },
    "column_name"     : {"type" : "string" },
    "vocabulary_name" : {"type" : "string" },
    "format"          : {"type" : "string", "enum" : [ "BERT", "GPT2",
"XLM" ]  },
    "cased"           : {"type" : "boolean" } },
  "additionalProperties" : false,
  "required" : [ "table_name", "column_name", "vocabulary_name" ]
}
```

– SCHEMA_LANG

```
{ "title" : "create_lang_data",
  "description" : "Create language data parameters",
  "type" : "object",
  "properties" : {
    "table_name"      : {"type" : "string" },
    "column_name"     : {"type" : "string" },
    "preference_name" : {"type" : "string" },
    "language"        : {"type" : "string" } },
  "additionalProperties" : false,
  "required" : [ "table_name", "column_name", "preference_name",
"language" ]
}
```

– SCHEMA_TEXT

```
{ "title": "utl_to_text",
  "description": "To text parameters",
  "type" : "object",
  "properties" : {
    "plaintext"       : {"type" : "boolean" },
    "charset"         : {"type" : "string", "enum" : [ "UTF8" ] } },
  "additionalProperties": false
}
```

– SCHEMA_DBEMB

```
{ "title" : "utl_to_embedding",
  "description" : "To DB embeddings parameters",
  "type" : "object",
  "properties" : {
    "provider"  : {"type" : "string" },
    "model"     : {"type" : "string" } },
  "additionalProperties": true,
```

```
    "required" : [ "provider", "model" ]
  }
```

– `SCHEMA_SUM`

```
  { "title" : "utl_to_summary",
    "description" : "To summary parameters",
    "type" : "object",
    "properties" : {
      "provider"       : {"type" : "string" },
      "numParagraphs"  : {"type" : "number" },
      "language"       : {"type" : "string" },
      "glevel"         : {"type" : "string" }
     },
    "additionalProperties" : true,
    "required" : [ "provider" ]
  }
```

- **For REST Providers**:

  `SCHEMA_REST`

```
{  "title" : "REST parameters",
   "description" : "REST versions of utl_to_embedding, utl_to_summary,
utl_to_generate_text",
   "type" : "object",
   "properties" : {
     "provider"         : {"type" : "string" },
     "credential_name"  : {"type" : "string" },
     "url"              : {"type" : "string" },
     "model"            : {"type" : "string" }
    },
   "additionalProperties" : true,
   "required" : [ "provider", "credential_name", "url", "model" ] }
```

  Note that all the REST calls to third-party service providers share the same schema for their respective embedding, summarization, and text generation operations.

Examples:

- To validate your JSON data against JSON schema, use the PL/SQL function or procedure `DBMS_JSON_SCHEMA.is_valid()`.

  The function returns `1` for valid and `0` for invalid (invalid data can optionally raise an error). The procedure returns `TRUE` for valid and `FALSE` for invalid as the value of an `OUT` parameter.

```
l_valid := sys.DBMS_JSON_SCHEMA.is_valid(params, json(SCHEMA),
dbms_json_schema.RAISE_ERROR);
```

- If you use the procedure (not function) `is_valid`, then you have access to the validation errors report as an `OUT` parameter. This use of the procedure checks data against schema, providing output in parameters validity (`BOOLEAN`) and errors (`JSON`).

```
sys.DBMS_JSON_SCHEMA.is_valid(params, json(SCHEMA), l_valid, l_errors);
```

- To read a detailed validation report of errors, use the PL/SQL procedure `DBMS_JSON_SCHEMA.validate_report`.

  If you use the function (not the procedure) `is_valid`, then you do not have access to such a report. Instead of using the function `is_valid`, you can use the PL/SQL function `DBMS_JSON_SCHEMA.validate_report` in a SQL query to validate and return the same full validation information that the reporting `OUT` parameter of the procedure `is_valid` provides, as a JSON type instance.

  ```
  SELECT JSON_SERIALIZE(DBMS_JSON_SCHEMA.validate_report('json',SCHEMA)
  returning varchar2 PRETTY);
  ```

Related Topics

- DBMS_VECTOR
  The `DBMS_VECTOR` package simplifies common operations with Oracle AI Vector Search, such as extracting chunks or embeddings from user data, generating text for a given prompt or an image, creating a vector index, or reporting on index accuracy.
- DBMS_VECTOR_CHAIN
  The `DBMS_VECTOR_CHAIN` package enables advanced operations with Oracle AI Vector Search, such as chunking and embedding data along with text generation and summarization capabilities. It is more suitable for text processing with similarity search and hybrid search, using functionality that can be pipelined together for an end-to-end search.
- DBMS_JSON_SCHEMA

Import Pretrained Models in ONNX Format

Oracle Database 23ai includes an ONNX runtime engine for running embedding models directly inside the database. This section covers the process of importing existing pretrained embedding models into Oracle database, including converting those models into the ONNX format if they are not already converted.

Open Neural Network Exchange or ONNX is an open standard format of machine learning models. Consider the following use cases:

- Using complex media input such as text or image for similarity search
- Perform text classification
- Perform reranking

You need vector embedding of the media input to perform all the tasks mentioned above. ONNX pipeline models allows you to convert text and/or image models into ONNX format if they are not already in ONNX format, import the ONNX format models into Oracle Database, and generate embeddings from your data within the database. The ONNX format models imported to the database could be used for tasks such as search and classification.

Pretrained models are models that are already trained on a media data (text, image, etc.) and saved to a storage format for future use. Hugging Face is the most popular platform that hosts pretrained models typically created with PyTorch.

The Python package `oml.utils` contains three classes: `ONNXPipeline`, `ONNXPipelineConfig`, and `MiningFunction`. The package handles ONNX pipeline generation and export, while also incorporating the necessary pre- and post-processing steps.

- **ONNXPipeline** : Allows you to import a pretrained model (Your own pretrained model or one of the supported models from Hugging Face)

- **ONNXPipelineConfig** : Allows you to configure the attributes of pretrained model (Your own pretrained model or one of the supported models from Hugging Face)
- **MiningFunction**: Allows you to choose from one of the following mining options:
  - `EMBEDDING` : Corresponds to text and image embedding
  - `CLASSIFICATION` : Corresponds to text classification
  - `REGRESSION` : Corresponds to re-ranking

> **WARNING:**
>
> EmbeddingModel and EmbeddingModelConfig are deprecated. Instead, please use ONNXPipeline and ONNXPipelineConfig respectively. The details of the deprecated classes can be found in Python Classes to Convert Pretrained Models to ONNX Models (Deprecated). If a you choose to use a deprecated class, a warning message will be shown indicating that the classes will be removed in the future and advising the user to switch to the new class.

The ONNX pipeline models are available for text embedding, image embedding, multi-modal embedding, text classification and re-ranking.

- ONNX Pipeline Models : Text Embedding
  ONNX pipeline models provides text embedding models that accepts text as input and produces embeddings as output. The pipeline models also provide the necessary pre-processing and post-processing needed for the text.
- ONNX Pipeline Models : Image Embedding
  ONNX pipeline models provides image embedding models that accepts image as an input and produces embeddings as output. The pipeline models also provide the necessary pre-processing.
- ONNX Pipeline Models: CLIP Multi-Modal Embedding
  ONNX pipeline models provides multi-modal embedding models that accepts both image and text as an input and produces embeddings as output. The pipeline models also provide the necessary pre-processing needed.
- ONNX Pipeline Models: Text Classification
  ONNX pipeline models provides text classification models that accepts text strings as input and produces the probablity of the input string belonging to a specific label. The pipeline models also provide the necessary pre-processing and post-processing.
- ONNX Pipeline Models: Reranking Pipeline
  ONNX pipeline models provide a reranking pipeline that calculates similarity score for a given pair of texts.
- Convert Pretrained Models to ONNX Model: End-to-End Instructions for Text Embedding
  This section provides end-to-end instructions from installing the OML4Py client to downloading a pretrained embedding model in ONNX-format using the Python utility package offered by Oracle.
- Import ONNX Models into Oracle Database End-to-End Example
  Learn to import a pretrained embedding model that is in ONNX format and generate vector embeddings.

ONNX Pipeline Models : Text Embedding

ONNX pipeline models provides text embedding models that accepts text as input and produces embeddings as output. The pipeline models also provide the necessary pre-processing and post-processing needed for the text.

> **Note:**
>
> - This API is updated for 23.7. The version used for 23.6 and below used python packages EmbeddingModel and EmbeddingModelConfig. These packages are replaced with ONNXPipeline and ONNXPipelineConfig respectively. Oracle recommends that you use the latest version. You can find the details of the deprecated python classes in Python Classes to Convert Pretrained Models to ONNX Models (Deprecated).
> - This feature will **only** work on OML4Py 2.1 client. It is not supported on the OML4Py server.
> - In-database embedding models must include tokenization and post-processing. Providing only the core ONNX model to `DBMS_VECTOR.LOAD_ONNX_MODEL` is insufficient because you need to handle tokenization externally, pass tensors into the SQL operator, and convert output tensors into vectors.
> - Oracle is providing a Hugging Face all-MiniLM-L12-v2 model in ONNX format, available to download directly to the database using `DBMS_VECTOR.LOAD_ONNX_MODEL`. For more information, see the blog post Now Available! Pre-built Embedding Generation model for Oracle Database 23ai.

If you do not have a pretrained embedding model in ONNX-format to generate embeddings for your data, Oracle offers a Python package that downloads pretrained models from an external source, converts the model to ONNX format augmented with pre-processing and post-processing steps, and imports the resulting ONNX-format model into Oracle Database. Use the `DBMS_VECTOR.LOAD_ONNX_MODEL` procedure or `export2db()` function to import the file as a mining model. (`eport2db()` is a method on the ONNXPipeline object). Then leverage the in-database ONNX Runtime with the ONNX model to produce vector embeddings.

At a high level, the Python package performs the following tasks:

- Downloads the pretrained model from external source (example: from Hugging Face repository) to your system
- Augments the model with pre-processing and post-processing steps and creates a new ONNX model
- Validates the augmented ONNX model
- Loads into the database as a data mining model or optionally exports to a file

The Python package can take any of the models in the preconfigured list as input. Alternatively, you can use the built-in template that contains common configurations for certain groups of models such as text or image models. To understand what a preconfigured list, what a built-in template is, and how to use them, read further.

Limitations

This table describes the limitations of the Python package.

> **Note:**
>
> This feature is available with the OML4Py 2.1 client only.

Parameter	Description
`Transformer Model Type`	Supports transformer models that supports text embedding.
`Model Size`	Model size should be less than 1GB. Quantization can help reduce the size.
`Tokenizers`	Must be either `BERT`, `GPT2`, `SENTENCEPIECE`, `CLIP`, or `ROBERTA`.

Preconfigured List of Models

Preconfigured list of models are common models from external resource repositories that are provided with the Python package. The preconfigured models have an existing specification. Users can create their own specification using the text template as a starting point. To get a list of all model names in the preconfigured list, you can use the `show_preconfigured` function.

Templates

The Python package provides built-in text template for you to configure the pretrained models with pre-processing and post-processing operations. The template has a default specification for the pretrained models. This specification can be changed or augmented to create custom configurations. The text template uses Mean Pooling and Normalization as post-processing operations by default.

End to End Example for Converting Text Models to ONNX Format and Storing them as a File or in a Database:

To use the Python package `oml.utils`, ensure that you have the following:

- OML4Py 2.1 Client running on Linux X64 for On-Premises Databases
- Python 3.12 (the earlier versions are not compatible)

.

1. Start Python in your work directory.

```
python3


Python 3.12.6 | (main, Feb 27 2024, 17:35:02) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
```

2. On the OML4Py client, load the Python classes:

```
from oml.utils import ONNXPipeline, ONNXPipelineConfig
```

3. You can get a list of all preconfigured models by running the following:

```
ONNXPipelineConfig.show_preconfigured()
```

4. To get a list of available templates:

```
ONNXPipelineConfig.show_templates()
```

5. Choose from:

- Generating a text pipeline using a preconfigured model "sentence-transformers/all-MiniLM-L6-v2" and save it in a local directory:

```
#generate from preconfigured model "sentence-transformers/all-MiniLM-L6-
v2"
pipeline = ONNXPipeline(model_name="sentence-transformers/all-MiniLM-L6-
v2")
pipeline.export2file("your_preconfig_file_name",output_dir=".")
```

- Generating a text pipeline using a preconfigured model "sentence-transformers/all-MiniLM-L6-v2" and save it in the database:

```
#generate from preconfigured model "sentence-transformers/all-MiniLM-L6-
v2"
pipeline = ONNXPipeline(model_name="sentence-transformers/all-MiniLM-L6-
v2")
pipeline.export2db("your_preconfig_file_name")
```

> **Note:**
>
> In order for export2db( ) to succeed, a connection is required to the database. It is assumed that you have provided your credentials in the following code and the connection is successful.
>
> ```
> import oml
> oml.connect(user="username",password="password",dsn="dsn");
> ```

- Generating a text pipeline with a template and save it in a local directory

```
from oml.utils import ONNXPipeline, ONNXPipelineConfig
config   = ONNXPipelineConfig.from_template("text", max_seq_length=256,
           distance_metrics=["COSINE"], quantize_model=True)
pipeline = ONNXPipeline(model_name="sentence-transformers/all-MiniLM-L6-
v2",
           config=config)
pipeline.export2file("your_preconfig_model_name", output_dir=".")
```

Let's understand the code:

`from oml.utils import ONNXPipeline, ONNXPipelineConfig`

This line imports two classes, `ONNXPipeline` and `ONNXPipelineConfig`.

In the preconfigured models first example, where the ONNX format model is saved in a particular directory:

- `pipeline = ONNXPipeline(model_name="sentence-transformers/all-MiniLM-L6-v2")` creates an instance of the `ONNXPipeline` class, loading a pretrained model specified by the `model_name` parameter. `pipeline` is the text embedding model object. `sentence-transformers/all-MiniLM-L6-v2` is the model name for computing sentence embeddings. This is the model name under Hugging Face. Oracle supports models from Hugging Face.
- The `export2file` command creates an ONNX format model with a user-specified model name in the file. `your_preconfig_file_name` is a user defined ONNX model file name.
- `output_dir="."` specifies the output directory where the file will be saved. The `"."` denotes the current directory (that is, the directory from which the script is running).

In the preconfigured models second example, where the ONNX model is saved in the databse:

- `pipeline = ONNXPipeline(model_name="sentence-transformers/all-MiniLM-L6-v2")` creates an instance of the `ONNXPipleline` class, loading a pretrained model specified by the `model_name` parameter. `pipeline` is the text embedding model object. `sentence-transformers/all-MiniLM-L6-v2` is the model name for computing sentence embeddings. This is the model name under Hugging Face. Oracle supports models from Hugging Face.
- The `export2db` command creates an ONNX format model with a user defined model name in the database. `your_preconfig_model_name` is a user defined ONNX model name.

In the template example:

- `ONNXPipelineConfig.from_template("text", max_seq_length=256 distance_metrics=["COSINE"], quantize_model=True)`: This line creates a configuration object for an embedding model using a method called `from_template`. The `"text"` argument indicates the name of the template. The `max_seq_length=512` parameter specifies the maximum length of input to the model as number of tokens. There is no default value. Specify this value for models that are not preconfigured.
- `pipeline = ONNXPipeline(model_name="sentence-transformers/all-MiniLM-L6-v2", config=config)` initializes an `ONNXPipeliene` instance with a specific model and the previously defined configuration. The `model_name="all-MiniLM-L6-v2"` argument specifies the name or identifier of the pretrained model to be loaded.
- The `export2file` command creates an ONNX format model with a user defined model name in the file. `your_preconfig_model_name` is a user defined ONNX model name.
- `output_dir="."` specifies the output directory where the file will be saved. The `"."` denotes the current directory (that is, the directory from which the script is running).

> **Note:**
>
> - The model size is limited to 1 gigabyte. For models larger than 400MB, Oracle recommends quantization.
>
> **Quantization** reduces the model size by converting model weights from high-precision representation to low-precision format. The quantization option converts the weights to INT8. The smaller model size enables you to cache the model in shared memory further improving the performance.
> - The `.onnx` file is created with opset version 17 and ir version 8. For more information about these version numbers, see https://onnxruntime.ai/docs/reference/compatibility.html#onnx-opset-support.

6. Exit Python.

```
exit()
```

7. Inspect if the converted models are present in your directory.

> **Note:**
>
> ONNX files are only created when `export2file` is used. If `export2db` is used, no local ONNX files will be generated and the model will be saved in the database.

List all files in the current directory that have the extension `onnx`.

```
ls -ltr *.onnx
```

The Python utility package validates the embedding text model before you can run them using ONNX Runtime. Oracle supports ONNX embedding models that conform to `string` as input and `float32 [<vector dimension>]` as output.

If the input or output of the model doesn't conform to the description, you receive an error during the import.

8. Choose from:
   - Load the ONNX model to your database using PL/SQL:
     a. You can load the ONNX model using this syntax:

```
BEGIN
   DBMS_VECTOR.LOAD_ONNX_MODEL(
     directory => 'ONNX_IMPORT',
     file_name => 'all-MiniLM-L6.onnx',
     model_name => 'ALL_MINILM_L6');
END;
```

b. Move the ONNX file named `your_template_file_name` to a directory on the database server, and create a directory on the file system and in the database for the import.

```
mkdir -p /tmp/models
sqlplus / as sysdba
alter session set container=ORCLPDB;
```

Apply the necessary permissions and grants. In this example, we are using a pluggable database named ORCLPDB.

```
-- directory to store ONNX files for import
CREATE DIRECTORY ONNX_IMPORT AS '/tmp/models';
-- grant your OML user read and write permissions on the directory
GRANT READ, WRITE ON DIRECTORY ONNX_IMPORT to OMLUSER;
-- grant to allow user to import the model
GRANT CREATE MINING MODEL TO OMLUSER;
```

- Load the ONNX model to your database using Python:

9. Verify the model is in the database:

At your operating system prompt, start SQL*Plus, connect to it :

```
sqlplus OML_USER/password@pdbname_medium;
```

```
SQL*Plus: Release 23.0.0.0.0 - Production on Wed May 1 15:33:29 2024
Version 23.4.0.24.05

Copyright (c) 1982, 2024, Oracle. All rights reserved.

Last Successful login time: Wed May 01 2024 15:27:06 -07:00

Connected to:
Oracle Database 23ai Enterprise Edition Release 23.0.0.0.0 - Production
Version 23.4.0.24.05
```

List the ONNX model in the database to make sure it was loaded:

```
SQL> select MODEL_NAME, ALGORITHM from user_mining_models WHERE
MODEL_NAME='YOUR_PRECONFIG_MODEL_NAME';
```

```
MODEL_NAME ALGORITHM
-----------------------------------
YOUR_PRECONFIG_MODEL_NAME ONNX
```

`DBMS_VECTOR.LOAD_ONNX_MODEL` is only needed if `export2file` was used to save the ONNX model file to the local system instead of using `export2db` to save the model in the database. The `DBMS_VECTOR.LOAD_ONNX_MODEL` imports the ONNX format model into the Oracle Database to leverage the in-database ONNX Runtime to produce vector embeddings using the `VECTOR_EMBEDDING` SQL operator.

> **See Also:**
>
> - *Oracle Database SQL Language Reference* for information about the `VECTOR_EMBEDDING` SQL function
> - *Oracle Database PL/SQL Packages and Types Reference* for information about the `IMPORT_ONNX_MODEL` procedure
> - *Oracle Database PL/SQL Packages and Types Reference* for information about the `LOAD_ONNX_MODEL` procedure
> - *Oracle Machine Learning for SQL Concepts* for more information about importing pretrained embedding models in ONNX format and generating vector embeddings
> - https://onnx.ai/onnx/intro/ for ONNX documentation

ONNX Pipeline Models : Image Embedding

ONNX pipeline models provides image embedding models that accepts image as an input and produces embeddings as output. The pipeline models also provide the necessary pre-processing.

Image Embedding Pipeline

1. **Input** : Input to the pipeline is an image. Each image is represented as an array of bytes. This can be a BLOB when using SQL or a similar in-memory byte representation like `io.ByetIO` in Python.
2. **Pre-Processing** : The image embedding pipeline utilizes an Image Processor in the pre-processing step. The Image Processor is dependent on the model configuration in the Hugging Face repository. When you provide a pretrained image model (e.g. "google/vit-base-patch16-224"), the pipeline builder will look up the Image Processor class from the model's configuration and determine if it's supported by looking up the Image Processor name in pipeline builder templates. If a match is found then the specific Image Processor is loaded and configured according to the template. The following table shows the relationship between pipeline builder templates and their corresponding Image Processor classes. Image Processor classes are provided by the transformer library.

> **Note:**
>
> You do not need to reference these templates when using preconfigured image models. However any non-preconfigured model is template driven and these templates can be used for such models.

Template	Image Processor Class
Image_ViT	transformers.ViTImageProcessor
Image_ConvNext	transformers.ConvNextImageProcessor

Each Image Processor has several possible operations that will modify the input image in a specific way. Each of these operations is represented by a configuration in the Image

Processor template. The modification of these operations and their respective configuration is not supported. You are able to view the configurations and should be aware of them. The following table shows the image operations and their respective configurations:

Table 4-1 Image Processor Configurations

Image Processor Operation	Description
Decode	Converts the compressed image to 3-D raster format.
Resize	Resizes the given image to the new shape.
Rescale	Rescales (element wise multiplication) an image with a numeric value.
Normalize	Normalizes (adjusts the intensity values of pixels to a desired range, often between 0 to 1) an image using the given mean and standard deviation (std) using the following formulation: *(image - mean) / std*
CenterCrop	Crops the image with a fixed size from the center.

> **Note:**
>
> The Image pre-processors are considered to be fixed and modifications are not allowed. We are exposing these details for your information and understanding.

3. **Original Model**: Pre-trained pytorch model from Hugging Face repository. The repository also contain preconfigured models which have an existing specification. To get a list of all model names in the preconfigured list, you can use the `show_preconfigured` function. You can also create your own specification using the above mentioned templates as a starting point.
4. **Output** : The pipeline generates embeddings for each input image.

Image Embedding Examples

1. Exporting a pre-configured image model:

```
from oml.utils import ONNXPipeline
pipeline =ONNXPipeline("google/vit-base-patch16-224")
pipeline.export2file("your_preconfig_model_name")
```

This example uses the `google/vit-base-patch16-224` model, which is a pre-configured model shipped in OML4Py 2.1. The pipeline is exported to a file which will result in a file **your_preconfig_model_name.onnx** in the current directory. This model can optionally be exported directly to the database using the pipeline.export2db function. Although the exported pipeline will just produce embeddings, not a classification output, it is possible to train an OML classification model to take these embeddings and produce the desired classification.

2. Exporting a non pre-configured model with a template:

> **Note:**
>
> Refer to this section for understanding the templates

```
from oml.utils import ONNXPipeline,ONNXPipelineConfig
import oml
config = ONNXPipelineConfig.from_template("Image_ViT")
pipeline = ONNXPipeline("nateraw/food",config=config)
oml.connect("pyquser","pyquser",dsn="pydsn")
oml.export2db("your_preconfig_model_name")
```

In this example, since the `nateraw/food` model is not included as a preconfigured model in OML4Py 2.1, a template approach was chosen. Since this is a ViT based model, the ViT template is selected.

The process of loading a image model converted to a ONNX format is very similar to the ONNX format model generated from text. Refer to steps 6-9 from ONNX Pipeline Models : Text Embedding for understanding how to load a ONNX model and use it further.

ONNX Pipeline Models: CLIP Multi-Modal Embedding

ONNX pipeline models provides multi-modal embedding models that accepts both image and text as an input and produces embeddings as output. The pipeline models also provide the necessary pre-processing needed.

CLIP Multi-Modal Embedding Pipeline

CLIP models are multi-modal which allows you to perform embedding on both text and image input data. The main advantage of this model is to do image-text similarity. You can compare vectors related to text snippets and a given image, to understand which text can better describe given image. The pipeline generator will generate two ONNX pipeline models for a pretrained CLIP model, distinguished by their suffixes. The pipeline model for images is suffixed with **_img** and the model for text is suffixed with **_txt**. The same models with their suffixes will be loaded to the database when using *export2db*. For performing CLIP related tasks such as image-text similarity, both models will need to be used at inference time.

1. **Input**: CLIP models consist of two pipelines: an image embedding pipeline and a text embedding pipeline. The Image pipeline takes images as described in the Image Embedding Pipeline section of ONNX Pipeline Models : Image Embedding, and the text Pipeline takes text as described in ONNX Pipeline Models : Text Embedding (Text Embedding support was introduced in OML4Py 2.0).
2. **Pre-processing**: The image pipeline for CLIP models utilizes the same pre-processing strategy as described in the Image Embedding Pipeline section of ONNX Pipeline Models : Image Embedding. That is, an image processor that matches the model's configuration is utilized to prepare the images. The text pipeline utilizes a specific tokenizer: the **CLIPTokenizer (transformers.models.clip.CLIPTokenizer)**.
3. **Post-processing**: As a post-processing step, Normalization is added to both the image and text pipelines.
4. **Output**: Both models will produce vectors of the same shape that can then be compared using some similarity measure.

CLIP Multi-Modal Embedding Examples

1. Exporting a pre-configured image model to a file:
   The following example will produce two pipelines called **clip_img.onnx** and **clip_txt.onnx**, which can be used for image and text embeddings respectively.

   ```
   from oml.utils import ONNXPipeline
   pipeline = ONNXPipeline("openai/clip-vit-large-patch14")
   pipeline.export2file("clip")
   ```

2. Exporting a pre-configured image model to the database:
   This example will produce two in-database models called **clip_img** and **clip_txt**, which can be used for image and text embeddings respectively.

   ```
   fromoml.utils import ONNXPipeline
   import oml
   pipeline = ONNXPipeline("openai/clip-vit-large-patch14")
   oml.connect("pyquser","pyquser",dsn="pydsn")
   pipeline.export2db("clip")
   ```

3. Exporting a non pre-configured with a template to a file:
   This example will work for clip models that are not preconfigured. It will create two files called **clip_14_img.onnx** and **clip_14_txt.onnx**.

   ```
   from oml.utils import ONNXPipeline, ONNXPipelineConfig
   config = ONNXPipelineConfig.from_template("multimodal_clip")
   pipeline = ONNXPipeline("openai/clip-vit-base-patch16",config=config)
   pipeline.export2file("clip_16")
   ```

For an end to end example of using CLIP to generate multi-modal embeddings, see Generate Multi-modal Embeddings Using CLIP.

ONNX Pipeline Models: Text Classification

ONNX pipeline models provides text classification models that accepts text strings as input and produces the probablity of the input string belonging to a specific label. The pipeline models also provide the necessary pre-processing and post-processing.

In addition to text embedding models, Hugging Face repository also hosts transformer models that can be used for text classification. These models are typically fine-tuned embedding models on which a classification "head" was appended. The head changes the output of the model from a vector of embeddings to a list of labels and probabilities. For text based classification tasks such as sentiment analysis, you can generate a Text Classification Pipeline using OML4Py 2.1.

Text Classification Pipeline

1. **Input**: Input to the text classification pipeline is provided in the form of a batch, or array, of 1 ore more text strings. Each text string provided in the input will correspond to an output list containing the labels and probabilities.
2. **Pre-Processing**: Similar to text embedding pipeline, the text classification pipeline also configures a tokenizer for tolenizing the text inputs. The following tokenizer classes are supported in OML4Py 2.1:

Table 4-2 Tokenizer Classes available for Text Classifiaction Pipeline

Tokenizer Class	Tokenizer Type
transformers.models.bert.BertTokenizer	BERT
transformers.models.clip.CLIPTokenizer	CLIP
transformers.models.distilbert.DistilBertTokenizer	BERT
transformers.models.gpt2.GPT2Tokenizer	GPT2
transformers.models.mpnet.MPNetTokenizer	BERT
transformers.models.roberta.tokenization_roberta.RobertaTokenizer	ROBERTA
transformers.models.xlm_roberta.XLMRobertaTokenizer	SENTENCEPIECE

> **Note:**
>
> A tokenizer is automatically configured based on the tokenizer class configured for the model on Hugging Face. Tokenizer classes are provided by the transformer library. If the tokenizer class configured for the model in Hugging Face is not supported by OML4Py 2.1, an error will be raised.

3. **Original Model**: The original model must be a pre-trained PyTorch model in Hugging Face repository or from the local file system. Models on the local file system must match the Hugging Face format.
4. **Post-Processing**: The text classification pipeline provides a softmax function for post-processing by default. The softmax function will normalize the set of scores (logits) produced by the model into a probability distribution where each value is normalized to a range of [0,1]and all values sum to 1. You can choose to not include the softmax post-processing by providing an empty list of post-processors when generating the pipeline in OML4Py 2.1.
5. **Output**: The output of the classification pipeline is a list of probabilities for each input string. The length of the list is equal to the total number of classification targets or labels.

The pipeline generator will attempt to use label data provided by the model's config in Hugging Face. This metadata is typically found in the config.json file in the label2id property. If this property is not found in a model's config.json then the pipeline generator will use the metadata that you provide or default to using the output index as the label. If you provide label metadata, it takes priority over the metadata provided in the config.json on the Hugging Face repositpry.

> **Note:**
>
> Label metadata does not become part of the ONNX model. It is only applied when exporting the model to the db with export2db.

Text Classification Pipeline Examples

In order to generate ONNX pipeline models for image classification, `MiningFunction` class is used from the python utilities. The `MiningFunction` class can take one of the three values:

EMBEDDING, CLASSIFICATION, and REGRESSION. You can choose one depending on the task which you would like to perform. The MiningFunction enum is defined as follows :

```
class MiningFunction(Enum) :
    EMBEDDING = 1
    CLASSIFCATION = 2
    REGRESSION = 3
```

Since you are working on text classification pipeline, you would choose CLASSIFICATION or 2 (Enum) for the class MiningFunction

> **WARNING:**
>
> EmbeddingModel and EmbeddingModelConfig are deprecated. Instead, please use ONNXPipeline and ONNXPipelineConfig respectively. The details of the deprecated classes can be found in Python Classes to Convert Pretrained Models to ONNX Models (Deprecated). If a you choose to use a deprecated class, a warning message will be shown indicating that the classes will be removed in the future and advising the user to switch to the new class.

1. Example for generating a text pipeline with a template:

```
from oml.utils import ONNXPipeline,ONNXPipelineConfig,MiningFunction
config = ONNXPipelineConfig.from_template("text",max_seq_length=512)
pipeline = ONNXPipeline("SamLowe/roberta-base-
go_emotions",config=config,function=MiningFunction.CLASSIFICATION)
pipeline.export2file("emotions","testouput")
```

2. Importing the text classification pipeline generated in the above example in the DB (SQL)

> **Note:**
>
> The following code assumes that you have created a directory in PL/SQL named 'ONNX_IMPORT'

```
BEGIN
    DBMS_VECTOR.LOAD_ONNX_MODEL(
    'ONNX_IMPORT',
    'emotions.onnx',
    'emotions',

JSON('{"function":"classification","classificationProbOutput":"logits","inp
ut":{"input":["DATA"]},
        "labels":
["admiration","amusement","anger","annoyance","approval","caring","confusio
n","curiosity",

"desire","disappointment","disapproval","disgust","embarrassment","exciteme
nt","fear","gratitude",

"grief","joy","love","nervousness","optimism","pride","realization","relief
```

```
","remorse","sadness","surprise","neut"]}')
    );
END;
```

In this example the label metadata is being applied when invoking the `DBMS_VECTOR.LOAD_ONNX_MODEL` function to import the ONNX pipeline into the database. Please recall that the "labels" property is part of the JSON argument to the function and not a part of the ONNX file. The labels must be provided separately (if not provided only the numerical indices of the output will be used). When exporting directly to the database from OML4Py using the `export2db` function, you can either rely on the default label metadata provided in the model's configuration on Hugging Face, or provide your own label metadata via a template argument.

3. Example for exporting a text classification pipeline with default label metadata:

```
from oml.utils importONNXPipeline,ONNXPipelineConfig,MiningFunction
import oml
config =
ONNXPipelineConfig.from_template("text",max_seq_length=512,labels=["admirat
ion","amusement","anger","annoyance","approval","caring",

"confusion","curiosity","desire","disappointment","disapproval","disgust","
embarrassment","excitement",

"fear","gratitude","grief","joy","love","nervousness","optimism","pride","r
ealization","relief",

"remorse","sadness","surprise","neut"])
pipeline = ONNXPipeline("SamLowe/roberta-base-
go_emotions",config=config,function=MiningFunction.CLASSIFICATION)
oml.connect("pyquser","pyquser",dsn="pydsn")
pipeline.export2db("emotions")
```

In this example the pipeline is exported with label metadata that you provided in the template. This example is a combination of the examples provided in the previous two steps.

4. Example for scoring a text classification pipeline:

```
select prediction(emotions using 'Today is a good day' as data),
         prediction_probability(emotions using 'Today is a good day'as
data) from dual;
```

ONNX Pipeline Models: Reranking Pipeline

ONNX pipeline models provide a reranking pipeline that calculates similarity score for a given pair of texts.

Reranking Pipeline

Reranking models (also known as cross-encoders or rerankers) calculate a similarity score for given pairs of texts. Instead of encoding the input text into a fixed-length vector as text embedding models do, rerankers encode two input texts simultaneously and produces a similarity score for the pair. By default it outputs a logit which can be converted to a number between 0 and 1 by applying a sigmoid activation function on it. Although you can compute

similarity score between two texts through embedding models by first computing the embeddings of the texts and then applying cosine similarity, the re-ranker models usually provide superior performance and they can be used to re-rank the top-k results from an embedding model.

1. **Input**: There are two inputs to the reranking pipeline named `first_input` and `second_input`. Each contains an array of one or more text strings. The two inputs should have equal number of text strings. An array will be produced for each pair of texts from `first_input` and`second_input`. For example, for the following input pairs two arrays are produced, one for the pair`'hi'` and `'hi'` and the other for the pair `'halloween'` and `'hello'`:
   `{'first_input':['hi','halloween'],'second_input':['hi','hello']}`
2. **Pre-Processing**:
   Pre-processing for reranking pipeline includes tokenization. The tokenizer class: `transformers.models.xlm_roberta.tokenization_xlm_roberta.XLMRobertaTokenizer` is supported in OML4Py 2.1 for reranking models.
3. **Output**: The output of a reranking pipeline is an array of similarity scores, one for each input text pair. For the above example in the Input section, the output similarity score is 9.290878 for the pair `'hi'` and `'hi'` and 4.3913193 for the pair `'halloween'` and `'hello'`.

Reranking Pipeline Examples

1. Generating a Reranking Pipeline:

```
mn = 'BAAI/bge-reranker-base'
em = ONNXPipeline(mn, function=MiningFunction.REGRESSION)
mf = 'bge-reranker-base'
em.export2file(mf)
```

2. Importing the reranker model to the database:

```
BEGIN
DBMS_VECTOR.LOAD_ONNX_MODEL('DM_DUMP','bge-reranker-
base.onnx','doc_model', JSON('{"function" : "regression",
                              "regressionOutput" : "output", "input":
{"first_input": ["DATA1"],"second_input": ["DATA2"]}}'));
END;
```

3. Obtain the similarity score in the database:

```
SELECT prediction(doc_model USING 'what is panda?' as DATA1, 'hi' as
DATA2) from dual;
```

   The above example produces a similarity score between two text strings using the re-ranker ONNX pipeline imported in step 2. The score range from -infinity to +infinity. Score close to +infinity indicates a high similarity between the two strings, and score closer to -infinity indicate low similarity between the strings. A sigmoid function can map the similarity score to a float value in the range [0,1]. A sample output score corresponding to the above example is shown here:

```
PREDICTION(DOC_MODELUSING'WHATISPANDA?'ASDATA1,'HI'ASDATA2)
-----------------------------------------------------------
                    -7.73E+000
```

Convert Pretrained Models to ONNX Model: End-to-End Instructions for Text Embedding

This section provides end-to-end instructions from installing the OML4Py client to downloading a pretrained embedding model in ONNX-format using the Python utility package offered by Oracle.

> **Note:**
>
> This example provides end to end instructions for converting pretrained text models to ONNX models. Steps 1 - 9 are identical for image models and multi-modals. You can use appropriate code/syntax mentioned in the corresponding topics to convert image models and multi models to ONNX pipeline models.

These instructions assume you have configured your Oracle Linux 8 repo in `/etc/yum.repos.d`, configured a Wallet if using an Autonomous Database, and set up a proxy if needed.

1. Install Python:

```
sudo yum install libffi-devel openssl openssl-devel tk-devel xz-devel zlib-
devel bzip2-devel readline-devel libuuid-devel ncurses-devel libaio
mkdir -p $HOME/python
wget https://www.python.org/ftp/python/3.12.6/Python-3.12.6.tgz
tar -xvzf Python-3.12.6.tgz --strip-components=1 -C $HOME/python
cd $HOME/python
./configure --prefix=$HOME/python
make clean; make
make altinstall
```

2. Set variables `PYTHONHOME`, `PATH`, and `LD_LIBRARY_PATH`:

```
export PYTHONHOME=$HOME/python
export PATH=$PYTHONHOME/bin:$PATH
export LD_LIBRARY_PATH=$PYTHONHOME/lib:$LD_LIBRARY_PATH
```

3. Create symlink for python3 and pip3:

```
cd $HOME/python/bin
ln -s python3.12 python3
ln -s pip3.12 pip3
```

4. Install Oracle Instant client if you will be exporting embedded models to the database from Python. If you will be exporting to a file, skip steps 4 and 5 and see the note under environment variables in step 6:

```
cd $HOME
wget https://download.oracle.com/otn_software/linux/instantclient/2340000/
instantclient-basic-linux.x64-23.4.0.24.05.zip
unzip instantclient-basic-linux.x64-23.4.0.24.05.zip
```

5. Set variable `LD_LIBRARY_PATH`:

```
export LD_LIBRARY_PATH=$HOME/instantclient_23_4:$LD_LIBRARY_PATH
```

6. Create an environment file, for example, `env.sh`, that defines the Python and Oracle Instant client environment variables and source these environment variables before each OML4Py client session. Alternatively, add the environment variable definitions to `.bashrc` so they are defined when the user logs into their Linux machine.

```
# Environment variables for Python
export PYTHONHOME=$HOME/python
export PATH=$PYTHONHOME/bin:$PATH
export LD_LIBRARY_PATH=$PYTHONHOME/lib:$LD_LIBRARY_PATH
```

> **Note:**
>
> Environment variable for Oracle Instant Client - only if the Oracle Instant Client is installed for exporting models to the database.
> `export LD_LIBRARY_PATH=$HOME/instantclient_23_4:$LD_LIBRARY_PATH`
>
> .

7. Create a file named requirements.txt that contains the required third-party packages listed below.

```
--extra-index-url https://download.pytorch.org/whl/cpu
pandas==2.2.2
setuptools==70.0.0
scipy==1.14.0
matplotlib==3.8.4
oracledb==2.4.1
scikit-learn==1.5.1
numpy==2.0.1
onnxruntime==1.20.0
onnxruntime-extensions==0.12.0
onnx==1.17.0
torch==2.6.0
transformers==4.49.0
sentencepiece==0.2.0
```

8. Upgrade pip3 and install the packages listed in requirements.txt.

```
pip3 install --upgrade pip
pip3 install -r requirements.txt
```

9. Install OML4Py client. Download OML4Py 2.1 client from OML4Py download page and upload it to the Linux machine.

```
unzip oml4py-client-linux-x86_64-2.1.zip
pip3 install client/oml-2.1-cp312-cp312-linux_x86_64.whl
```

10. Get a list of all preconfigured models. Start Python and import ONNXPipelineConfig from `oml.utils`.

```
python3

from oml.utils import ONNXPipelineConfig

ONNXPipelineConfig.show_preconfigured()


['sentence-transformers/all-mpnet-base-v2',
'sentence-transformers/all-MiniLM-L6-v2',
'sentence-transformers/multi-qa-MiniLM-L6-cos-v1',
'sentence-transformers/distiluse-base-multilingual-cased-v2',
'sentence-transformers/all-MiniLM-L12-v2',
'BAAI/bge-small-en-v1.5',
'BAAI/bge-base-en-v1.5',
'taylorAI/bge-micro-v2',
'intfloat/e5-small-v2',
'intfloat/e5-base-v2',
'thenlper/gte-base',
'thenlper/gte-small',
'TaylorAI/gte-tiny',
'sentence-transformers/paraphrase-multilingual-mpnet-base-v2',
'intfloat/multilingual-e5-base',
'intfloat/multilingual-e5-small',
'sentence-transformers/stsb-xlm-r-multilingual',
'Snowflake/snowflake-arctic-embed-xs',
'Snowflake/snowflake-arctic-embed-s',
'Snowflake/snowflake-arctic-embed-m',
'mixedbread-ai/mxbai-embed-large-v1',
'openai/clip-vit-large-patch14',
'google/vit-base-patch16-224',
'microsoft/resnet-18',
'microsoft/resnet-50',
'WinKawaks/vit-tiny-patch16-224',
'Falconsai/nsfw_image_detection',
'WinKawaks/vit-small-patch16-224',
'nateraw/vit-age-classifier',
'rizvandwiki/gender-classification',
'AdamCodd/vit-base-nsfw-detector',
'trpakov/vit-face-expression',
'BAAI/bge-reranker-base']
```

11. Choose from:

    - To generate an ONNX file that you can manually upload to the database using the `DBMS_VECTOR.LOAD_ONNX_MODEL`, refer to step 3 of SQL Quick Start and skip steps 12 and 13.
    - To upload the model directly into the database, skip this step and proceed to step 12.

Export a preconfigured embedding model to a local file. Import ONNXPipeline and ONNXPipelineConfig from `oml.utils`. This exports the ONNX-format model to your local file system.

```
from oml.utils import ONNXPipeline, ONNXPipelineConfig

# Export to file
pipeline = ONNXPipeline(model_name="sentence-transformers/all-MiniLM-L6-
v2")
pipeline.export2file("your_preconfig_file_name",output_dir=".")
```

Move the ONNX file to a directory on the database server, and create a directory on the file system and in the database for the import.

```
mkdir -p /tmp/models
sqlplus / as sysdba
alter session set container=<name of pluggable database>;
```

Apply the necessary permissions and grants.

```
-- directory to store ONNX files for import
CREATE DIRECTORY ONNX_IMPORT AS '/tmp/models';
-- grant your OML user read and write permissions on the directory
GRANT READ, WRITE ON DIRECTORY ONNX_IMPORT to OMLUSER;
-- grant to allow user to import the model
GRANT CREATE MINING MODEL TO OMLUSER;
```

Use the `DBMS_VECTOR.LOAD_ONNX_MODEL` procedure to load the model in your OML user schema. In this example, the procedure loads the ONNX model file named `all-MiniLM-L6.onnx` from the ONNX_IMPORT directory into the database as a model named ALL_MINILM_L6.

```
BEGIN
   DBMS_VECTOR.LOAD_ONNX_MODEL(
     directory => 'ONNX_IMPORT',
     file_name => 'all-MiniLM-L6-v2.onnx',
     model_name => 'ALL_MINILM_L6',
     metadata => JSON('{"function" : "embedding", "embeddingOutput" :
"embedding", "input": {"input": ["DATA"]}}'));
END;
```

12. Export a preconfigured embedding model to the database. If using a database connection to update to match your credentials and database environment.

> **Note:**
>
> To ensure step 12 works properly, complete steps 4 and 5 first.

```
# Import oml library and EmbeddingModel from oml.utils
import oml
from oml.utils import ONNXPipeline, ONNXPipelineConfig
```

```
# Set embedded mode to false for Oracle Database on premises. This is not
supported or required for Oracle Autonomous Database.
oml.core.methods.__embed__ = False

# Create a database connection.

# Oracle Database on-premises
oml.connect("<user>", "<password>", port=<port number> host="<hostname>",
service_name="<service name>")

# Oracle Autonomous Database
oml.connect(user="<user>", password="<password>", dsn="myadb_low")
pipeline = ONNXPipeline(model_name="sentence-transformers/all-MiniLM-L6-
v2")
em.export2db("ALL_MINILM_L6")
```

Query the model and its views, and you can generate embeddings from Python or SQL.

```
import oracledb
cr = oml.cursor()
data = cr.execute("select vector_embedding(ALL_MINILM_L6 using 'RES' as
DATA)AS embedding from dual")
data.fetchall()


SELECT VECTOR_EMBEDDING(ALL_MINILM_L6 USING 'RES' as DATA) AS embedding;
```

13. Verify the model exists using SQL:

```
sqlplus $USER/pass@PDBNAME;


select model_name, algorithm, mining_function from user_mining_models
where  model_name='ALL_MINILM_L6';


------------------------------------------------------------------------------
MODEL_NAME                ALGORITHM                      MINING_FUNCTION
------------------------------ ----------------------------------------
ALL_MINILM_L6             ONNX                           EMBEDDING
```

Import ONNX Models into Oracle Database End-to-End Example

Learn to import a pretrained embedding model that is in ONNX format and generate vector embeddings.

Follow the steps below to import a pretrained ONNX formatted embedding model into the Oracle Database.

Prepare Your Data Dump Directory

Prepare your data dump directory and provide the necessary access and privileges to `dmuser`.

1. Choose from:

a. If you already have a pretrained ONNX embedding model, store it in your working folder.

b. If you do not have pretrained embedding model in ONNX format, perform the steps listed in Convert Pretrained Models to ONNX Format.

2. Login to SQL*Plus as `SYSDBA` in your PDB.

```
CONN sys/<password>@pdb as sysdba;
```

3. Grant the `DB_DEVELOPER_ROLE` to `dmuser`.

```
GRANT DB_DEVELOPER_ROLE TO dmuser identified by <password>;
```

4. Grant `CREATE MINING MODEL` privilege to `dmuser`.

```
GRANT create mining model TO dmuser;
```

5. Set your working folder as the data dump directory (`DM_DUMP`) to load the ONNX embedding model.

```
CREATE OR REPLACE DIRECTORY DM_DUMP as '<work directory path>';
```

6. Grant `READ` permissions on the `DM_DUMP` directory to `dmuser`.

```
GRANT READ ON DIRECTORY dm_dump TO dmuser;
```

7. Grant `WRITE` permissions on the `DM_DUMP` directory to `dmuser`.

```
GRANT WRITE ON DIRECTORY dm_dump TO dmuser;
```

8. Drop the model if it already exits.

```
exec DBMS_VECTOR.DROP_ONNX_MODEL(model_name => 'doc_model', force => true);
```

Import ONNX Model Into the Database

You created a data dump directory and now you load the ONNX model into the Database. Use the `DBMS_VECTOR.LOAD_ONNX_MODEL` procedure to load the model. The `DBMS_VECTOR.LOAD_ONNX_MODEL` procedure facilitates the process of importing ONNX format model into the Oracle Database. In this example, the procedure loads an ONNX model file, named `my_embedding_model.onnx` from the `DM_DUMP` directory, into the Database as `doc_model`, specifying its use for embedding tasks.

1. Connect as `dmuser`.

```
CONN dmuser/<password>@<pdbname>;
```

2. Load the ONNX model into the Database.

   If the ONNX model to be imported already includes an output tensor named `embeddingOutput` and an input string tensor named `data`, JSON metadata is unnecessary.

Embedding models converted from OML4Py follow this convention and can be imported without the JSON metadata.

```
EXECUTE DBMS_VECTOR.LOAD_ONNX_MODEL(
  'DM_DUMP',
  'my_embedding_model.onnx',
  'doc_model');
```

Alternately, you can load the ONNX embedding model by specifying the JSON metadata.

```
EXECUTE DBMS_VECTOR.LOAD_ONNX_MODEL(
  'DM_DUMP',
  'my_embedding_model.onnx',
  'doc_model',
  JSON('{"function" : "embedding", "embeddingOutput" : "embedding", "input": {"input":
["DATA"]}}'));
```

The procedure `LOAD_ONNX_MODEL` declares these parameters:

- `DM_DUMP`: specifies the directory name of the data dump.

  > **Note:**
  >
  > Ensure that the `DM_DUMP` directory is defined.

- `my_embedding_model`: is a `VARCHAR2` type parameter that specifies the name of the ONNX model.
- `doc_model`: This parameter is a user-specified name under which the model is stored in the Oracle Database.
- The JSON metadata associated with the ONNX model is declared as:

  `"function" : "embedding"`: Indicates the function name for text embedding model.

  `"embeddingOutput" : "embedding"`: Specifies the output variable which contains the embedding results.

- `"input": {"input": ["DATA"]}`: Specifies a JSON object (`"input"`) that describes the input expected by the model. It specifies that there is an input named `"input"`, and its value should be an array with one element, `"DATA"`. This indicates that the model expects a single string input to generate embeddings.

For more information about the `LOAD_ONNX_MODEL` procedure, see *Oracle Database PL/SQL Packages and Types Reference*.

Alternatively, if your ONNX embedding model is loaded on cloud object storage, the `LOAD_ONNX_MODEL_CLOUD` procedure can be used. For more information, see *Oracle Database PL/SQL Packages and Types Reference*.

Query Model Statistics

You can view model attributes and learn about the model by querying machine learning dictionary views and model detail views.

> **Note:**
>
> *DOC_MODEL* is the user-specified name of the embedding text model.

1. Query `USER_MINING_MODEL_ATTRIBUTES` view.

   ```
   SELECT model_name, attribute_name, attribute_type, data_type, vector_info
   FROM user_mining_model_attributes
   WHERE model_name = 'DOC_MODEL'
   ORDER BY ATTRIBUTE_NAME;
   ```

   To learn about `USER_MINING_MODEL_ATTRIBUTES` view, see USER_MINING_MODEL_ATTRIBUTES.

2. Query `USER_MINING_MODELS` view.

   ```
   SELECT MODEL_NAME, MINING_FUNCTION, ALGORITHM,
   ALGORITHM_TYPE, MODEL_SIZE
   FROM user_mining_models
   WHERE model_name = 'DOC_MODEL'
   ORDER BY MODEL_NAME;
   ```

   To learn about `USER_MINING_MODELS` view, see USER_MINING_MODELS.

3. Check model statistics by viewing the model detail views. Query the `DM$VM`*`DOC_MODEL`* view.

   ```
   SELECT * FROM DM$VMDOC_MODEL ORDER BY NAME;
   ```

   To learn about model details views for ONNX embedding models, see Model Details Views for ONNX Models.

4. Query the `DM$VP`*`DOC_MODEL`* model detail view.

   ```
   SELECT * FROM DM$VPDOC_MODEL ORDER BY NAME;
   ```

5. Query the `DM$VJ`*`DOC_MODEL`* model detail view.

   ```
   SELECT * FROM DM$VJDOC_MODEL;
   ```

Generate Embeddings

Apply the model and generate vector embeddings for your input. Here, the input is *hello*.

Generate vector embeddings using the `VECTOR_EMBEDDING` function.

```
SELECT TO_VECTOR(VECTOR_EMBEDDING(doc_model USING 'hello' as data)) AS
embedding;
```

To learn about the `VECTOR_EMBEDDING` SQL function, see VECTOR_EMBEDDING. You can use the `UTL_TO_EMBEDDING` function in the `DBMS_VECTOR_CHAIN` PL/SQL package to generate vector embeddings generically through REST endpoints. To explore these functions, see the example Convert Text String to Embedding.

Example: Importing a Pretrained ONNX Model to Oracle Database

The following presents a comprehensive step-by-step example of importing ONNX embedding and generating vector embeddings.

```
conn sys/<password>@pdbname as sysdba
grant db_developer_role to dmuser identified by <password>;
grant create mining model to dmuser;

create or replace directory DM_DUMP as '<work directory path>';
grant read on directory dm_dump to dmuser;
grant write on directory dm_dump to dmuser;
>conn dmuser/<password>@<pdbname>;

-- Drop the model if it exits
exec DBMS_VECTOR.DROP_ONNX_MODEL(model_name => 'doc_model', force => true);

-- Load Model
EXECUTE DBMS_VECTOR.LOAD_ONNX_MODEL(
    'DM_DUMP',
    'my_embedding_model.onnx',
    'doc_model',
    JSON('{"function" : "embedding", "embeddingOutput" : "embedding"}'));
/

--check the attributes view
set linesize 120
col model_name format a20
col algorithm_name format a20
col algorithm format a20
col attribute_name format a20
col attribute_type format a20
col data_type format a20

SQL> SELECT model_name, attribute_name, attribute_type, data_type, vector_info
FROM user_mining_model_attributes
WHERE model_name = 'DOC_MODEL'
ORDER BY ATTRIBUTE_NAME;


OUTPUT:

MODEL_NAME           ATTRIBUTE_NAME       ATTRIBUTE_TYPE       DATA_TYPE
VECTOR_INFO
-------------------- -------------------- -------------------- ----------
---------------
DOC_MODEL                INPUT_STRING         TEXT                 VARCHAR2
DOC_MODEL                ORA$ONNXTARGET       VECTOR               VECTOR
VECTOR(128,FLOA
                                                                       T32)



SQL> SELECT MODEL_NAME, MINING_FUNCTION, ALGORITHM,
ALGORITHM_TYPE, MODEL_SIZE
```

```
FROM user_mining_models
WHERE model_name = 'DOC_MODEL'
ORDER BY MODEL_NAME;

OUTPUT:
MODEL_NAME           MINING_FUNCTION                ALGORITHM
ALGORITHM_ MODEL_SIZE
-------------------- ------------------------------ --------------------
---------- ----------
DOC_MODEL                EMBEDDING                      ONNX
NATIVE       17762137



SQL> select * from DM$VMDOC_MODEL ORDER BY NAME;

OUTPUT:
NAME                                   VALUE
---------------------------------------
---------------------------------------
Graph Description                      Graph combining g_8_torch_jit and
torch_
                                       jit
                                       g_8_torch_jit



                                       torch_jit


Graph Name                             g_8_torch_jit_torch_jit
Input[0]                               input:string[1]
Output[0]                              embedding:float32[?,128]
Producer Name                          onnx.compose.merge_models
Version                                1

6 rows selected.


SQL> select * from DM$VPDOC_MODEL ORDER BY NAME;

OUTPUT:
NAME                                   VALUE
---------------------------------------
---------------------------------------
batching                               False
embeddingOutput                        embedding


SQL> select * from DM$VJDOC_MODEL;

OUTPUT:
METADATA
--------------------------------------------------------------------------------
--
{"function":"embedding","embeddingOutput":"embedding","input":{"input":
```

```
["DATA"]}}



--apply the model
SQL> SELECT TO_VECTOR(VECTOR_EMBEDDING(doc_model USING 'hello' as data)) AS
embedding;

-------------------------------------------------------------------------------
--
[-9.76553112E-002,-9.89954844E-002,7.69771636E-003,-4.16760892E-003,-9.6930563
4E-002,
-3.01141385E-002,-2.63396613E-002,-2.98553891E-002,5.96499592E-002,4.13885899E
-002,
5.32859489E-002,6.57707453E-002,-1.47056757E-002,-4.18472625E-002,4.1588001E-0
02,
-2.86354572E-002,-7.56499246E-002,-4.16395674E-003,-1.52879998E-001,6.60010576
E-002,
-3.9013084E-002,3.15719917E-002,1.2428958E-002,-2.47651711E-002,-1.16851285E-0
01,
-7.82847106E-002,3.34323719E-002,8.03267583E-002,1.70483496E-002,-5.42407483E-
002,
6.54291287E-002,-4.81935125E-003,6.11041225E-002,6.64106477E-003,-5.47
```

Oracle AI Vector Search SQL Scenario

To learn how you can chunk *database-concepts23ai.pdf* and *oracle-ai-vector-search-users-guide.pdf*, generate vector embeddings, and perform similarity search using vector indexes, see Quick Start SQL.

- Alternate Method to Import ONNX Models
  Use the `DBMS_DATA_MINING.IMPORT_ONNX_MODEL` procedure to import the model and declare the input name. A PL/SQL helper block is used to facilitate the process of importing the ONNX format model into the Oracle Database in the included example.

Alternate Method to Import ONNX Models

Use the `DBMS_DATA_MINING.IMPORT_ONNX_MODEL` procedure to import the model and declare the input name. A PL/SQL helper block is used to facilitate the process of importing the ONNX format model into the Oracle Database in the included example.

Perform the following steps to import ONNX model into the Database using `DBMS_DATA_MINING` PL/SQL package.

- Connect as `dmuser`.

  ```
  CONN dmuser/<password>@<pdbname>;
  ```

- Run the following helper PL/SQL block:

  ```
  DECLARE
      m_blob BLOB default empty_blob();
      m_src_loc BFILE ;
      BEGIN
      DBMS_LOB.createtemporary (m_blob, FALSE);
      m_src_loc := BFILENAME('DM_DUMP', 'my_embedding_model.onnx');
  ```

```
    DBMS_LOB.fileopen (m_src_loc, DBMS_LOB.file_readonly);
    DBMS_LOB.loadfromfile (m_blob, m_src_loc, DBMS_LOB.getlength
(m_src_loc));
    DBMS_LOB.CLOSE(m_src_loc);
    DBMS_DATA_MINING.import_onnx_model ('doc_model', m_blob,
JSON('{"function" : "embedding", "embeddingOutput" : "embedding", "input":
{"input": ["DATA"]}}'));
    DBMS_LOB.freetemporary (m_blob);
    END;
    /
```

The code sets up a `BLOB` object and a `BFILE` locator, creates a temporary `BLOB` for storing the `my_embedding_model.onnx` file from the `DM_DUMP` directory, and reads its contents into the `BLOB`. It then closes the file and uses the content to import an ONNX model into the database with specified metadata, before releasing the temporary `BLOB` resources.

The schema of the `IMPORT_ONNX_MODEL` procedure is as follows: `DBMS_DATA_MINING.IMPORT_ONNX_MODEL(model_data, model_name, metadata)`. This procedure loads `IMPORT_ONNX_MODEL` from the `DBMS_DATA_MINING` package to import the ONNX model into the Database using the name provided in `model_name`, the BLOB content in `m_blob`, and the associated `metadata`.

- `doc_model`: This parameter is a user-specified name under which the imported model is stored in the Oracle Database.
- `m_blob`: This is a model data in `BLOB` that holds the ONNX representation of the model.
- `"function" : "embedding"`: Indicates the function name for text embedding model.
- `"embeddingOutput" : "embedding"`: Specifies the output variable which contains the embedding results.
- `"input": {"input": ["DATA"]}`: Specifies a JSON object (`"input"`) that describes the input expected by the model. It specifies that there is an input named `"input"`, and its value should be an array with one element, `"DATA"`. This indicates that the model expects a single string input to generate embeddings.

Alternately, the `DBMS_DATA_MINING.IMPORT_ONNX_MODEL` procedure can also accept a `BLOB` argument representing an ONNX file stored and loaded from OCI Object Storage. The following is an example to load an ONNX model stored in an OCI Object Storage.

```
DECLARE
  model_source BLOB := NULL;
BEGIN
  -- get BLOB holding onnx model
  model_source := DBMS_CLOUD.GET_OBJECT(
    credential_name => 'myCredential',
    object_uri => 'https://objectstorage.us-phoenix -1.oraclecloud.com/' ||
      'n/namespace -string/b/bucketname/o/myONNXmodel.onnx');

  DBMS_DATA_MINING.IMPORT_ONNX_MODEL(
    "myonnxmodel",
    model_source,
    JSON('{ function : "embedding" })
  );
END;
/
```

This PL/SQL block starts by initializing a `model_source` variable as a `BLOB` type, initially set to NULL. It then retrieves an ONNX model from Oracle Cloud Object Storage using the `DBMS_CLOUD.GET_OBJECT` procedure, specifying the credentials `(OBJ_STORE_CRED)` and the URI of the model. The ONNX model resides in a specific bucket named `bucketname` in this case, and is accessible through the provided URL. Then, the script loads the ONNX model into the `model_source` BLOB. The `DBMS_DATA_MINING.IMPORT_ONNX_MODEL` procedure then imports this model into the Oracle Database as `myonnxmodel`. During the import, a JSON metadata specifies the model's function as `embedding`, for embedding operations.

See IMPORT_ONNX_MODEL Procedure and GET_OBJECT Procedure and Function to learn about the PL/SQL procedure.

Example: Importing a Pretrained ONNX Model to Oracle Database

The following presents a comprehensive step-by-step example of importing ONNX embedding and generating vector embeddings.

```
conn sys/<password>@pdb as sysdba
grant db_developer_role to dmuser identified by dmuser;
grant create mining model to dmuser;

create or replace directory DM_DUMP as '<work directory path>';
grant read on directory dm_dump to dmuser;
grant write on directory dm_dump to dmuser;
>conn dmuser/<password>@<pdbname>;

-- Drop the model if it exits
exec DBMS_VECTOR.DROP_ONNX_MODEL(model_name => 'doc_model', force => true);

-- Load Model
EXECUTE DBMS_VECTOR.LOAD_ONNX_MODEL(
    'DM_DUMP',
    'my_embedding_model.onnx',
    'doc_model',
    JSON('{"function" : "embedding", "embeddingOutput" : "embedding"}'));
/
--Alternately, load the model
EXECUTE DBMS_DATA_MINING.IMPORT_ONNX_MODEL(
       'my_embedding_model.onnx',
    'doc_model',
    JSON('{"function" : "embedding",
    "embeddingOutput" : "embedding",
    "input": {"input": ["DATA"]}}')
    );

--check the attributes view
set linesize 120
col model_name format a20
col algorithm_name format a20
col algorithm format a20
col attribute_name format a20
col attribute_type format a20
col data_type format a20

SQL> SELECT model_name, attribute_name, attribute_type, data_type, vector_info
FROM user_mining_model_attributes
```

```
WHERE model_name = 'DOC_MODEL'
ORDER BY ATTRIBUTE_NAME;


OUTPUT:

MODEL_NAME           ATTRIBUTE_NAME       ATTRIBUTE_TYPE       DATA_TYPE
VECTOR_INFO
-------------------- -------------------- -------------------- ----------
---------------
DOC_MODEL                INPUT_STRING         TEXT                 VARCHAR2
DOC_MODEL                ORA$ONNXTARGET       VECTOR               VECTOR
VECTOR(128,FLOA
                                                                          T32)



SQL> SELECT MODEL_NAME, MINING_FUNCTION, ALGORITHM,
ALGORITHM_TYPE, MODEL_SIZE
FROM user_mining_models
WHERE model_name = 'DOC_MODEL'
ORDER BY MODEL_NAME;

OUTPUT:
MODEL_NAME           MINING_FUNCTION                ALGORITHM
ALGORITHM_ MODEL_SIZE
-------------------- ------------------------------ --------------------
---------- ----------
DOC_MODEL                EMBEDDING                      ONNX
NATIVE       17762137



SQL> select * from DM$VMDOC_MODEL ORDER BY NAME;

OUTPUT:
NAME                                   VALUE
---------------------------------------
---------------------------------------
Graph Description                      Graph combining g_8_torch_jit and
torch_
                                       jit
                                       g_8_torch_jit


                                       torch_jit


Graph Name                             g_8_torch_jit_torch_jit
Input[0]                               input:string[1]
Output[0]                              embedding:float32[?,128]
Producer Name                          onnx.compose.merge_models
Version                                1

6 rows selected.
```

```
SQL> select * from DM$VPDOC_MODEL ORDER BY NAME;

OUTPUT:
NAME                                    VALUE
----------------------------------------
----------------------------------------
batching                                False
embeddingOutput                         embedding


SQL> select * from DM$VJDOC_MODEL;

OUTPUT:
METADATA
--------------------------------------------------------------------------------
--
{"function":"embedding","embeddingOutput":"embedding","input":{"input":
["DATA"]}}



--apply the model
SQL> SELECT TO_VECTOR(VECTOR_EMBEDDING(doc_model USING 'hello' as data)) AS
embedding;

--------------------------------------------------------------------------------
--
[-9.76553112E-002,-9.89954844E-002,7.69771636E-003,-4.16760892E-003,-9.6930563
4E-002,
-3.01141385E-002,-2.63396613E-002,-2.98553891E-002,5.96499592E-002,4.13885899E
-002,
5.32859489E-002,6.57707453E-002,-1.47056757E-002,-4.18472625E-002,4.1588001E-0
02,
-2.86354572E-002,-7.56499246E-002,-4.16395674E-003,-1.52879998E-001,6.60010576
E-002,
-3.9013084E-002,3.15719917E-002,1.2428958E-002,-2.47651711E-002,-1.16851285E-0
01,
-7.82847106E-002,3.34323719E-002,8.03267583E-002,1.70483496E-002,-5.42407483E-
002,
6.54291287E-002,-4.81935125E-003,6.11041225E-002,6.64106477E-003,-5.47
```

Access Third-Party Models for Vector Generation Leveraging Third-Party REST APIs

You can access third-party vector embedding models to generate vector embeddings from your data, outside the database by calling third-party REST APIs.

The Vector Utility PL/SQL packages `DBMS_VECTOR` and `DBMS_VECTOR_CHAIN` (outlined in Supplied Vector Utility PL/SQL Packages) provide the third-party REST APIs that let you interact with external embedding models.

You can remotely access third-party service providers, such as Cohere, Google AI, Hugging Face, Generative AI, OpenAI, and Vertex AI. Alternatively, you can install Ollama on your local host to access open LLMs, such as Llama 3, Phi 3, Mistral, and Gemma 2. All the supported providers and corresponding REST operations allowed for each provider are listed in Supported Third-Party Provider Operations and Endpoints.

Review these high-level steps involved in generating embeddings by calling third-party REST APIs:

1. **Understand the terms of using third-party embedding models**.

   > **WARNING:**
   >
   > Certain features of the database may allow you to access services offered separately by third-parties, for example, through the use of JSON specifications that facilitate your access to REST APIs.
   >
   > Your use of these features is solely at your own risk, and you are solely responsible for complying with any terms and conditions related to use of any such third-party services. Notwithstanding any other terms and conditions related to the third-party services, your use of such database features constitutes your acceptance of that risk and express exclusion of Oracle's responsibility or liability for any damages resulting from such access.

2. **Configure your REST API connection**.

   By configuring your REST API connection, you enable Oracle Database to communicate with the REST endpoint URL where you want to send requests to access data from the third-party embedding service. The requested embedding service then processes the data and returns a vector representation.

   This involves the following tasks:

   a. **Create a user, set up storage, and grant necessary privileges**.

      Create a tablespace and a user, and then grant the `DB_DEVELOPER_ROLE` to that user. This role assigns all basic roles and privileges that are necessary for a database developer.

      You can pass the input directly as a text string, or prepare a data dump directory to upload your data for vector generation. Ensure to grant the user access to read and write your data dump directory.

   b. **Grant the connect privilege to allow connection to the third-party host**.

      Grant the `CONNECT` privilege to the user for connecting to your third-party host. You use the `DBMS_NETWORK_ACL_ADMIN` PL/SQL procedure (as described in DBMS_NETWORK_ACL_ADMIN) to specify the users and their privilege assignments that can access external network services from within the database.

      This procedure appends an access control entry (ACE) to the network access control list (ACL) for the specified host. The ACE grants a privilege to a principal (user name), which enables the user to connect to the external host that has been authorized in the database's networking ACL. This increases security by controlling the users that can connect to the specified hosts, and thus prevents unauthorized connections.

   c. **Set the proxy server, if configured**.

If you have configured a proxy server on your network, then you must specify the proxy server's host name and port number. This helps you access external web resources or embedding services by routing requests through your proxy server.

You use the `UTL_HTTP.SET_PROXY` PL/SQL procedure (as described in UTL_HTTP.SET_PROXY) to set the proxy server.

d. **Set up credentials to enable access to the REST provider**.

You require authentication credentials to enable access to your chosen third-party service provider. A credential name holds authentication parameters, such as user name, password, access token, private key, or fingerprint.

You use the credential helper procedures `CREATE_CREDENTIAL` and `DROP_CREDENTIAL` to securely manage credentials in the database. These procedures are available with both the `DBMS_VECTOR` and `DBMS_VECTOR_CHAIN` PL/SQL packages.

3. **Generate vector embeddings**.

You use chainable utility functions `UTL_TO_EMBEDDING` and `UTL_TO_EMBEDDINGS` to convert input data to one or more vector embeddings. These APIs are available with both the `DBMS_VECTOR` and the `DBMS_VECTOR_CHAIN` PL/SQL packages.

Determine which API to use:

- `UTL_TO_EMBEDDING` converts plain text (`CLOB`) or image (`BLOB`) to a single embedding (`VECTOR`).
- `UTL_TO_EMBEDDINGS` converts an array of chunks (`VECTOR_ARRAY_T`) to an array of embeddings (`VECTOR_ARRAY_T`).

For example, a typical `DBMS_VECTOR.UTL_TO_EMBEDDING` call specifies the REST provider, credential name, REST endpoint URL for an embedding service, and embedding model name parameters in a JSON object, as follows:

```
var params clob;
exec :params := '
{
 "provider": "cohere",
  "credential_name": "COHERE_CRED",
  "url": "https://api.cohere.example.com/embed",
  "model": "embed-model"
}';

select DBMS_VECTOR.UTL_TO_EMBEDDING('hello', json(:params)) from dual;
```

In this example, the input for generating embedding is `hello`.

Run end-to-end embedding examples:

To see how to apply these steps for performing various embedding use cases, you can run some end-to-end example scenarios listed in Vector Generation Examples.

Related Topics

- DBMS_VECTOR
  The `DBMS_VECTOR` package simplifies common operations with Oracle AI Vector Search, such as extracting chunks or embeddings from user data, generating text for a given prompt or an image, creating a vector index, or reporting on index accuracy.

- DBMS_VECTOR_CHAIN
  The `DBMS_VECTOR_CHAIN` package enables advanced operations with Oracle AI Vector Search, such as chunking and embedding data along with text generation and summarization capabilities. It is more suitable for text processing with similarity search and hybrid search, using functionality that can be pipelined together for an end-to-end search.

Vector Generation Examples

Run these end-to-end examples to see how you can generate vector embeddings, both within and outside the database.

- Generate Embeddings
  In these examples, you can see how to use the `VECTOR_EMBEDDING` SQL function or the `UTL_TO_EMBEDDING` PL/SQL function to generate a vector embedding from input text strings and images.
- Perform Chunking With Embedding
  In these examples, you can see how to explore the `VECTOR_CHUNKS` SQL function along with chainable utility PL/SQL functions to split large textual extracts and documents into chunks and then represent each chunk as a vector embedding.
- Configure Chunking Parameters
  Oracle AI Vector Search provides many parameters for chunking text data, such as `SPLIT [BY]`, `OVERLAP`, or `NORMALIZE`. In these examples, you can see how to configure these parameters to define your own chunking specifications and strategies, so that you can create meaningful chunks.

Generate Embeddings

In these examples, you can see how to use the `VECTOR_EMBEDDING` SQL function or the `UTL_TO_EMBEDDING` PL/SQL function to generate a vector embedding from input text strings and images.

- Convert Text String to Embedding Within Oracle Database
  Perform a text-to-embedding transformation by accessing a vector embedding model stored in the database.
- Convert Text String to BINARY Embedding Outside Oracle Database
  Perform a text-to-`BINARY`-embedding transformation by accessing a third-party `BINARY` vector embedding model.
- Convert Text String to Embedding Using Public REST Providers
  Perform a text-to-embedding transformation, using publicly hosted third-party embedding models by Cohere, Generative AI, Google AI, Hugging Face, OpenAI, or Vertex AI.
- Convert Text String to Embedding Using the Local REST Provider Ollama
  Perform a text-to-embedding transformation by accessing open embedding models, using the local host REST endpoint provider Ollama.
- Convert Image to Embedding Using Public REST Providers
  Perform an image-to-embedding transformation by making a REST call to the third-party service provider, Vertex AI. In this example, you can see how to vectorize both image and text inputs using a multimodal embedding model and then query a vector space containing vectors from both content types.
- Generate Multi-modal Embeddings Using CLIP
  This section provides end-to-end instructions from installing the OML4Py client to generating multi-modal embeddings using CLIP.

- Vectorize Relational Tables Using OML Feature Extraction Algorithms

Convert Text String to Embedding Within Oracle Database

Perform a text-to-embedding transformation by accessing a vector embedding model stored in the database.

You can download an embedding machine learning model, convert it into ONNX format (if not already in ONNX format), and load the model into Oracle Database. You can then access that model to vectorize your data that is used to populate a vector index. Note that you must use the same embedding model on both the data to be indexed and the user's input query. In this example, you can see how to vectorize a user's input query on the fly.

Here, you can call either the `VECTOR_EMBEDDING` SQL function or the `UTL_TO_EMBEDDING` PL/SQL function (note the singular "embedding"). Both `VECTOR_EMBEDDING` and `UTL_TO_EMBEDDING` directly return a `VECTOR` type (not an array).

To convert a user's input text string "`hello`" to a vector embedding, using an embedding model in ONNX format:

1. Connect to Oracle Database as a local user.
   a. Log in to SQL*Plus as the `SYS` user, connecting as `SYSDBA`:

```
conn sys/password as sysdba


CREATE TABLESPACE tbs1
DATAFILE 'tbs5.dbf' SIZE 20G AUTOEXTEND ON
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;


SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 10000
SET LONG 10000
```

   b. Create a local user (`docuser`) and grant necessary privileges:

```
DROP USER docuser cascade;


CREATE USER docuser identified by docuser DEFAULT TABLESPACE tbs1 quota
unlimited on tbs1;


GRANT DB_DEVELOPER_ROLE, create credential to docuser;
```

   c. Connect as the local user (`docuser`):

```
CONN docuser/password
```

2. Call either `VECTOR_EMBEDDING` or `UTL_TO_EMBEDDING`.

   a. Load your ONNX format embedding model into Oracle Database.

      For detailed information on how to perform this step, see Import ONNX Models into Oracle Database End-to-End Example.

   b. Call `VECTOR_EMBEDDING` or `UTL_TO_EMBEDDING`. You can use `UTL_TO_EMBEDDING` either from the `DBMS_VECTOR` or the `DBMS_VECTOR_CHAIN` package, depending on your use case.

      - `VECTOR_EMBEDDING`:

        ```
        SELECT TO_VECTOR(VECTOR_EMBEDDING(doc_model USING 'hello' as data))
        AS embedding;
        ```

      - `DBMS_VECTOR.UTL_TO_EMBEDDING`:

        ```
        var params clob;
        exec :params := '{"provider":"database", "model":"doc_model"}';

        select dbms_vector.utl_to_embedding('hello', json(:params)) from
        dual;
        ```

      Here, `doc_model` specifies the name under which your embedding model is stored in the database.

   The generated embedding appears as follows:

   ```
   EMBEDDING
   ----------------------------------------------------------------------------
   ------------------------------------------------------------
   [8.78423732E-003,-4.29633334E-002,-5.93001908E-003,-4.65480909E-002,2.14333
   013E-002,6.53376281E-002,-5.93746938E-002,2.10403297E-002,
   4.38376889E-002,5.22960871E-002,1.25104953E-002,6.49512559E-002,-9.26998071
   E-003,-6.97442219E-002,-3.02916039E-002,-4.74979728E-003,
   -1.08755399E-002,-4.63751052E-003,3.62781435E-002,-9.35919806E-002,-1.13934
   642E-002,-5.74270077E-002,-1.36667723E-002,2.42995787E-002,
   -6.96804151E-002,4.93822657E-002,1.01460628E-002,-1.56464987E-002,-2.394105
   68E-002,-4.27529104E-002,-5.65665103E-002,-1.74160264E-002,
   5.05326502E-002,4.31500375E-002,-2.6994409E-002,-1.72731467E-002,9.30535868
   E-002,6.85951149E-004,5.61876409E-003,-9.0233935E-003,
   -2.55788807E-002,-2.04174276E-002,3.74175981E-002,-1.67872179E-002,1.074793
   04E-001,-6.64602639E-003,-7.65537247E-002,-9.71965566E-002,
   -3.99636962E-002,-2.57076006E-002,-5.62455431E-002,-1.3583754E-001,3.459460
   29E-002,1.85191762E-002,3.01524661E-002,-2.62163244E-002,
   -4.05582506E-003,1.72979087E-002,-3.66434865E-002,-1.72491539E-002,3.952284
   16E-002,-1.05518714E-001,-1.27463877E-001,1.42578809E-002
   ```

Related Topics

- UTL_TO_EMBEDDING and UTL_TO_EMBEDDINGS
  Use the `DBMS_VECTOR.UTL_TO_EMBEDDING` and `DBMS_VECTOR.UTL_TO_EMBEDDINGS` chainable utility functions to generate one or more vector embeddings from textual documents and images.

- VECTOR_EMBEDDING
  Use `VECTOR_EMBEDDING` to generate a single vector embedding for different data types using embedding or feature extraction machine learning models.

Convert Text String to BINARY Embedding Outside Oracle Database

Perform a text-to-`BINARY`-embedding transformation by accessing a third-party `BINARY` vector embedding model.

> **WARNING:**
>
> Certain features of the database may allow you to access services offered separately by third-parties, for example, through the use of JSON specifications that facilitate your access to REST APIs.
>
> Your use of these features is solely at your own risk, and you are solely responsible for complying with any terms and conditions related to use of any such third-party services. Notwithstanding any other terms and conditions related to the third-party services, your use of such database features constitutes your acceptance of that risk and express exclusion of Oracle's responsibility or liability for any damages resulting from such access.

To generate a vector embedding with "`hello`" as the input using Cohere ubinary embed-english-v3.0 model:

1. Connect to Oracle Database as a local user.
   a. Log in to SQL*Plus as the `SYS` user, connecting as `SYSDBA`:

```
conn sys/password as sysdba


CREATE TABLESPACE tbs1
DATAFILE 'tbs5.dbf' SIZE 20G AUTOEXTEND ON
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;


SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 10000
SET LONG 10000
```

b. Create a local user (`docuser`) and grant necessary privileges:

```
DROP USER docuser cascade;


CREATE USER docuser identified by docuser DEFAULT TABLESPACE tbs1 quota
unlimited on tbs1;


GRANT DB_DEVELOPER_ROLE, create credential to docuser;
```

c. Connect as the local user (`docuser`):

```
CONN docuser/password
```

2. Set the HTTP proxy server, if configured.

```
EXEC UTL_HTTP.SET_PROXY('<proxy-hostname>:<proxy-port>');
```

3. Grant connect privilege to allow connection to the host.

   Grant connect privilege to `docuser` for connecting to the host, using the `DBMS_NETWORK_ACL_ADMIN` procedure. This example uses `*` to allow any host. However, you can explicitly specify the host that you want to connect to.

```
BEGIN
  DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE(
    host => '*',
    ace => xs$ace_type(privilege_list => xs$name_list('connect'),
                       principal_name => 'docuser',
                       principal_type => xs_acl.ptype_db));
END;
/
```

4. Set up your credentials for the REST provider (in this case, Cohere) and then call `UTL_TO_EMBEDDING`.

   a. Run `DBMS_VECTOR.CREATE_CREDENTIAL` to create and store a credential.

      Cohere requires the following authentication parameter:

```
{ "access_token": "<access token>" }
```

      Replace `<access token>` with your own values. You will later refer to this credential name when declaring JSON parameters for the `UTL_TO_EMBEDDING` call.

```
EXEC DBMS_VECTOR.DROP_CREDENTIAL('COHERE_CRED');

DECLARE
  jo json_object_t;
BEGIN
  jo := json_object_t();
  jo.put('access_token', '<access token>');
  DBMS_VECTOR.CREATE_CREDENTIAL(
    credential_name   => 'COHERE_CRED',
    params            => json(jo.to_string));
```

```
END;
/
```

b. Call `DBMS_VECTOR.UTL_TO_EMBEDDING` to generate the `BINARY` embedding.

> **Note:**
>
> For a list of all supported REST endpoints, see Supported Third-Party Provider Operations and Endpoints.

```
var params clob;

BEGIN
:params := '
             {
              "provider": "cohere",
              "credential_name": "COHERE_CRED",
              "url": "https://api.cohere.ai/v1/embed",
              "model": "embed-english-v3.0",
              "input_type": "search_query",
              "embedding_types": ["ubinary"]
             }';
END;
/

SELECT TO_VECTOR(FROM_VECTOR(DBMS_VECTOR.UTL_TO_EMBEDDING('hello',
JSON(:params))),*,BINARY);
```

The generated `BINARY` embedding appears as follows:

```
TO_VECTOR(FROM_VECTOR(DBMS_VECTOR.UTL_TO_EMBEDDING('HELLO',JSON(:PARAMS))),
*,BIN
--------------------------------------------------------------------------------
-----
[137,218,245,195,211,132,169,63,43,22,12,93,112,93,85,208,145,27,76,245,99,
222,1
21,63,1,161,200,24,1,30,202,233,208,2,113,27,119,78,123,192,132,115,187,146
,58,1
36,40,63,221,52,68,241,53,88,20,99,85,248,114,177,100,248,100,158,94,53,57,
97,18
2,129,14,64,173,236,107,109,37,195,173,49,128,113,204,183,158,55,139,10,205
,65,4
0,53,243,247,134,63,125,133,55,230,129,64,165,103,102,46,251,164,213,139,22
7,225
,66,98,112,100,64,145,98,80,97,192,149,77,43,114,146,197]
```

Related Topics

- UTL_TO_EMBEDDING and UTL_TO_EMBEDDINGS
  Use the `DBMS_VECTOR.UTL_TO_EMBEDDING` and `DBMS_VECTOR.UTL_TO_EMBEDDINGS` chainable utility functions to generate one or more vector embeddings from textual documents and images.

Convert Text String to Embedding Using Public REST Providers

Perform a text-to-embedding transformation, using publicly hosted third-party embedding models by Cohere, Generative AI, Google AI, Hugging Face, OpenAI, or Vertex AI.

You can use third-party embedding models to vectorize your data that is used to populate a vector index. Note that you must use the same embedding model on both the data to be indexed and the user's input query. In this example, you can see how to vectorize a user's input query on the fly.

Here, you can call the chainable utility function `UTL_TO_EMBEDDING` (note the singular "embedding") from either the `DBMS_VECTOR` or the `DBMS_VECTOR_CHAIN` package, depending on your use case. This example uses the `DBMS_VECTOR.UTL_TO_EMBEDDING` API.

`UTL_TO_EMBEDDING` directly returns a `VECTOR` type (not an array).

> **WARNING:**
>
> Certain features of the database may allow you to access services offered separately by third-parties, for example, through the use of JSON specifications that facilitate your access to REST APIs.
>
> Your use of these features is solely at your own risk, and you are solely responsible for complying with any terms and conditions related to use of any such third-party services. Notwithstanding any other terms and conditions related to the third-party services, your use of such database features constitutes your acceptance of that risk and express exclusion of Oracle's responsibility or liability for any damages resulting from such access.

To convert a user's input text "`hello`" to a vector embedding, using a public third-party embedding model:

1. Connect to Oracle Database as a local user.
   a. Log in to SQL*Plus as the `sys` user, connecting as `sysdba`:

```
conn sys/password as sysdba


CREATE TABLESPACE tbs1
DATAFILE 'tbs5.dbf' SIZE 20G AUTOEXTEND ON
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;


SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 10000
SET LONG 10000
```

b. Create a local user (`docuser`) and grant necessary privileges:

```
DROP USER docuser cascade;


CREATE USER docuser identified by docuser DEFAULT TABLESPACE tbs1 quota
unlimited on tbs1;


GRANT DB_DEVELOPER_ROLE, create credential to docuser;
```

c. Connect as the local user (`docuser`):

```
CONN docuser/password
```

2. Set the HTTP proxy server, if configured.

```
EXEC UTL_HTTP.SET_PROXY('<proxy-hostname>:<proxy-port>');
```

3. Grant connect privilege to `docuser` for allowing connection to the host, using the `DBMS_NETWORK_ACL_ADMIN` procedure.

   This example uses `*` to allow any host. However, you can explicitly specify the host that you want to connect to.

```
BEGIN
  DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE(
    host => '*',
    ace => xs$ace_type(privilege_list => xs$name_list('connect'),
                       principal_name => 'docuser',
                       principal_type => xs_acl.ptype_db));
END;
/
```

4. Set up your credentials for the REST provider that you want to access and then call `UTL_TO_EMBEDDING`.

   - **Using Generative AI**:

     a. Run `DBMS_VECTOR.CREATE_CREDENTIAL` to create and store an OCI credential (`OCI_CRED`).

        Generative AI requires the following authentication parameters:

```
{
"user_ocid"       : "<user ocid>",
"tenancy_ocid"    : "<tenancy ocid>",
"compartment_ocid": "<compartment ocid>",
"private_key"     : "<private key>",
"fingerprint"     : "<fingerprint>"
}
```

        You will later refer to this credential name when declaring JSON parameters for the `UTL_to_EMBEDDING` call.

> **Note:**
>
> The generated private key may appear as:
>
> ```
> -----BEGIN RSA PRIVATE KEY-----
> <private key string>
> -----END RSA PRIVATE KEY-----
> ```
>
> You pass the *<private key string>* value (excluding the `BEGIN` and `END` lines), either as a single line or as multiple lines.

```
exec dbms_vector.drop_credential('OCI_CRED');


declare
  jo json_object_t;
begin
  jo := json_object_t();
  jo.put('user_ocid','<user ocid>');
  jo.put('tenancy_ocid','<tenancy ocid>');
  jo.put('compartment_ocid','<compartment ocid>');
  jo.put('private_key','<private key>');
  jo.put('fingerprint','<fingerprint>');
  dbms_vector.create_credential(
    credential_name   => 'OCI_CRED',
    params            => json(jo.to_string));
end;
/
```

Replace all the authentication parameter values. For example:

```
declare
  jo json_object_t;
begin
  jo := json_object_t();

jo.put('user_ocid','ocid1.user.oc1..aabbalbbaa1112233aabbaabb1111222
aa1111bb');

jo.put('tenancy_ocid','ocid1.tenancy.oc1..aaaaalbbbb1112233aaaabbaa1
111222aaa111a');

jo.put('compartment_ocid','ocid1.compartment.oc1..ababalabab1112233a
babababb1111222aba11ab');
  jo.put('private_key','AAAaaaBBB11112222333...AAA111AAABBB222aaa1a/
+');

jo.put('fingerprint','01:1a:a1:aa:12:a1:12:1a:ab:12:01:ab:a1:12:ab:1
a');
  dbms_vector.create_credential(
    credential_name   => 'OCI_CRED',
    parameters        => json(jo.to_string));
```

```
end;
/
```

b. Call `DBMS_VECTOR.UTL_TO_EMBEDDING`:

Here, the cohere.embed-english-v3.0 model is used. You can replace `model` with your own value, as required.

> **Note:**
>
> For a list of all REST endpoint URLs and models that are supported to use with Generative AI, see Supported Third-Party Provider Operations and Endpoints.

```
-- select example

var params clob;
exec :params := '
{
  "provider": "ocigenai",
  "credential_name": "OCI_CRED",
  "url": "https://inference.generativeai.us-
chicago-1.oci.oraclecloud.com/20231130/actions/embedText",
  "model": "cohere.embed-english-v3.0",
  "batch_size": 10
}';

select dbms_vector.utl_to_embedding('hello', json(:params)) from
dual;

-- PL/SQL example

declare
  input clob;
  params clob;
  v vector;
begin
  input := 'hello';
  params := '
{
  "provider": "ocigenai",
  "credential_name": "OCI_CRED",
  "url": "https://inference.generativeai.us-
chicago-1.oci.oraclecloud.com/20231130/actions/embedText",
  "model": "cohere.embed-english-v3.0",
  "batch_size": 10
}';

  v := dbms_vector.utl_to_embedding(input, json(params));
  dbms_output.put_line(vector_serialize(v));
exception
  when OTHERS THEN
    DBMS_OUTPUT.PUT_LINE (SQLERRM);
    DBMS_OUTPUT.PUT_LINE (SQLCODE);
```

```
end;
/
```

- **Using Cohere, Google AI, Hugging Face, OpenAI, and Vertex AI**:

  a. Run `DBMS_VECTOR.CREATE_CREDENTIAL` to create and store a credential.

  Cohere, Google AI, Hugging Face, OpenAI, and Vertex AI require the following authentication parameter:

  ```
  { "access_token": "<access token>" }
  ```

  You will later refer to this credential name when declaring JSON parameters for the `UTL_to_EMBEDDING` call.

  ```
  exec dbms_vector.drop_credential('<credential name>');


  declare
    jo json_object_t;
  begin
    jo := json_object_t();
    jo.put('access_token', '<access token>');
    dbms_vector.create_credential(
      credential_name   => '<credential name>',
      params            => json(jo.to_string));
  end;
  /
  ```

  Replace the `access_token` and `credential_name` values. For example:

  ```
  declare
    jo json_object_t;
  begin
    jo := json_object_t();
    jo.put('access_token', 'AbabA1B123aBc123AbabAb123a1a2ab');
    dbms_vector.create_credential(
      credential_name   => 'HF_CRED',
      params            => json(jo.to_string));
  end;
  /
  ```

  b. Call `DBMS_VECTOR.UTL_TO_EMBEDDING`:

  ```
  -- select example

  var params clob;
  exec :params := '
  {
    "provider": "<REST provider>",
    "credential_name": "<credential name>",
    "url": "<REST endpoint URL for embedding service>",
    "model": "<embedding model name>"
  }';

  select dbms_vector.utl_to_embedding('hello', json(:params)) from
  dual;
  ```

```
-- PL/SQL example

declare
  input clob;
  params clob;
  v vector;
begin
  input := 'hello';

  params := '
{
  "provider": "<REST provider>",
  "credential_name": "<credential name>",
  "url": "<REST endpoint URL for embedding service>",
  "model": "<embedding model name>"
}';

  v := dbms_vector.utl_to_embedding(input, json(params));
  dbms_output.put_line(vector_serialize(v));
exception
  when OTHERS THEN
    DBMS_OUTPUT.PUT_LINE (SQLERRM);
    DBMS_OUTPUT.PUT_LINE (SQLCODE);
end;
/
```

> **Note:**
>
> For a complete list of all supported REST endpoint URLs, see Supported Third-Party Provider Operations and Endpoints.

Replace `provider`, `credential_name`, `url`, and `model` with your own values. Optionally, you can specify additional REST provider parameters. This is shown in the following examples:

Cohere example:

```
{
  "provider"       : "cohere",
  "credential_name": "COHERE_CRED",
  "url"            : "https://api.cohere.ai/v1/embed",
  "model"          : "embed-english-light-v2.0",
  "input_type"     : "search_query"
}
```

Google AI example:

```
{
  "provider"       : "googleai",
  "credential_name": "GOOGLEAI_CRED",
  "url"            : "https://generativelanguage.googleapis.com/
v1beta/models/",
```

```
  "model"          : "embedding-001"
}
```

Hugging Face example:

```
{
  "provider"       : "huggingface",
  "credential_name": "HF_CRED",
  "url"            : "https://api-inference.huggingface.co/pipeline/
feature-extraction/",
  "model"          : "sentence-transformers/all-MiniLM-L6-v2"
}
```

OpenAI example:

```
{
  "provider"       : "openai",
  "credential_name": "OPENAI_CRED",
  "url"            : "https://api.openai.com/v1/embeddings",
  "model"          : "text-embedding-3-small"
}
```

Vertex AI example:

```
{
  "provider"       : "vertexai",
  "credential_name": "VERTEXAI_CRED",
  "url"            : "https://LOCATION-aiplatform.googleapis.com/v1/
projects/PROJECT/locations/LOCATION/publishers/google/models/",
  "model"          : "textembedding-gecko:predict"
}
```

The generated embedding may appear as:

```
EMBEDDING
--------------------------------------------------------------------------------
--------------------------------------------------------
[8.78423732E-003,-4.29633334E-002,-5.93001908E-003,-4.65480909E-002,2.14333
013E-002,6.53376281E-002,-5.93746938E-002,2.10403297E-002,
4.38376889E-002,5.22960871E-002,1.25104953E-002,6.49512559E-002,-9.26998071
E-003,-6.97442219E-002,-3.02916039E-002,-4.74979728E-003,
-1.08755399E-002,-4.63751052E-003,3.62781435E-002,-9.35919806E-002,-1.13934
642E-002,-5.74270077E-002,-1.36667723E-002,2.42995787E-002,
-6.96804151E-002,4.93822657E-002,1.01460628E-002,-1.56464987E-002,-2.394105
68E-002,-4.27529104E-002,-5.65665103E-002,-1.74160264E-002,
5.05326502E-002,4.31500375E-002,-2.6994409E-002,-1.72731467E-002,9.30535868
E-002,6.85951149E-004,5.61876409E-003,-9.0233935E-003,
-2.55788807E-002,-2.04174276E-002,3.74175981E-002,-1.67872179E-002,1.074793
04E-001,-6.64602639E-003,-7.65537247E-002,-9.71965566E-002,
-3.99636962E-002,-2.57076006E-002,-5.62455431E-002,-1.3583754E-001,3.459460
29E-002,1.85191762E-002,3.01524661E-002,-2.62163244E-002,
-4.05582506E-003,1.72979087E-002,-3.66434865E-002,-1.72491539E-002,3.952284
16E-002,-1.05518714E-001,-1.27463877E-001,1.42578809E-002
```

This example uses the default settings for each provider. For detailed information on additional parameters, refer to your third-party provider's documentation.

Related Topics

- UTL_TO_EMBEDDING and UTL_TO_EMBEDDINGS
  Use the `DBMS_VECTOR.UTL_TO_EMBEDDING` and `DBMS_VECTOR.UTL_TO_EMBEDDINGS` chainable utility functions to generate one or more vector embeddings from textual documents and images.
- DBMS_VECTOR
  The `DBMS_VECTOR` package simplifies common operations with Oracle AI Vector Search, such as extracting chunks or embeddings from user data, generating text for a given prompt or an image, creating a vector index, or reporting on index accuracy.
- DBMS_VECTOR_CHAIN
  The `DBMS_VECTOR_CHAIN` package enables advanced operations with Oracle AI Vector Search, such as chunking and embedding data along with text generation and summarization capabilities. It is more suitable for text processing with similarity search and hybrid search, using functionality that can be pipelined together for an end-to-end search.

Convert Text String to Embedding Using the Local REST Provider Ollama

Perform a text-to-embedding transformation by accessing open embedding models, using the local host REST endpoint provider Ollama.

Ollama is a free and open-source command-line interface tool that allows you to run open embedding models and LLMs locally and privately on your Linux, Windows, or macOS systems. You can access Ollama as a service using SQL and PL/SQL commands.

You can call embedding models, such as all-minilm, mxbai-embed-large, or nomic-embed-text to vectorize your data. Note that you must use the same embedding model on both the data to be indexed and the user's input query. In this example, you can see how to vectorize a user's input query on the fly.

Here, you can call the chainable utility function `UTL_TO_EMBEDDING` (note the singular "embedding") from either the `DBMS_VECTOR` or the `DBMS_VECTOR_CHAIN` package, depending on your use case. This example uses the `DBMS_VECTOR.UTL_TO_EMBEDDING` API. `UTL_TO_EMBEDDING` directly returns a `VECTOR` type (not an array).

> **WARNING:**
>
> Certain features of the database may allow you to access services offered separately by third-parties, for example, through the use of JSON specifications that facilitate your access to REST APIs.
>
> Your use of these features is solely at your own risk, and you are solely responsible for complying with any terms and conditions related to use of any such third-party services. Notwithstanding any other terms and conditions related to the third-party services, your use of such database features constitutes your acceptance of that risk and express exclusion of Oracle's responsibility or liability for any damages resulting from such access.

To convert a user's input text string "`hello`" to a query vector, by calling a local embedding model using Ollama:

1. Connect to Oracle Database as a local user.
   a. Log in to SQL*Plus as the `SYS` user, connecting as `SYSDBA`:

```
conn sys/password as sysdba


CREATE TABLESPACE tbs1
DATAFILE 'tbs5.dbf' SIZE 20G AUTOEXTEND ON
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;


SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 10000
SET LONG 10000
```

   b. Create a local user (`docuser`) and grant necessary privileges:

```
DROP USER docuser cascade;


CREATE USER docuser identified by docuser DEFAULT TABLESPACE tbs1 quota
unlimited on tbs1;


GRANT DB_DEVELOPER_ROLE, create credential to docuser;
```

   c. Connect as the local user (`docuser`):

```
CONN docuser/password
```

2. Install Ollama and run an embedding model locally.
   a. Download and run the Ollama application from https://ollama.com/download.

      You can either install Ollama as a service that runs in the background or as a standalone binary with a manual install. For detailed installation-specific steps, see Quick Start in the Ollama Documentation.

      Note the following:

      - The Ollama server needs to be able to connect to the internet so that it can download the models. If you require a proxy server to access the internet, remember to set the appropriate environment variables before running the Ollama server. For example, to set for Linux:

```
-- set a proxy if you require one

export https_proxy=<proxy-hostname>:<proxy-port>
export http_proxy=<proxy-hostname>:<proxy-port>
export no_proxy=localhost,127.0.0.1,.example.com
export ftp_proxy=<proxy-hostname>:<proxy-port>
```

- If you are running Ollama and the database on different machines, then on the database machine, you must change the `URL` to refer to the host name or IP address that is running Ollama instead of the local host.
- You may need to change your firewall settings on the machine that is running Ollama to allow the port through.

b. If running Ollama as a standalone binary from a manual install, then start the server:

```
ollama serve
```

c. Run a model using the `ollama pull <embedding_model_name>` command.

For example, to call the all-minilm model:

```
ollama pull all-minilm
```

For detailed information on this step, see Ollama Readme.

d. Verify that Ollama is running locally by using a cURL command.

For example:

```
-- get embeddings

curl http://localhost:11434/api/embeddings -d '{
  "model" : "all-minilm",
  "prompt": "What is Oracle AI Vector Search?"}'
```

3. Set the HTTP proxy server, if configured.

```
EXEC UTL_HTTP.SET_PROXY('<proxy-hostname>:<proxy-port>');
```

4. Grant connect privilege to `docuser` for allowing connection to the host, using the `DBMS_NETWORK_ACL_ADMIN` procedure.

This example uses `*` to allow any host. However, you can explicitly specify the host that you want to connect to.

```
BEGIN
  DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE(
    host => '*',
    ace => xs$ace_type(privilege_list => xs$name_list('connect'),
                       principal_name => 'docuser',
                       principal_type => xs_acl.ptype_db));
END;
/
```

5. Call `UTL_TO_EMBEDDING`.

The Ollama service has a REST API endpoint for generating embedding. Specify the URL and other configuration parameters in a JSON object.

```
var embed_ollama_params clob;
exec :embed_ollama_params := '{
     "provider": "ollama",
     "host"    : "local",
     "url"     : "http://localhost:11434/api/embeddings",
```

```
     "model"   : "all-minilm"
}';

select dbms_vector.utl_to_embedding('hello', json(:embed_ollama_params))
ollama_output from dual;
```

You can replace the `url` and `model` with your own values, as required.

> **Note:**
>
> For a complete list of all supported REST endpoint URLs, see Supported Third-Party Provider Operations and Endpoints.

An excerpt from the generated embedding output is as follows:

```
OLLAMA_OUTPUT
--------------------------------------------------------------------------------
-----
[-2.31221581E+000,-3.26045007E-001,2.48111725E-001,-1.65610778E+000,1.10871
601E+
000,-1.78519666E-001,-2.44365096E+000,-3.32534742E+000,-1.3771069E+000,1.88
42382
4E+000,1.26494539E+000,-2.05359578E+000,-1.78593469E+000,-3.16150457E-001,-
5.362
42545E-001,6.42113638E+000,-2.36518741E+000,2.21405053E+000,6.52316332E-001
,-7.8
1692028E-001,2.32031775E+000,5.31627655E-001,-5.02781868E-001,7.03743398E-0
01,5.
48391223E-001,-3.16579014E-001,5.28999329E+000,1.63369191E+000,1.34206653E-
001,9
.54429448E-001,-1.94197679E+000,2.39797616E+000,3.5270077E-001,-1.6536833E+
000,-
5.74707508E-001,1.60994816E+000,3.80332375E+000,-6.30351126E-001,-1.5865227
E+000
,-2.48650503E+000,-1.42142653E+000,-2.79453158E+000,1.76355612E+000,-2.4869
0337E
-001,1.5521245E+000,-1.95240334E-001,1.42441893E+000,-3.57098508E+000,4.020
83158
E+000,-2.38530707E+000,2.34579134E+000,-2.79158998E+000,-5.92314243E-001,-9
.7153

OLLAMA_OUTPUT
--------------------------------------------------------------------------------
-----
9557E-001,1.6514441E-002,1.03710043E+000,1.96799666E-001,-2.18394065E+000,-
2.786
71598E+000,-1.1549623E+000,1.92903787E-001,7.72498465E+000,-1.63462329E+000
,3.33
839393E+000,-7.17389703E-001,-3.99817854E-001,7.7395606E-001,6.43829286E-00
1,1.8
5182285E+000,2.95272923E+000,-8.72635692E-002,1.77895337E-001,-1.19716954E+
000,9
.43063736E-001,1.51703429E+000,-7.93301344E-001,1.92319798E+000,3.33290434E
```

```
+000,
-1.29852867E+000,2.02961493E+000,-4.35824203E+000,1.30186975E+000,-1.097941
4E+00
0,-7.07521975E-001,-4.50183332E-001,3.77224755E+000,-2.49138021E+000,-1.125
75901
E+000,1.15370192E-001,1.66767395E+000,3.52229095E+000,-3.78049397E+000,-6.1
71851
75E-001,1.71992481E+000,2.33226371E+000,-3.35983014E+000,-3.78417182E+000,3
.8380
127E+000,2.59293962E+000,-1.33574629E+000,-1.76218402E+000,3.53816301E-001,
-1.80
47899E+000,9.85167921E-001,1.93026745E+000,2.15959966E-001,9.94020045E-001,
6.189
```

Related Topics

- UTL_TO_EMBEDDING and UTL_TO_EMBEDDINGS
  Use the `DBMS_VECTOR.UTL_TO_EMBEDDING` and `DBMS_VECTOR.UTL_TO_EMBEDDINGS` chainable utility functions to generate one or more vector embeddings from textual documents and images.
- DBMS_VECTOR
  The `DBMS_VECTOR` package simplifies common operations with Oracle AI Vector Search, such as extracting chunks or embeddings from user data, generating text for a given prompt or an image, creating a vector index, or reporting on index accuracy.
- DBMS_VECTOR_CHAIN
  The `DBMS_VECTOR_CHAIN` package enables advanced operations with Oracle AI Vector Search, such as chunking and embedding data along with text generation and summarization capabilities. It is more suitable for text processing with similarity search and hybrid search, using functionality that can be pipelined together for an end-to-end search.

Convert Image to Embedding Using Public REST Providers

Perform an image-to-embedding transformation by making a REST call to the third-party service provider, Vertex AI. In this example, you can see how to vectorize both image and text inputs using a multimodal embedding model and then query a vector space containing vectors from both content types.

You can directly generate a vector embedding based on an image, which can be used for classifying or detecting images, comparing large datasets of images, or for performing a more effective similarity search on documents that include images. To get image embeddings, you can use any image embedding model or multimodal embedding model supported by Vertex AI. By analyzing an image, a model generates image embedding that encodes each visual element of the image (shape, color, pattern, texture, action, or object) as a vector representation.

Multimodal embedding is a technique that vectorizes data from different modalities such as text and images. This lets you use the same embedding model to generate embeddings for both types of content. By doing so, the resulting embeddings are compatible and situated in the same vector space, which allows for effective comparison between the two modalities (text and image) during similarity searches.

Here, you can use the `UTL_TO_EMBEDDING` function from either the `DBMS_VECTOR` or the `DBMS_VECTOR_CHAIN` package.

> **WARNING:**
>
> Certain features of the database may allow you to access services offered separately by third-parties, for example, through the use of JSON specifications that facilitate your access to REST APIs.
>
> Your use of these features is solely at your own risk, and you are solely responsible for complying with any terms and conditions related to use of any such third-party services. Notwithstanding any other terms and conditions related to the third-party services, your use of such database features constitutes your acceptance of that risk and express exclusion of Oracle's responsibility or liability for any damages resulting from such access.

To find a bird that is similar to a given image or text input, using similarity search:

1. Connect as a local user and prepare your data dump directory.
   a. Log in to SQL*Plus as the `SYS` user, connecting as `SYSDBA`:

```
conn sys/password as sysdba


CREATE TABLESPACE tbs1
DATAFILE 'tbs5.dbf' SIZE 20G AUTOEXTEND ON
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;


SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 10000
SET LONG 10000
```

   b. Create a local user (`docuser`) and grant necessary privileges:

```
drop user docuser cascade;


create user docuser identified by docuser DEFAULT TABLESPACE tbs1 quota
unlimited on tbs1;


grant DB_DEVELOPER_ROLE to docuser;
```

c. Create a local directory (`VEC_DUMP`) to store image files. Grant necessary privileges:

```
create or replace directory VEC_DUMP as '/my_local_dir/';


grant read, write on directory VEC_DUMP to docuser;

commit;
```

d. Connect as the local user (`docuser`):

```
conn docuser/password
```

2. Set the HTTP proxy server, if configured.

```
EXEC UTL_HTTP.SET_PROXY('<proxy-hostname>:<proxy-port>');
```

3. Grant connect privilege to `docuser` for allowing connection to the host, using the `DBMS_NETWORK_ACL_ADMIN` procedure.

   This example uses `*` to allow any host. However, you can explicitly specify the host that you want to connect to.

```
BEGIN
  DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE(
    host => '*',
    ace => xs$ace_type(privilege_list => xs$name_list('connect'),
                       principal_name => 'docuser',
                       principal_type => xs_acl.ptype_db));
END;
/
```

4. Set up credentials for Vertex AI.

   Vertex AI requires the following authentication parameter:

```
{ "access_token": "<access token>" }

begin
  DBMS_VECTOR_CHAIN.DROP_CREDENTIAL(credential_name  => 'VERTEXAI_CRED');
exception
  when others then null;
end;
/
```

   Here, replace `<access token>` with your own value:

```
declare
  jo json_object_t;
begin
  jo := json_object_t();
  jo.put('access_token', '<access token>');
  DBMS_VECTOR_CHAIN.CREATE_CREDENTIAL(
    credential_name   =>  'VERTEXAI_CRED',
    params            => json(jo.to_string));
```

```
end;
/
```

5. Create a relational table (`docs`) and insert some images in it.

   You will later compare your query image to this database of images.

   Here, you first create a `docs` table with `id` and `content` columns. Then, using the `to_blob` function, you convert the contents of your image files into `BLOB` for storing into `content` (blob column), reading the files from the `VEC_DUMP` directory.

```
drop table docs;


create table docs(id number primary key, content blob);

insert into docs(id, content)
values(1, to_blob(bfilename('VEC_DUMP', 'cat.jpg')));
insert into docs (id, content)
values(2, to_blob(bfilename('VEC_DUMP', 'eagle.jpg')));
```

6. Get image embedding for your query image (`parrots.jpg`):

   First, upload your query image to the `VEC_DUMP` directory.

   Then, use `UTL_TO_EMBEDDING` to vectorize that image. Here, the input is specified as `parrots.jpg` (with a pointer to the `VEC_DUMP` directory), the modality is specified as `image`, and the provider-specific embedding parameters for Vertex AI are passed as JSON.

> **Note:**
>
> For a list of all supported REST endpoints, see Supported Third-Party Provider Operations and Endpoints.

```
-- declare embedding parameters

var params clob;

begin
  :params := '
{
  "provider": "vertexai",
  "credential_name": "VERTEXAI_CRED",
  "url": "https://LOCATION-aiplatform.googleapis.com/v1/projects/PROJECT/
locations/LOCATION/publishers/google/models/",
  "model": "multimodalembedding:predict"
}';
end;
/

-- get image embedding: PL/SQL example

declare
  v vector;
  output clob;
```

```
begin
  v := dbms_vector_chain.utl_to_embedding(
    to_blob(bfilename('VEC_DUMP', 'parrots.jpg')), 'image', json(:params));
  output := vector_serialize(v);
  dbms_output.put_line('vector data=' || dbms_lob.substr(output, 100) ||
'...');
end;
/

-- get image embedding: select example

select dbms_vector_chain.utl_to_embedding(
  to_blob(bfilename('VEC_DUMP', 'parrots.jpg')), 'image', json(:params));
```

7. Search using an image input. Here, input is the generated embedding for your query image (`parrots.jpg`):

```
select docs.id, vector_distance(
  dbms_vector_chain.utl_to_embedding(to_blob(bfilename('VEC_DUMP',
'parrots.jpg')), 'image', json(:params)),
  dbms_vector_chain.utl_to_embedding(docs.content, 'image', json(:params)),
  cosine) dist
from docs
order by dist asc;
```

An output appears as:

```
ID       DIST
-------
-----------
2
5.847E-001
1        6.295E-001
```

This query shows that `ID 2` (`eagle.jpg`) is the closest match to the given image input, while `ID 1` (`cat.jpg`) is less similar.

8. Search using a text input. Here, input is the image's description as "`bald eagle`":

```
select docs.id, vector_distance(
  dbms_vector_chain.utl_to_embedding('bald eagle', json(:params)),
  dbms_vector_chain.utl_to_embedding(docs.content, 'image', json(:params)),
  cosine) dist
from docs
order by dist asc;
```

An output appears as:

```
ID      DIST
------- -----------
2       8.449E-001
1       9.87E-001
```

This query also implies that `ID 2` is the closest match to your text input, while `ID 1` is less similar. However, both values are lower than those in the previous query where you specified the image as input.

Related Topics

- UTL_TO_EMBEDDING and UTL_TO_EMBEDDINGS
  Use the `DBMS_VECTOR_CHAIN.UTL_TO_EMBEDDING` and `DBMS_VECTOR_CHAIN.UTL_TO_EMBEDDINGS` chainable utility functions to generate one or more vector embeddings from textual documents and images.
- DBMS_VECTOR
  The `DBMS_VECTOR` package simplifies common operations with Oracle AI Vector Search, such as extracting chunks or embeddings from user data, generating text for a given prompt or an image, creating a vector index, or reporting on index accuracy.
- DBMS_VECTOR_CHAIN
  The `DBMS_VECTOR_CHAIN` package enables advanced operations with Oracle AI Vector Search, such as chunking and embedding data along with text generation and summarization capabilities. It is more suitable for text processing with similarity search and hybrid search, using functionality that can be pipelined together for an end-to-end search.

Generate Multi-modal Embeddings Using CLIP

This section provides end-to-end instructions from installing the OML4Py client to generating multi-modal embeddings using CLIP.

These instructions assume you have configured your Oracle Linux 8 repo in `/etc/yum.repos.d`, configured a Wallet if using an Autonomous Database, and set up a proxy if needed.

1. Install Python:

```
sudo yum install libffi-devel openssl openssl-devel tk-devel xz-devel zlib-
devel bzip2-devel readline-devel libuuid-devel ncurses-devel libaio
mkdir -p $HOME/python
wget https://www.python.org/ftp/python/3.12.3/Python-3.12.3.tgz
tar -xvzf Python-3.12.3.tgz --strip-components=1 -C $HOME/python
cd $HOME/python
./configure --prefix=$HOME/python
make clean; make
make altinstall
```

2. Set variables `PYTHONHOME`, `PATH`, and `LD_LIBRARY_PATH`:

```
export PYTHONHOME=$HOME/python
export PATH=$PYTHONHOME/bin:$PATH
export LD_LIBRARY_PATH=$PYTHONHOME/lib:$LD_LIBRARY_PATH
```

3. Create symlink for python3 and pip3:

```
cd $HOME/python/bin
ln -s python3.12 python3
ln -s pip3.12 pip3
```

4. Install Oracle Instant client if you will be exporting embedded models to the database from Python. If you will be exporting to a file, skip steps 4 and 5 and see the note under environment variables in step 6:

```
cd $HOME
wget https://download.oracle.com/otn_software/linux/instantclient/2340000/
instantclient-basic-linux.x64-23.4.0.24.05.zip
unzip instantclient-basic-linux.x64-23.4.0.24.05.zip
```

5. Set variable `LD_LIBRARY_PATH`:

```
export LD_LIBRARY_PATH=$HOME/instantclient_23_4:$LD_LIBRARY_PATH
```

6. Create an environment file, for example, `env.sh`, that defines the Python and Oracle Instant client environment variables and source these environment variables before each OML4Py client session. Alternatively, add the environment variable definitions to `.bashrc` so they are defined when the user logs into their Linux machine.

```
# Environment variables for Python
export PYTHONHOME=$HOME/python
export PATH=$PYTHONHOME/bin:$PATH
export LD_LIBRARY_PATH=$PYTHONHOME/lib:$LD_LIBRARY_PATH
```

> **Note:**
>
> Environment variable for Oracle Instant Client - only if the Oracle Instant Client is installed for exporting models to the database.
> `export LD_LIBRARY_PATH=$HOME/instantclient_23_4:$LD_LIBRARY_PATH`
>
> .

7. Create a file named requirements.txt that contains the required third-party packages listed below.

```
--extra-index-url https://download.pytorch.org/whl/cpu
pandas==2.1.1
setuptools==68.0.0
scipy==1.12.0
matplotlib==3.8.4
oracledb==2.2.0
scikit-learn==1.4.1.post1
numpy==1.26.4
onnxruntime==1.17.0
onnxruntime-extensions==0.10.1
onnx==1.16.0
torch==2.2.0+cpu
transformers==4.38.1
sentencepiece==0.2.0
```

8. Upgrade pip3 and install the packages listed in requirements.txt.

```
pip3 install --upgrade pip
pip3 install -r requirements.txt
```

9. Install OML4Py client. Download OML4Py 2.0.1 client from OML4Py download page and upload it to the Linux machine.

```
unzip oml4py-client-linux-x86_64-2.0.1.zip
pip3 install client/oml-2.0-cp312-cp312-linux_x86_64.whl
```

10. Get a list of all preconfigured models. Start Python and import ONNXPipelineConfig from `oml.utils`.

```
python3

from oml.utils import ONNXPipelineConfig

ONNXPipelineConfig.show_preconfigured()


['sentence-transformers/all-mpnet-base-v2',
'sentence-transformers/all-MiniLM-L6-v2',
'sentence-transformers/multi-qa-MiniLM-L6-cos-v1',
'sentence-transformers/distiluse-base-multilingual-cased-v2',
'sentence-transformers/all-MiniLM-L12-v2',
'BAAI/bge-small-en-v1.5',
'BAAI/bge-base-en-v1.5',
'taylorAI/bge-micro-v2',
'intfloat/e5-small-v2',
'intfloat/e5-base-v2',
'thenlper/gte-base',
'thenlper/gte-small',
'TaylorAI/gte-tiny',
'sentence-transformers/paraphrase-multilingual-mpnet-base-v2',
'intfloat/multilingual-e5-base',
'intfloat/multilingual-e5-small',
'sentence-transformers/stsb-xlm-r-multilingual',
'Snowflake/snowflake-arctic-embed-xs',
'Snowflake/snowflake-arctic-embed-s',
'Snowflake/snowflake-arctic-embed-m',
'mixedbread-ai/mxbai-embed-large-v1',
'openai/clip-vit-large-patch14',
'google/vit-base-patch16-224',
'microsoft/resnet-18',
'microsoft/resnet-50',
'WinKawaks/vit-tiny-patch16-224',
'Falconsai/nsfw_image_detection',
'WinKawaks/vit-small-patch16-224',
'nateraw/vit-age-classifier',
'rizvandwiki/gender-classification',
'AdamCodd/vit-base-nsfw-detector',
'trpakov/vit-face-expression',
'BAAI/bge-reranker-base']
```

11. Use OML4Py to load a multi-modal model on the database.

    - To use an alternate method other than OML4Py, skip this step and proceed to step 12.

    Export a preconfigured embedding model to the database. Import the oml library and import `ONNXPipeline` and `ONNXPipelineConfig` from `oml.utils`. This exports the ONNX-

format model to your local file system. In the following steps, replace the placeholders with your own credentials.

```
import oml
from oml.utils import ONNXPipeline, ONNXPipelineConfig
```

If your Oracle Database is on premises, set embedded mode to false. This step is not supported or required for Oracle Autonomous Database.

```
oml.core.methods.__embed__ = False
```

Create a database connection.

- Using Oracle Database on premises:

  ```
  oml.connect("<user>", "<password>", port=<port number>
  host="<hostname>",
  service_name="<service name>")

  pipeline = ONNXPipeline(model_name="openai/clip-vit-large-patch14")
  pipeline.export2db("CLIP")
  ```

- Using Oracle Autonomous Database:

  ```
  oml.connect(user="<user>", password="<password>", dsn="myadb_low")

  pipeline = ONNXPipeline(model_name="openai/clip-vit-large-patch14")
  pipeline.export2db("CLIP")
  ```

Once this step is complete, there will be two models loaded on the database called "CLIP_TXT" and "CLIP_IMG".

12. Export a preconfigured embedding model to a local file.

    This exports the ONNX-format model to your local file system:

    ```
    # Export to file
    pipeline = ONNXPipeline(model_name="openai/clip-vit-large-patch14")
    pipeline.export2file("clip",output_dir="/tmp/models")
    ```

    Move the ONNX file to a directory on the database server and create a directory on the file system and in the database for the import.

    ```
    mkdir -p /tmp/models
    sqlplus / as sysdba
    alter session set container=<name of pluggable database>;
    ```

    Apply the necessary permissions and grants.

    ```
    -- directory to store ONNX files for import
    CREATE DIRECTORY ONNX_IMPORT AS '/tmp/models';
    -- grant your OML user read and write permissions on the directory
    GRANT READ, WRITE ON DIRECTORY ONNX_IMPORT to OMLUSER;
    ```

```
-- grant to allow user to import the model
GRANT CREATE MINING MODEL TO OMLUSER;
```

Use the `DBMS_VECTOR.LOAD_ONNX_MODEL` procedure to load the models in your OML user schema. In this example, the procedure loads the ONNX model files named `clip_txt.onnx` and `clip_img.onnx` from the `ONNX_IMPORT` directory into the database as models named `CLIP_TXT` and `CLIP_IMG`, respectively.

```
BEGIN
    DBMS_VECTOR.LOAD_ONNX_MODEL(
    directory => 'ONNX_IMPORT',
    file_name => 'clip_txt.onnx',
    model_name => 'CLIP_TXT',
    metadata => JSON('{"function" : "embedding", "embeddingOutput" :
"embedding", "input": {"input": ["DATA"]}}'));
END;
BEGIN
    DBMS_VECTOR.LOAD_ONNX_MODEL(
    directory => 'ONNX_IMPORT',
    file_name => 'clip_img.onnx',
    model_name => 'CLIP_IMG',
    metadata => JSON('{"function" : "embedding", "embeddingOutput" :
"embedding", "input": {"input": ["DATA"]}}'));
END;
```

13. Verify the models exist using SQL.

```
sqlplus $USER/pass@PDBNAME;


SELECT model_name, algorithm, mining_function
FROM user_mining_models
WHERE model_name='CLIP_TXT' OR model_name='CLIP_IMG';


------------------------------------------------------------
MODEL_NAME          ALGORITHM            MINING_FUNCTION
------------------------------------------------------------
CLIP_TXT            ONNX                 EMBEDDING
CLIP_IMG            ONNX                 EMBEDDING
```

14. Generate embeddings with exported models using Python.

```
from oracledb import DB_TYPE_BLOB
with open('cat.jpg', 'rb') as f:
    img = f.read()
cr = oml.cursor()
blob = cr. var(DB_TYPE_BLOB)
blob.setvalue(0, img)
data = cr.execute("select vector_embedding(CLIP_TXT using 'RES' as DATA)
from dual")
txt_embed = data.fetchall()
data = cr.execute("select vector_embedding(CLIP_IMG using to_blob(:1) as
```

```
DATA) from dual", [blob])
img_embed = data.fetchall()
```

Calculate similarity between an image and text using Python:

```
from oracledb import DB_TYPE_BLOB
with open('cat.jpg', 'rb') as f:
    img = f.read()
cr = oml.cursor()
blob = cr.var(DB_TYPE_BLOB)
blob.setvalue(0, img)
data = cr.execute("""select 1-vector_distance(vector_embedding(CLIP_TXT
using 'RES' as DATA),
                    vector_embedding(CLIP_IMG using to_blob(:1) as DATA))
from dual""", [blob])
data.fetchall()
```

Result:

```
[(0.1637756726800217,)]
```

Generate embeddings with the exported models using SQL:

- ```
  SELECT VECTOR_EMBEDDING(CLIP_TXT USING 'RES' as DATA) AS embedding;
  ```

  An example of results are shown in the following excerpt:

  ```
  EMBEDDING
  ----------------------------------------------------------------------------
  --------
  [2.86132172E-002,-5.59654366E-003,8.66401661E-003,-1.4299524E-002,1.0201
  2949E-00
  2,6.00034464E-003,1.86244473E-002,-7.81036681E-003, ...
  ```

- ```
  SELECT VECTOR_EMBEDDING(CLIP_IMG USING TO_BLOB(BFILENAME('ONNX_IMPORT',
  'cat.jpg')) as DATA) AS embedding;
  ```

  An example of results are shown in the following excerpt:

  ```
  EMBEDDING
  ----------------------------------------------------------------------------
  --------
  [-2.2028232E-002,1.29058748E-003,-2.0222881E-004,4.58140159E-003,1.98919
  605E-002
  ,-9.51210782E-003,8.22519697E-003,1.06151737E-002, ...
  ```

Calculate the similarity between an image and text using SQL:

```
SELECT 1-VECTOR_DISTANCE(VECTOR_EMBEDDING(
            CLIP_IMG USING TO_BLOB(BFILENAME('ONNX_IMPORT', 'cat.jpg')) as
DATA),
            VECTOR_EMBEDDING(CLIP_TXT USING 'RES' as DATA)) AS similarity;
```

Example result:

```
SIMILARITY
----------
1.638E-001
```

Vectorize Relational Tables Using OML Feature Extraction Algorithms

This example shows you how to use OML's Feature Extraction algorithms in conjunction with the `VECTOR_EMBEDDING()` operator to vectorize sets of relational data, build similarity indexes, and perform similarity searches on the resulting vectors.

Feature Extraction algorithms help in extracting the most informative features/columns from the data and aim to reduce the dimensionality of large data sets by identifying the principal components that capture the most variance in the data. This reduction simplifies the data set while retaining the most important information, making it easier to analyze correlations and redundancies in the data.

The Principal Component Analysis (PCA) algorithm, a widely used dimensionality reduction technique in machine learning, is used in this tutorial.

Note:

This example uses customer bank marketing data available at https://archive.ics.uci.edu/dataset/222/bank+marketing.

The relational data table includes a mix of numeric and categorical columns. It has more than 4000 records.

```
SELECT column_name, data_type
FROM user_tab_columns
WHERE table_name = 'BANK'
ORDER BY data_type, column_name;
```

Output:

```
COLUMN_NAME          DATA_TYPE
-------------------- --------------------
AGE                  NUMBER
CAMPAIGN             NUMBER
CONS_CONF_IDX        NUMBER
CONS_PRICE_IDX       NUMBER
DURATION             NUMBER
EMP_VAR_RATE         NUMBER
EURIBOR3M            NUMBER
ID                   NUMBER
NR_EMPLOYED          NUMBER
PDAYS                NUMBER
PREVIOUS             NUMBER
CONTACT              VARCHAR2
CREDIT_DEFAULT       VARCHAR2
DAY_OF_WEEK          VARCHAR2
EDUCATION            VARCHAR2
HOUSING              VARCHAR2
JOB                  VARCHAR2
LOAN                 VARCHAR2
MARITAL              VARCHAR2
MONTH                VARCHAR2
POUTCOME             VARCHAR2
Y                    VARCHAR2
```

To perform a similarity search, you need to vectorize the relational data. To do so, you can first use the OML Feature Extraction algorithm to project the data onto a more compact numeric space. In this example, you configure the SVD algorithm to perform a Principal Component Analysis (PCA) projection of the original data table. The number of features/columns (5 in this case) is specified in the setting table. The input number determines the number of principal features or columns that will be retained after the dimensionality reduction process. Each of these columns represent a direction in the feature space along which the data varies the most.

Because you need to use the `DBMS_DATA_MINING` package to create the model, you need the `CREATE MINING MODEL` privilege in addition to the other privileges relevant to vector indexes and similarity search. For more information about the `CREATE MINING MODEL` privilege, see *Oracle Machine Learning for SQL User's Guide*.

1. Create a setting table, insert values, and then create a model.

   Use the `DBMS_DATA_MINING` package to create a model, using `mod_sett` as the setting table:

```
CREATE TABLE mod_sett(
  setting_name  VARCHAR2(30),
  setting_value VARCHAR2(30)
);


BEGIN
  INSERT INTO mod_sett (setting_name, setting_value) VALUES
        (dbms_data_mining.algo_name,
dbms_data_mining.algo_singular_value_decomp);
  INSERT INTO mod_sett (setting_name, setting_value) VALUES
        (dbms_data_mining.prep_auto, dbms_data_mining.prep_auto_on);
  INSERT INTO mod_sett (setting_name, setting_value) VALUES
        (dbms_data_mining.svds_scoring_mode,
dbms_data_mining.svds_scoring_pca);
  INSERT INTO mod_sett (setting_name, setting_value) VALUES
        (dbms_data_mining.feat_num_features, 5);
  commit;
END;
/


BEGIN
  DBMS_DATA_MINING.CREATE_MODEL(
    model_name          => 'pcamod',
    mining_function     => dbms_data_mining.feature_extraction,
    data_table_name     => 'bank',
    case_id_column_name => 'id',
    settings_table_name => 'mod_sett');
END;
/
```

2. Use the `VECTOR_EMBEDDING()` function to output the SVD projection results as vectors.

   The dimension of the vector column is the same as the number of features in the PCA model and the value of the vector represents the PCA projection results of the original row data.

> **Note:**
>
> `USING *` results in the use of all the relevant features present in the input row.

```
SELECT id, vector_embedding(pcamod USING *) embedding
FROM bank
WHERE id=10000;
```

Output:

```
            ID      EMBEDDING
--------------     --------------------------------------------------
         10000     [-2.3551013972411354E+002,2.8160084506788273E+001,
                     5.2821278275005774E+001,-1.8960922352439308E-002,
                    -2.5441143639048378E+000]
```

3. Create a table to hold the vector output results for all the data records.

   This represents a vectorization of your original relational data for the top-5 most important columns.

```
CREATE TABLE pca_output AS
(SELECT id, vector_embedding(pcamod USING *) embedding
 FROM bank);
```

4. Build a vector index based on the vector table.

   For example, you can build an IVF index with cosine distance:

```
CREATE VECTOR INDEX my_ivf_idx ON pca_output(embedding)
ORGANIZATION NEIGHBOR PARTITIONS
DISTANCE COSINE WITH TARGET ACCURACY 95;
```

5. Perform a similarity search using the `my_ivf_idx` index.

   In this example, you search for the top-3 results that are closest to `id=10000` based on the cosine distance and join the vector table with the original bank table to retrieve the most impactful attributes from the original table. To identify the most impactful columns for this row, use the `FEATURE_DETAILS()` function.

```
SELECT feature_details(pcamod, 5 USING *) features
FROM bank
WHERE id=10000;
```

   Output:

```
FEATURES
--------------------------------------------------------------------------------
--------------------
<Details algorithm="Singular Value Decomposition" feature="5">
<Attribute name="PDAYS" actualValue="999" weight=".041" rank="1"/>
<Attribute name="EURIBOR3M" actualValue="4.959" weight=".028" rank="2"/>
<Attribute name="CONTACT" actualValue="telephone" weight=".016" rank="3"/>
<Attribute name="EMP_VAR_RATE" actualValue="1.4" weight=".014" rank="4"/>
<Attribute name="DAY_OF_WEEK" actualValue="wed" weight=".002" rank="5"/>
</Details>
```

   Join the original data table to retrieve the most impactful information:

```
SELECT p.id id, b.PDAYS PDAYS, b.EURIBOR3M EURIBOR3M, b.CONTACT CONTACT,
    b.EMP_VAR_RATE EMP_VAR_RATE, b.DAY_OF_WEEK DAY_OF_WEEK
FROM pca_output p, bank b
```

```
WHERE p.id <> 10000 AND p.id=b.id
ORDER BY VECTOR_DISTANCE(embedding, (select embedding from pca_output
where id=10000), COSINE)
FETCH FIRST 3 ROWS ONLY;
```

Output:

```
        ID      PDAYS  EURIBOR3M   CONTACT EMP_VAR_RATE DAY_OF_WEEK
---------- ---------- ---------- --------- ------------ -----------
      9416        999      4.967 telephone          1.4         fri
     13485        999      4.963 telephone          1.4         thu
      9800        999      4.959 telephone          1.4         wed
```

The results of the previous query illustrate that the closest 3 records are very similar. In contrast, the distributions of these features across the data set are dispersed as shown in the following queries:

```
SELECT avg(PDAYS) avg, stddev(PDAYS) std, min(PDAYS) min, max(PDAYS) max
FROM bank;
```

Output:

```
       AVG        STD        MIN        MAX
---------- ---------- ---------- ----------
962.475454 186.910907          0        999
```

```
SELECT avg(EURIBOR3M) avg, stddev(EURIBOR3M) std, min(EURIBOR3M) min,
max(EURIBOR3M) max
FROM bank;
```

Output:

```
       AVG       STD        MIN        MAX
---------- --------- ---------- ----------

3.62129081 1.7344474       .634      5.045
```

This tutorial demonstrates how you can vectorize relational data very efficiently and achieve significant compression while maintaining a high quality similarity search.

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for more information about Feature Extraction algorithms

Perform Chunking With Embedding

In these examples, you can see how to explore the `VECTOR_CHUNKS` SQL function along with chainable utility PL/SQL functions to split large textual extracts and documents into chunks and then represent each chunk as a vector embedding.

To embed large textual data, you first need to prepare it in a format that can be processed by embedding models. You first transform the data into plain text, split the resulting text into smaller chunks of text, and then transform each chunk into a vector. This is done to comply with the input limits set by embedding models. Chunks can be words (to capture specific words or word pieces), sentences (to capture a specific context), or paragraphs (to capture broader themes).

- Convert Text to Chunks With Custom Chunking Specifications
  A chunked output, especially for long and complex documents, sometimes loses contextual meaning or coherence with its parent content. In this example, you can see how to refine your chunks by applying custom chunking specifications.
- Convert File to Text to Chunks to Embeddings Within Oracle Database
  First convert a PDF file to text, split the text into chunks, and then create vector embeddings on each chunk by accessing a vector embedding model stored in the database.
- Convert File to Embeddings Within Oracle Database
  Directly extract vector embeddings from a PDF document, using a single-step statement, by accessing a vector embedding model stored in the database.
- Generate and Use Embeddings for an End-to-End Search
  First generate vector embeddings from textual content by using a vector embedding model stored in the database, and then populate and query a vector index. At query time, you also vectorize the query criteria on the fly.

Convert Text to Chunks With Custom Chunking Specifications

A chunked output, especially for long and complex documents, sometimes loses contextual meaning or coherence with its parent content. In this example, you can see how to refine your chunks by applying custom chunking specifications.

Here, you use the `VECTOR_CHUNKS` SQL function or the `UTL_TO_CHUNKS()` PL/SQL function from the `DBMS_VECTOR_CHAIN` package.

> **Note:**
>
> This example shows how to apply some of the custom chunking parameters. To further explore all the supported chunking parameters with more detailed examples, see Explore Chunking Techniques and Examples.

1. Connect to Oracle Database as a local user.

a. Log in to SQL*Plus as the `SYS` user, connecting as `SYSDBA`:

```
conn sys/password as sysdba


CREATE TABLESPACE tbs1
DATAFILE 'tbs5.dbf' SIZE 20G AUTOEXTEND ON
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;


SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 10000
SET LONG 10000
```

b. Create a local test user (`docuser`) and grant necessary privileges:

```
DROP USER docuser cascade;


CREATE USER docuser identified by docuser DEFAULT TABLESPACE tbs1 quota
unlimited on tbs1;


GRANT DB_DEVELOPER_ROLE to docuser;
```

c. Connect as the local user (`docuser`):

```
CONN docuser/password
```

2. Create a relational table (`documentation_tab`) to store unstructured text chunks in it:

```
DROP TABLE IF EXISTS documentation_tab;

CREATE TABLE documentation_tab (
    id NUMBER,
    text VARCHAR2(2000));


INSERT INTO documentation_tab VALUES(1,
'Oracle AI Vector Search stores and indexes vector embeddings'||
' for fast retrieval and similarity search.'||CHR(10)||CHR(10)||
'    About Oracle AI Vector Search'||CHR(10)||
'    Vector Indexes are a new classification of specialized indexes'||
' that are designed for Artificial Intelligence (AI) workloads that
allow'||
' you to query data based on semantics, rather than keywords.'||CHR(10)||
CHR(10)||
'    Why Use Oracle AI Vector Search?'||CHR(10)||
' The biggest benefit of Oracle AI Vector Search is that semantic search'||
```

```
' on unstructured data can be combined with relational search on
business'||
' data in one single system.'||CHR(10));

COMMIT;


SET LINESIZE 1000;
SET PAGESIZE 200;
COLUMN doc FORMAT 999;
COLUMN id  FORMAT 999;
COLUMN pos FORMAT 999;
COLUMN siz FORMAT 999;
COLUMN txt FORMAT a60;
COLUMN data FORMAT a80;
```

3. Call the `VECTOR_CHUNKS` SQL function and specify the following custom chunking parameters. Setting the `NORMALIZE` parameter to `all` ensures that the formatting of the results is easier to read, which is especially helpful when the data includes PDF documents:

```
SELECT D.id doc, C.chunk_offset pos, C.chunk_length siz, C.chunk_text txt
FROM documentation_tab D, VECTOR_CHUNKS(D.text
    BY words
    MAX 50
    OVERLAP 0
    SPLIT BY recursively
    LANGUAGE american
    NORMALIZE all) C;
```

To call the same operation using the `UTL_TO_CHUNKS` function from the `DBMS_VECTOR_CHAIN` package, run:

```
SELECT D.id doc,
    JSON_VALUE(C.column_value, '$.chunk_id' RETURNING NUMBER) AS id,
    JSON_VALUE(C.column_value, '$.chunk_offset' RETURNING NUMBER) AS pos,
    JSON_VALUE(C.column_value, '$.chunk_length' RETURNING NUMBER) AS siz,
    JSON_VALUE(C.column_value, '$.chunk_data') AS txt
FROM documentation_tab D,
   dbms_vector_chain.utl_to_chunks(D.text,
   JSON('{"by":"words",
          "max":"50",
          "overlap":"0",
          "split":"recursively",
          "language":"american",
          "normalize":"all"}')) C;
```

This returns a set of three chunks, which are split by words recursively using blank lines, new lines, and spaces:

```
DOC  POS  SIZ  TXT
---- ---- ---- ------------------------------------------------------------
1    1    108   Oracle AI Vector Search stores and indexes vector embeddings
          for fast retrieval and similarity search.
```

```
1   109  234   About Oracle AI Vector Search
           Vector Indexes are a new classification of specialized index
           es that are designed for Artificial Intelligence (AI) worklo
           ads that allow you to query data based on semantics, rather
           than keywords.

1   343  204   Why Use Oracle AI Vector Search?
           The biggest benefit of Oracle AI Vector Search is that seman
           tic search on unstructured data can be combined with relatio
           nal search on business data in one single system.
```

The chunking results contain:

- `chunk_id` as `DOC`: ID for each chunk
- `chunk_offset` as `POS`: Original position of each chunk in the source document, relative to the start of document (which has a position of `1`)
- `chunk_length` as `SIZ`: Character length of each chunk
- `chunk_data` as `TXT`: Textual content from each chunk

Related Topics

- VECTOR_CHUNKS
  Use `VECTOR_CHUNKS` to split plain text into smaller chunks to generate vector embeddings that can be used with vector indexes or hybrid vector indexes.
- UTL_TO_CHUNKS
  Use the `DBMS_VECTOR_CHAIN.UTL_TO_CHUNKS` chainable utility function to split a large plain text document into smaller chunks of text.

Convert File to Text to Chunks to Embeddings Within Oracle Database

First convert a PDF file to text, split the text into chunks, and then create vector embeddings on each chunk by accessing a vector embedding model stored in the database.

You can run parallel or step-by-step transformations like this for standalone applications where you want to review, inspect, and accordingly amend results at each stage and then proceed further.

Here, you use a set of functions from the `DBMS_VECTOR_CHAIN` package, such as `UTL_TO_TEXT`, `UTL_TO_CHUNKS`, and `UTL_TO_EMBEDDINGS`.

To generate embeddings using a vector embedding model stored in the database, through step-by-step transformation chains:

1. **Connect as a local user and prepare your data dump directory**.
   a. Log in to SQL*Plus as the `SYS` user, connecting as `SYSDBA`:

```
conn sys/password as sysdba


CREATE TABLESPACE tbs1
DATAFILE 'tbs5.dbf' SIZE 20G AUTOEXTEND ON
```

```
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;


SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 10000
SET LONG 10000
```

b. Create a local user (`docuser`) and grant necessary privileges:

```
drop user docuser cascade;


create user docuser identified by docuser DEFAULT TABLESPACE tbs1 quota
unlimited on tbs1;


grant DB_DEVELOPER_ROLE to docuser;
```

c. Create a local directory (`VEC_DUMP`) to store your input data and model files. Grant necessary privileges:

```
create or replace directory VEC_DUMP as '/my_local_dir/';


grant read, write on directory VEC_DUMP to docuser;

commit;
```

d. Connect as the local user (`docuser`):

```
conn docuser/password
```

2. **Convert file to text**.

a. Create a relational table (`documentation_tab`) and store your PDF document (*Oracle Database Concepts*) in it:

```
drop table documentation_tab purge;


CREATE TABLE documentation_tab (id number, data blob);

INSERT INTO documentation_tab values(1, to_blob(bfilename('VEC_DUMP',
'database-concepts23ai.pdf')));

commit;


SELECT dbms_lob.getlength(t.data) from documentation_tab t;
```

b. Call `UTL_TO_TEXT` to convert the PDF document into text format:

```
SELECT dbms_vector_chain.utl_to_text(dt.data) from documentation_tab dt;
```

An excerpt from the output is as follows:

```
DBMS_VECTOR_CHAIN.UTL_TO_TEXT(DT.DATA)
--------------------------------------------------------------------------------
Database Concepts
23ai
Oracle Database Database Concepts, 23ai

This software and related documentation are provided under a license
agreement containing restrictions on
use and disclosure and are protected by intellectual property laws. Except
as expressly permitted in your
license agreement or allowed by law, you may not use, copy, reproduce,
translate
, broadcast, modify, license, transmit, distribute, exhibit, perform,
publish, or display any part, in any for
m, or by any means. Reverse engineering, disassembly, or decompilation of
this software, unless required by
law for interoperability, is prohibited.

Contents

Preface
Audience
xxiii
Documentation Accessibility
xxiii
Related Documentation
xxiv
Conventions
xxiv
1
Introduction to Oracle Database
About Relational Databases
1-1
Database Management System (DBMS)
1-2
Relational Model
1-2
Relational Database Management System (RDBMS)
1-3
Brief History of Oracle Database
1-3
Schema Objects
1-5
Tables
1-5
Indexes
1-6
Data Access
```

```
1 row selected.
```

3. **Convert text to chunks**.

   a. Call `UTL_TO_CHUNKS` to chunk the text document:

   Here, you create the chunks using the default chunking parameters.

```
SELECT ct.*
  from documentation_tab dt,

dbms_vector_chain.utl_to_chunks(dbms_vector_chain.utl_to_text(dt.data))
ct;
```

   An excerpt from the output is as follows:

```
{"chunk_id":1,"chunk_offset":1508024,"chunk_length":579,"chunk_data":"In
ventory
\n\n\n\nAnalysis \n\n\n\nReporting \n\n\n\nMining\n\n\n\nSummary
\n\n\n\nData
 \n\n\n\nRaw Data\n\n\n\nMetadata\n\n\n\nSee Also:\n\n\n\nOracle
Database Data
Warehousing Guide to learn about transformation
\n\n\n\nmechanisms\n\n\n\nOvervie
w of Extraction, Transformation, and Loading (ETL) \n\n\n\nThe process
of extrac
ting data from source systems and bringing it into the warehouse is
\n\n\n\ncomm
only called ETL: extraction, transformation, and loading. ETL refers to
a broad
process \n\n\n\nrather than three well-defined steps.\n\n\n\nIn a
typical scenar
io, data from one or more operational systems is extracted and then"}

{"chunk_id":2,"chunk_offset":1508603,"chunk_length":607,"chunk_data":"ph
ysica
lly transported to the target system or an intermediate system for
processing. \
n\n\n\nDepending on the method of transportation, some transformations
can occur
 during this \n\n\n\nprocess. For example, a SQL statement that
directly accesse
s a remote target through a \n\n\n\ngateway can concatenate two columns
as part
of the \n\n\n\nSELECT\n\n\n\nstatement. \n\n\n\nOracle Database is not
itself an
 ETL tool. However, Oracle Database provides a rich set of
\n\n\n\ncapabilities
usable by ETL tools and customized ETL solutions. ETL capabilities
provided by \
n\n\n\nOracle Database include:\n\n\n\n? \n\n\n\nTransportable
tablespaces"}

3728 rows selected.
```

Notice the extra spaces and newline characters (`\n\n`) in the chunked output. This is because normalization is not applied by default. As shown in the next step, you can apply custom chunking specifications, such as `normalize` to omit duplicate characters, extra spaces, or newline characters from the output. You can further refine your chunks by applying other chunking specifications, such as split conditions or maximum size limits.

b. Apply the following JSON parameters to use normalization and some of the custom chunking specifications (described in Explore Chunking Techniques and Examples):

```
SELECT ct.*
  from documentation_tab dt,
  dbms_vector_chain.utl_to_chunks(dbms_vector_chain.utl_to_text(
    dt.data),
    JSON('{
           "by" : "words",
           "max" : "100",
           "overlap" : "0",
           "split" : "recursively",
           "language" : "american",
           "normalize" : "all"
          }')) ct;
```

The output may now appear as:

```
{"chunk_id":2536,"chunk_offset":1372527,"chunk_length":633,"chunk_data":
"The dat
abase maps granules to parallel execution servers at execution time.
When a para
llel execution server finishes reading the rows corresponding to a
granule, and
when granules remain, it obtains another granule from the query
coordinator. This
operation continues until the table has been read. The execution
servers send
results back to the coordinator, which assembles the pieces into the
desired full
table scan. Oracle Database VLDB and Partitioning Guide to learn how to
use
parallel execution. Oracle Database Data Warehousing Guide to learn
about
recommended"}

{"chunk_id":2537,"chunk_offset":1373160,"chunk_length":701,"chunk_data":
"initial
ization parameters for parallelism\n\nChapter 18\n\nOverview of
Background Proce
sses\n\nApplication and Oracle Net Services Architecture\n\nThis
chapter defines
application architecture and describes how an Oracle database and
database appli
cations work in a distributed processing environment. This material
applies to
almost every type of Oracle Database environment. Overview of Oracle
Application
Architecture In the context of this chapter, application architecture
```

```
refers to
the computing environment in which a database application connects to
an Oracle
database."}

3728 rows selected.
```

The chunking results contain:

- `chunk_id`: Chunk ID for each chunk
- `chunk_offset`: Original position of each chunk in the source document, relative to the start of document (which has a position of `1`)
- `chunk_length`: Character length of each chunk
- `chunk_data`: Text pieces from each chunk

4. **Convert chunks to embeddings**.

   a. Load your embedding model into Oracle Database by calling the `load_onnx_model` procedure.

```
EXECUTE dbms_vector.drop_onnx_model(model_name => 'doc_model', force =>
true);


EXECUTE dbms_vector.load_onnx_model('VEC_DUMP',
'my_embedding_model.onnx', 'doc_model', JSON('{"function" :
"embedding", "embeddingOutput" : "embedding" , "input": {"input":
["DATA"]}}'));
```

   In this example, the procedure loads an ONNX model file, named `my_embedding_model.onnx` from the `VEC_DUMP` directory, into the database as `doc_model`. You must replace `my_embedding_model.onnx` with an ONNX export of your embedding model and `doc_model` with the name under which the imported model is stored in the database.

> **Note:**
>
> If you do not have an embedding model in ONNX format, then perform the steps listed in ONNX Pipeline Models : Text Embedding.

   b. Call `UTL_TO_EMBEDDINGS` to generate a set of vector embeddings corresponding to the chunks:

```
var embed_params clob;
exec :embed_params := '{"provider":"database", "model":"doc_model"}';

SELECT et.* from
  documentation_tab dt,
  dbms_vector_chain.utl_to_embeddings(

dbms_vector_chain.utl_to_chunks(dbms_vector_chain.utl_to_text(dt.data)),
    json(:embed_params)) et;
```

An excerpt from the output is as follows:


```
{"embed_id":"1","embed_data":"Introduction to Oracle Database\n\n\nThis
chapter provides an overview of Oracle Database. Every organization has
informat
ion that it must store and manage to meet its requirements. For example, a
corpo
ration must collect and maintain human resources records for its
employees. About
Relational Databases and Schema Objects\n\n\nData Access tables with
parent keys,
base, upgrade, UROWID data type, user global area (UGA), user program
interface,
","embed_vector":"[0.111119926,0.0423980951,-0.00929224491,-0.0352411047,-0
.0144
591287,0.0277361721,0.183199733,-0.0245029964,-0.137614027,0.0730137378,0.0
17934
6036,0.0788726509,0.0176453814,0.100403085,-0.0518687107,-0.0152645027,0.02
83792
187,-0.114087239,0.0139923804,0.0747490972,-0.181839675,-0.130034953,0.1012
07718
,0.117135495,-0.0682030097,-0.217743069,0.0613380745,0.0150767341,0.0361393
057,-
0.113082513,-0.0550440662,-0.000983044971,-0.00719357422,0.1590323,-0.02204
14512
,-0.0723528489,-0.0126240514,-0.175765082,0.168952227,0.0466451086,-0.12136
507,-
0.0442310236,0.0139067639,0.054659389,-0.29653421,-0.0988782048,0.079434923
8,-0.
0758788213,0.0152856084,-0.0260562375,0.0652966872,-0.0782724097,-0.0226081
386,0
.0909011662,-0.184569761,0.159565002,-0.15350005,-0.0108382348,0.101788878,
1.919
59683E-002,2.54665539E-002,2.50248201E-002,5.29858321E-002,1.42359538E-002,
5.655
82886E-002,3.41602638E-002,3.18607911E-002,-3.07250433E-002,-3.60006578E-00
2,-3.
26940455E-002,-5.13980947E-002,-9.18597169E-003,-2.40122043E-002,2.15246622
E-002
,-3.89301814E-002,1.09825116E-002,-8.59739035E-002,-3.34327705E-002,-6.5231
0252E
-002,2.46418975E-002,6.27725571E-003,6.54156879E-002,-2.97986511E-003,-1.48
5541E
-003,-9.00155635E-003]"}

{"embed_id":2,"embed_data":"1-18 Database, 19-6 write-ahead,18-17 Learn
session
memory in a large pool. The multitenant architecture enables an Oracle
database
to function as a multitenant container database (CDB). XStream,20-38\nZero
Data
Loss Recovery Appliance See Recovery Appliance zone maps, An application
contai
ner consists of exactly one application root, and PDBs plugged in to this
root.
Index-21\n D:20231208125114-08'00' D:20231208125114-08'00'
```

```
D:20231208125114-08'0
0'
19-6","embed_vector":"[6.30229115E-002,6.80943206E-002,-6.94655553E-002,-2.
58
157589E-002,-1.89648587E-002,-9.02655348E-002,1.97774544E-002,-9.39233322E-
003,-
5.06882742E-002,2.0078931E-002,-1.28898735E-003,-4.10606936E-002,2.09831214
E-003
,-4.53372523E-002,-7.09890276E-002,5.38906306E-002,-5.81014939E-002,-1.3959
175E-
004,-1.08725717E-002,-3.79145369E-002,-4.39973129E-003,3.25602405E-002,6.58
87302
2E-002,-4.27212603E-002,-3.00925318E-002,3.07144262E-002,-2.26370787E-004,-
4.623
15865E-002,1.11807801E-001,7.36674219E-002,-1.61244173E-003,7.35205635E-002
,4.16
726843E-002,-5.08309156E-002,-3.55720241E-003,4.49763797E-003,5.03803678E-0
02,2.
32542045E-002,-2.58533042E-002,9.06257033E-002,8.49585086E-002,8.65213498E-
002,5
.84013164E-002,-7.72946924E-002,6.65430725E-002,-1.64568517E-002,3.23978886
E-002
,2.24988302E-003,3.02566844E-003,-2.43405364E-002,9.75424573E-002,4.1463053
8E-00
3,1.89351812E-002,-1.10467218E-001,-1.24333188E-001,-2.36738548E-002,7.5427
7706E
-002,-1.64660662E-002,-1.38906585E-002,3.42438952E-003,-1.88432514E-005,-2.
47511
379E-002,-3.42802797E-003,3.23110656E-003,4.24311385E-002,6.59448802E-002,-
3.311
67318E-002,-5.14010936E-002,2.38897409E-002,-9.00154635E-002]"}

3728 rows selected.
```

The embedding results contain:

- `embed_id`: ID number of each vector embedding
- `embed_data`: Input text that is transformed into embeddings
- `embed_vector`: Generated vector representations

Related Topics

- DBMS_VECTOR
  The `DBMS_VECTOR` package simplifies common operations with Oracle AI Vector Search, such as extracting chunks or embeddings from user data, generating text for a given prompt or an image, creating a vector index, or reporting on index accuracy.
- DBMS_VECTOR_CHAIN
  The `DBMS_VECTOR_CHAIN` package enables advanced operations with Oracle AI Vector Search, such as chunking and embedding data along with text generation and summarization capabilities. It is more suitable for text processing with similarity search and hybrid search, using functionality that can be pipelined together for an end-to-end search.

Convert File to Embeddings Within Oracle Database

Directly extract vector embeddings from a PDF document, using a single-step statement, by accessing a vector embedding model stored in the database.

Here, you perform a file-to-text-to-chunks-to-embeddings transformation by calling a set of `DBMS_VECTOR_CHAIN.UTL` functions in a single `CREATE TABLE` statement.

This statement creates a relational table (`doc_chunks`) from unstructured text chunks and the corresponding vector embeddings:

```
CREATE TABLE doc_chunks as
(select dt.id doc_id, et.embed_id, et.embed_data, to_vector(et.embed_vector)
embed_vector
 from
   documentation_tab dt,
   dbms_vector_chain.utl_to_embeddings(

dbms_vector_chain.utl_to_chunks(dbms_vector_chain.utl_to_text(dt.data),
json('{"normalize":"all"}')),
       json('{"provider":"database", "model":"doc_model"}')) t,
   JSON_TABLE(t.column_value, '$[*]' COLUMNS (embed_id NUMBER PATH
'$.embed_id', embed_data VARCHAR2(4000) PATH '$.embed_data', embed_vector
CLOB PATH '$.embed_vector')) et
);
```

Note that each successive function depends on the output of the previous function, so the order of chains is important here. First, the output from `utl_to_text` (`dt.data` column) is passed as an input for `utl_to_chunks` and then the output from `utl_to_chunks` is passed as an input for `utl_to_embeddings`.

For complete example, run SQL Quick Start Using a Vector Embedding Model Uploaded into the Database, where you can see how to embed Oracle Database Documentation books in the `doc_chunks` table and perform similarity searches using vector indexes.

Generate and Use Embeddings for an End-to-End Search

First generate vector embeddings from textual content by using a vector embedding model stored in the database, and then populate and query a vector index. At query time, you also vectorize the query criteria on the fly.

This example covers the entire workflow of Oracle AI Vector Search (as explained in Understand the Stages of Data Transformations). If you are not yet familiar with the concepts beyond generating embeddings (such as creating and querying vector indexes), review the remaining sections before running this scenario.
To run an end-to-end similarity search workflow by accessing a vector embedding model stored in the database:

1. Start SQL*Plus and connect to Oracle Database as a local user:

a. Log in to SQL*Plus as the `SYS` user, connecting as `SYSDBA`:

```
conn sys/password as sysdba


CREATE TABLESPACE tbs1
DATAFILE 'tbs5.dbf' SIZE 20G AUTOEXTEND ON
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;


SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 10000
SET LONG 10000
```

b. Create a local user (`docuser`) and grant necessary privileges:

```
drop user docuser cascade;


create user docuser identified by docuser DEFAULT TABLESPACE tbs1 quota
unlimited on tbs1;


grant DB_DEVELOPER_ROLE to docuser;
```

c. Create a local directory (`VEC_DUMP`) to store your input data and model files. Grant necessary privileges:

```
create or replace directory VEC_DUMP as '/my_local_dir/';


grant read, write on directory VEC_DUMP to docuser;

commit;
```

d. Connect as the local user (`docuser`):

```
conn docuser/password
```

2. Create a relational table (`documentation_tab`) and store your textual content in it.

```
drop table documentation_tab purge;


create table documentation_tab (id number, text clob);


insert into documentation_tab values (1,
    'Analytics empowers business analysts and consumers with modern, AI-
```

```
powered, self-service analytics capabilities for data preparation,
visualization, enterprise reporting, augmented analysis, and natural
language processing.
     Oracle Analytics Cloud is a scalable and secure public cloud service
that provides capabilities to explore and perform collaborative analytics
for you, your workgroup, and your enterprise.

     Oracle Analytics Cloud is available on Oracle Cloud Infrastructure
Gen 2 in several regions in North America, EMEA, APAC, and LAD when you
subscribe through Universal Credits. You can subscribe to Professional
Edition or Enterprise Edition.');

insert into documentation_tab values (3,
    'Generative AI Data Science is a fully managed and serverless platform
for data science teams to build, train, and manage machine learning models
in the Oracle Cloud Infrastructure.');

insert into documentation_tab values (4,
    'Language allows you to perform sophisticated text analysis at scale.
Using the pretrained and custom models, you can process unstructured text
to extract insights without data science expertise.
     Pretrained models include sentiment analysis, key phrase extraction,
text classification, and named entity recognition. You can also train
custom models for named entity recognition and text
     classification with domain specific datasets. Additionally, you can
translate text across numerous languages.');

insert into documentation_tab values (5,
    'When you work with Oracle Cloud Infrastructure, one of the first
steps is to set up a virtual cloud network (VCN) for your cloud resources.
This topic gives you an overview of Oracle Cloud
     Infrastructure Networking components and typical scenarios for using
a VCN. A virtual, private network that you set up in Oracle data centers.
It closely resembles a traditional network, with
     firewall rules and specific types of communication gateways that you
can choose to use. A VCN resides in a single Oracle Cloud Infrastructure
region and covers one or more CIDR blocks
     (IPv4 and IPv6, if enabled). See Allowed VCN Size and Address Ranges.
The terms virtual cloud network, VCN, and cloud network are used
interchangeably in this documentation.
     For more information, see VCNs and Subnets.');

insert into documentation_tab values (6,
    'NetSuite banking offers several processing options to accurately
track your income. You can record deposits to your bank accounts to
capture customer payments and other monies received in the
     course of doing business. For a deposit, you can select payments
received for existing transactions, add funds not related to transaction
payments, and record any cash received back from the bank.');

commit;
```

3. Load your embedding model by calling the `load_onnx_model` procedure.

```
EXECUTE dbms_vector.drop_onnx_model(model_name => 'doc_model', force =>
true);


EXECUTE dbms_vector.load_onnx_model(
   'VEC_DUMP',
   'my_embedding_model.onnx',
   'doc_model',
   json('{"function" : "embedding", "embeddingOutput" : "embedding" ,
"input": {"input": ["DATA"]}}')
);
```

In this example, the procedure loads an ONNX model file, named `my_embedding_model.onnx` from the `VEC_DUMP` directory, into the database as `doc_model`. You must replace `my_embedding_model.onnx` with an ONNX export of your embedding model and `doc_model` with the name under which the imported model is stored in the database.

> **Note:**
>
> If you do not have an embedding model in ONNX format, then perform the steps listed in ONNX Pipeline Models : Text Embedding.

4. Create a relational table (`doc_chunks`) to store unstructured data chunks and associated vector embeddings, by using `doc_model`.

```
create table doc_chunks as (
  SELECT d.id id,
         row_number() over (partition by d.id order by d.id) chunk_id,
         vc.chunk_offset chunk_offset,
         vc.chunk_length chunk_length,
         vc.chunk_text chunk,
         vector_embedding(doc_model using vc.chunk_text as data) vector
  FROM documentation_tab d,
       vector_chunks(d.text by words max 100 overlap 10 split RECURSIVELY)
vc
);
```

The `CREATE TABLE` statement reads the text from the `DOCUMENTATION_TAB` table, first applies the `VECTOR_CHUNKS` SQL function to split the text into chunks based on the specified chunking parameters, and then applies the `VECTOR_EMBEDDING` SQL function to generate corresponding vector embedding on each resulting chunk text.

5. Explore the `doc_chunks` table by selecting rows from it to see the chunked output.

```
desc doc_chunks;
set linesize 100
set long 1000
col id for 999
col chunk_id for 99999
col chunk_offset for 99999
```

```
col chunk_length for 99999
col chunk for a30
col vector for a100


select id, chunk_id, chunk_offset, chunk_length, chunk from doc_chunks;
```

The chunking output returns a set of seven chunks, which are split recursively, that is, using the `BLANKLINE`, `NEWLINE`, `SPACE`, `NONE` sequence. Note that Document 5 produces two chunks when the maximum word limit of `100` is reached.

You can see that the first chunk ends at a blank line. The text from the first chunk overlaps onto the second chunk, that is, 10 words (including comma and period; underlined below) are overlapping. Similarly, there is an overlap of 10 (also underlined below) between the fifth and sixth chunks.

```
  ID CHUNK_ID CHUNK_OFFSET CHUNK_LENGTH CHUNK
---- -------- ------------ ------------ ------------------------------
   1        1            1          418  Analytics empowers business an
                                        alysts and consumers with mode
                                        rn, AI-powered, self-service a
                                        nalytics capabilities for data
                                        preparation, visualization, e
                                        nterprise reporting, augmented
                                        analysis, and natural languag
                                        e processing.
                                        Oracle Analytics Cloud is
                                        a scalable and secure public
                                        cloud service that provides ca
                                        pabilities to explore and perf
                                        orm collaborative analytics for
                                        you, your workgroup, and your
                                        enterprise.

   1        2          373          291 for you, your workgroup, and
                                        your enterprise.
                                        Oracle Analytics Cloud is
                                        available on Oracle Cloud Inf
                                        rastructure Gen 2 in several r
                                        egions in North America, EMEA,
                                        APAC, and LAD when you subscr
                                        ibe through Universal Credits.
                                        You can subscribe to Professi
                                        onal Edition or Enterprise Edi
                                        tion.

   3        1            1          180  Generative AI Data Science is
                                          a fully managed and serverless
                                          platform for data science tea
                                          ms to build, train, and manage
                                          machine learning models in th
                                          e Oracle Cloud Infrastructure.

   4        1            1          505  Language allows you to perform
                                          sophisticated text analysis a
```

```
                                    t scale. Using the pretrained
                                    and custom models, you can pro
                                    cess unstructured text to extr
                                    act insights without data scie
                                    nce expertise.
                                    Pretrained models include
                                    sentiment analysis, key phras
                                    e extraction, text classificat
                                    ion, and named entity recognit
                                    ion. You can also train custom
                                    models for named entity recog
                                    nition and text
                                    classification with domai
                                    n specific datasets. Additiona
                                    lly, you can translate text ac
                                    ross numerous languages.

5     1        1     386  When you work with Oracle Clou
                          d Infrastructure, one of the f
                          irst steps is to set up a virt
                          ual cloud network (VCN) for yo
                          ur cloud resources. This topic
                          gives you an overview of Orac
                          le Cloud
                          Infrastructure Networking
                          components and typical scenar
                          ios for using a VCN. A virtual
                          , private network that you set
                          up in Oracle data centers. It
                          closely resembles a tradition
                          al network, with

5     2         329     474  centers. It closely resembles
                          a traditional network, with
                          firewall rules and specif
                          ic types of communication gate
                          ways that you can choose to us
                          e. A VCN resides in a single O
                          racle Cloud Infrastructure reg
                          ion and covers one or more CID
                          R blocks
                          (IPv4 and IPv6, if enable
                          d). See Allowed VCN Size and A
                          ddress Ranges. The terms virtu
                          al cloud network, VCN, and clo
                          ud network are used interchang
                          eably in this documentation.
                          For more information, see
                          VCNs and Subnets.

6     1        1     393  NetSuite banking offers severa
                          l processing options to accura
                          tely track your income. You ca
                          n record deposits to your bank
                          accounts to capture customer
                          payments and other monies rece
```

```
                                ived in the
                                course of doing business.
                                For a deposit, you can select
                                payments received for existin
                                g transactions, add funds not
                                related to transaction payment
                                s, and record any cash receive
                                d back from the bank.

7 rows selected.
```

6. Explore the first vector result by selecting rows from the `doc_chunks` table to see the embedding output.

```
select vector from doc_chunks where rownum <= 1;
```

An excerpt from the output is as follows:

```
[1.18813422E-002,2.53968383E-003,-5.33896387E-002,1.46877998E-003,5.7720981
5E-002,-1.58939194E-002,3
.12595293E-002,-1.13087103E-001,8.5138239E-002,1.10731693E-002,3.70671228E-
002,4.03710492E-002,1.503
95066E-001,3.31836529E-002,-1.98343433E-002,6.16453104E-002,4.2827677E-002,
-4.02921103E-002,-7.84291
551E-002,-4.79201972E-002,-5.06678E-002,-1.36317732E-002,-3.7761624E-003,-2
.3332756E-002,1.42400926E
-002,-1.11553416E-001,-3.70503664E-002,-2.60722954E-002,-1.2074843E-002,-3.
55089158E-002,-1.03518805
E-002,-7.05051869E-002,5.63110895E-002,4.79055084E-002,-1.46315445E-003,8.8
3129537E-002,5.12795225E-
002,7.5858552E-003,-4.13030013E-002,-5.2099824E-002,5.75958602E-002,3.72097
567E-002,6.11167103E-002,
,-1.23207876E-003,-5.46219759E-003,3.04734893E-002,1.80617068E-002,-2.85708
476E-002,-1.01670986E-002
,6.49402961E-002,-9.76506807E-003,6.15146831E-002,5.27246818E-002,7.4499443
2E-002,-5.86469211E-002,8
.84285953E-004,2.77456306E-002,1.99283361E-002,2.37570312E-002,2.33389344E-
002,-4.07911092E-002,-7.6
1070028E-002,1.23929314E-001,6.65794984E-002,-6.15389943E-002,2.62510721E-0
02,-2.48490628E-002]
```

7. Create an index on top of the `doc_chunks` table's `vector` column.

```
create vector index vidx on doc_chunks (vector)
   organization neighbor partitions
   with target accuracy 95
   distance EUCLIDEAN parameters (
    type IVF,
    neighbor partitions 2);
```

8. Run queries using the vector index.

- Query about Machine Learning:

```
select id, vector_distance(
    vector,
    vector_embedding(doc_model using 'machine learning models' as data),
    EUCLIDEAN) results
FROM doc_chunks order by results;
```

```
  ID    RESULTS
---- ----------
   3 1.074E+000
   4 1.086E+000
   5 1.212E+000
   5 1.296E+000
   1 1.304E+000
   6 1.309E+000
   1 1.365E+000

7 rows selected.
```

- Query about Generative AI:

```
select id, vector_distance(
    vector,
    vector_embedding(doc_model using 'gen ai' as data),
    EUCLIDEAN) results
FROM doc_chunks order by results;
```

```
  ID    RESULTS
---- ----------
   4 1.271E+000
   3 1.297E+000
   1 1.309E+000
   5  1.32E+000
   1 1.352E+000
   5 1.388E+000
   6 1.424E+000

7 rows selected.
```

- Query about Networks:

```
select id, vector_distance(
  vector,
  vector_embedding(doc_model using 'computing networks' as data),
  MANHATTAN) results
FROM doc_chunks order by results;
```

```
  ID    RESULTS
---- ----------
   5 1.387E+001
   5 1.441E+001
```

```
   3 1.636E+001
   1 1.707E+001
   4 1.758E+001
   1 1.795E+001
   6 1.902E+001

7 rows selected.
```

- Query about Banking:

```
select id, vector_distance(
    vector,
    vector_embedding(doc_model using 'banking, money' as data),
    MANHATTAN) results
FROM doc_chunks order by results;


  ID    RESULTS
---- ----------
   6 1.363E+001
   1 1.969E+001
   5 1.978E+001
   5 1.997E+001
   3 1.999E+001
   1 2.058E+001
   4 2.079E+001

7 rows selected.
```

Related Topics

- DBMS_VECTOR_CHAIN
  The `DBMS_VECTOR_CHAIN` package enables advanced operations with Oracle AI Vector Search, such as chunking and embedding data along with text generation and summarization capabilities. It is more suitable for text processing with similarity search and hybrid search, using functionality that can be pipelined together for an end-to-end search.

Configure Chunking Parameters

Oracle AI Vector Search provides many parameters for chunking text data, such as `SPLIT [BY]`, `OVERLAP`, or `NORMALIZE`. In these examples, you can see how to configure these parameters to define your own chunking specifications and strategies, so that you can create meaningful chunks.

- Explore Chunking Techniques and Examples
  Review these examples of all the supported chunking parameters. These examples can provide an idea on what are good and bad chunking techniques, and thus help you define a strategy when chunking your data.
- Create and Use Custom Vocabulary
  Create and use your own vocabulary of tokens when chunking data.
- Create and Use Custom Language Data
  Create and use your own language-specific conditions (such as common abbreviations) when chunking data.

Explore Chunking Techniques and Examples

Review these examples of all the supported chunking parameters. These examples can provide an idea on what are good and bad chunking techniques, and thus help you define a strategy when chunking your data.

Here, you can see how the following sample text of five lines is split when you apply various chunking parameters to it:

1[15]Oracle AI Vector Search stores and indexes vector embeddings for fast retrieval and similarity search.
[blank line 1]
2[05] About Oracle AI Vector Search
3[33] Vector Indexes are a new classification of specialized indexes
that are designed for Artificial Intelligence (AI) workloads that allow you
to query data based on semantics, rather than keywords.
[blank line 2]
4[07] Why Use Oracle AI Vector Search?
5[29]The biggest benefit of Oracle AI Vector Search is that semantic search on unstructured data
can be combined with relational search on business data in one single system.

Note the following:

- The lines are numbered as a reference for the explanations and include the word count in square brackets (for example, `1[15]`). The blank lines are also noted.
- The start and end boundaries of chunks are represented with colored markers.
- To perform examples with the `BY VOCABULARY` mode, you must create custom vocabulary beforehand (for example, `DOC_VOCAB`). See Create and Use Custom Vocabulary.

Example 4-1 BY chars MAX 200 OVERLAP 0 SPLIT BY none

This example shows the simplest form of chunking, where you split the text by a fixed number of characters (including the end-of-line characters), at whatever point that occurs in the text.

The text from the first chunk is split at an absolute maximum character of `200`, which divides the word `indexes` between the first two chunks. Similarly, you can see the word `Oracle` splitting between second and third chunks.

Oracle AI Vector Search stores and indexes vector embeddings for fast retrieval and similarity search.
About Oracle AI Vector Search
Vector Indexes are a new classification of specialized indexes
that are designed for Artificial Intelligence (AI) workloads that allow you
to query data based on semantics, rather than keywords.
Why Use Oracle AI Vector Search?
The biggest benefit of Oracle AI Vector Search is that semantic search on unstructured data
can be combined with relational search on business data in one single system.

Syntax:

```
SELECT C.*
FROM documentation_tab D, VECTOR_CHUNKS(D.text BY chars MAX 200 OVERLAP 0
                          SPLIT BY none LANGUAGE american NORMALIZE none) C;
```

Output:

```
CHUNK_OFFSET CHUNK_LENGTH CHUNK_TEXT
--------------------------------------------------------------------------------
1         200          Oracle AI Vector Search stores and indexes vector
embeddings for fast retrieval and similarity search.

                          About Oracle AI Vector Search
                          Vector Indexes are a new classification of
specialized ind

201         200          exes that are designed for Artificial Intelligence
(AI) workloads that allow you to query data based on semantics, rather than
keywords.

                          Why Use Oracle AI Vector Search?
                          The biggest benefit of O

401         146          racle AI Vector Search is that semantic search on
unstructured data can be combined with relational search on business data in
one single system.
```

Example 4-2 BY chars MAX 200 OVERLAP 0 SPLIT BY newline

In this example, the text is split into four chunks at new lines, if possible, within the maximum limit of `200` characters.

The text from the first chunk is split after the second line because the third line would exceed the maximum. The third line fits within the maximum perfectly. The fourth and fifth line would also exceed the maximum, so it produces two chunks.

Oracle AI Vector Search stores and indexes vector embeddings for fast retrieval and similarity search.
About Oracle AI Vector Search
Vector Indexes are a new classification of specialized indexes
that are designed for Artificial Intelligence (AI) workloads that allow you
to query data based on semantics, rather than keywords.
Why Use Oracle AI Vector Search?
The biggest benefit of Oracle AI Vector Search is that semantic search
on unstructured data can be combined with relational search on business
data in one single system.

Syntax:

```
SELECT C.*
FROM documentation_tab D, VECTOR_CHUNKS(D.text BY chars MAX 200 OVERLAP 0
```

```
                                SPLIT BY newline LANGUAGE american NORMALIZE none)
C;
```

Output:

```
CHUNK_OFFSET CHUNK_LENGTH CHUNK_TEXT
--------------------------------------------------------------------------------
--------------------
1            138          Oracle AI Vector Search stores and indexes vector
embeddings for fast retrieval and similarity search.

                             About Oracle AI Vector Search

143          196          Vector Indexes are a new classification of
specialized indexes that are designed for Artificial Intelligence (AI)
workloads that allow you to query data based on semantics, rather than
keywords.

343          33           Why Use Oracle AI Vector Search?

377          170          The biggest benefit of Oracle AI Vector Search is
that semantic search on unstructured data can be combined with relational
search on business data in one single system.
```

Example 4-3 BY chars MAX 200 OVERLAP 0 SPLIT BY recursively

In this example, the text is split into five chunks recursively using blank lines, newlines and then spaces, if possible, within the maximum of `200` characters.

The first chunk is split after the first blank line because including the text after the second blank line would exceed the maximum. The second passage exceeds the maximum on its own, so it is broken into two chunks at the new lines. Similarly, the third section is also broken into two chunks at the new lines.

Oracle AI Vector Search stores and indexes vector embeddings for fast retrieval and similarity search.

About Oracle AI Vector Search

Vector Indexes are a new classification of specialized indexes that are designed for Artificial Intelligence (AI) workloads that allow you to query data based on semantics, rather than keywords.

Why Use Oracle AI Vector Search?

The biggest benefit of Oracle AI Vector Search is that semantic search on unstructured data can be combined with relational search on business data in one single system.

Syntax:

```
SELECT C.*
FROM documentation_tab D, VECTOR_CHUNKS(D.text BY chars MAX 200 OVERLAP 0
                              SPLIT BY recursively LANGUAGE american NORMALIZE
none) C;
```

Output:

```
CHUNK_OFFSET CHUNK_LENGTH CHUNK_TEXT
--------------------------------------------------------------------------------
--------------------
1            104          Oracle AI Vector Search stores and indexes vector
embeddings for fast retrieval and similarity search.

109          30           About Oracle AI Vector Search

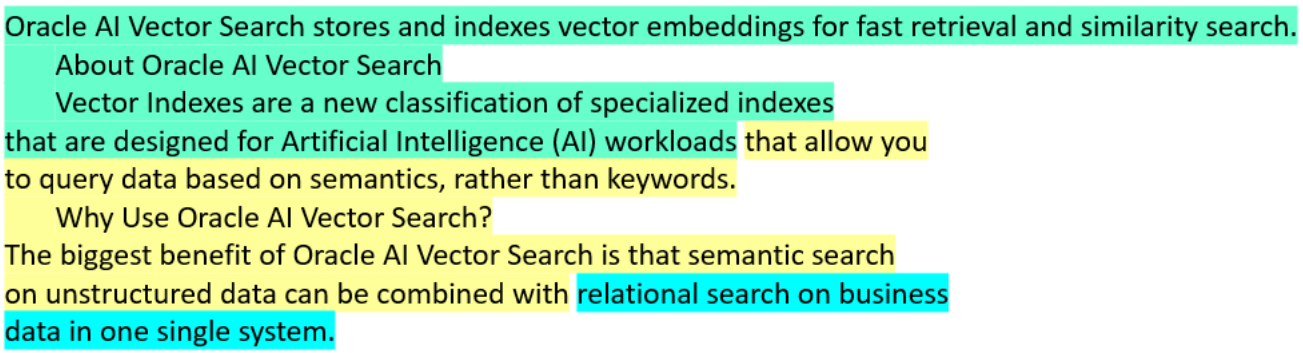
143          196          Vector Indexes are a new classification of
specialized indexes that are designed for Artificial Intelligence (AI)
workloads that allow you to query data based on semantics, rather than
keywords.

343          33           Why Use Oracle AI Vector Search?

377          170          The biggest benefit of Oracle AI Vector Search is
that semantic search on unstructured data can be combined with relational
search on business data in one single system.
```

Example 4-4 BY words MAX 40 OVERLAP 0 SPLIT BY none

In this example, the text is split into three chunks at an absolute maximum word of `40`, the third line after `wordloads` and the fifth line after `with`.



Syntax:

```
SELECT C.*
FROM documentation_tab D, VECTOR_CHUNKS(D.text BY words MAX 40 OVERLAP 0
                           SPLIT BY none LANGUAGE american NORMALIZE none) C;
```

Output:

```
CHUNK_OFFSET CHUNK_LENGTH CHUNK_TEXT
--------------------------------------------------------------------------------
--------------------
1         266          Oracle AI Vector Search stores and indexes vector
embeddings for fast retrieval and similarity search.

                           About Oracle AI Vector Search
```

```
                          Vector Indexes are a new classification of
specialized indexes that are designed for Artificial
                          Intelligence (AI) workloads

267         223          that allow you to query data based on semantics,
rather than keywords.

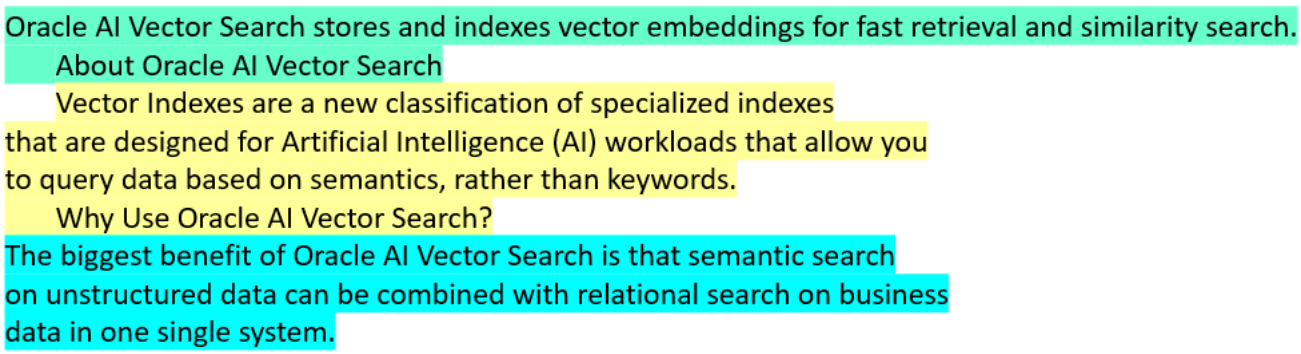
                          Why Use Oracle AI Vector Search?
                          The biggest benefit of Oracle AI Vector Search is
that semantic search on unstructured data can be combined with


490         57           relational search on business data in one single
system.
```

Example 4-5 BY words MAX 40 OVERLAP 0 SPLIT BY newline

In this example, the text is split into chunks at new lines, if possible, within the maximum of 40 words.

The first chunk (of 21 words) is split after the second line, as the third line would exceed the maximum number of words (21+33 words). The third and fourth lines fit within the maximum. The fifth line is 29 words, so fits in the last chunk.



Syntax:

```
SELECT C.*
FROM documentation_tab D, VECTOR_CHUNKS(D.text BY words MAX 40 OVERLAP 0
                           SPLIT BY newline LANGUAGE american NORMALIZE none)
C;
```

Output:

```
CHUNK_OFFSET CHUNK_LENGTH CHUNK_TEXT
--------------------------------------------------------------------------------
--------------------
1            138          Oracle AI Vector Search stores and indexes vector
embeddings for fast retrieval and similarity search.

                          About Oracle AI Vector Search
```

```
143         233          Vector Indexes are a new classification of
specialized indexes that are designed for Artificial Intelligence (AI)
workloads that allow you to query data based on semantics, rather than
keywords.

                          Why Use Oracle AI Vector Search?

377          170          The biggest benefit of Oracle AI Vector Search is
that semantic search on unstructured data can be combined with relational
search on business data in one single system.
```

Example 4-6 BY words MAX 40 OVERLAP 0 SPLIT BY recursively

In this example, the chunks are split by words recursively using blank lines, newlines, and spaces.

The text after the second blank line exceeds the maximum words, so the first chunk ends at the first blank line. The second chunk (of 38 words) ends at the next blank line. The final chunk (of 35 words) consists of the rest of the input.

Oracle AI Vector Search stores and indexes vector embeddings for fast retrieval and similarity search.

About Oracle AI Vector Search

Vector Indexes are a new classification of specialized indexes that are designed for Artificial Intelligence (AI) workloads that allow you to query data based on semantics, rather than keywords.

Why Use Oracle AI Vector Search?

The biggest benefit of Oracle AI Vector Search is that semantic search on unstructured data can be combined with relational search on business data in one single system.

Syntax:

```
SELECT C.*
FROM documentation_tab D, VECTOR_CHUNKS(D.text BY words MAX 40 OVERLAP 0
                          SPLIT BY recursively LANGUAGE american NORMALIZE
none) C;
```

Output:

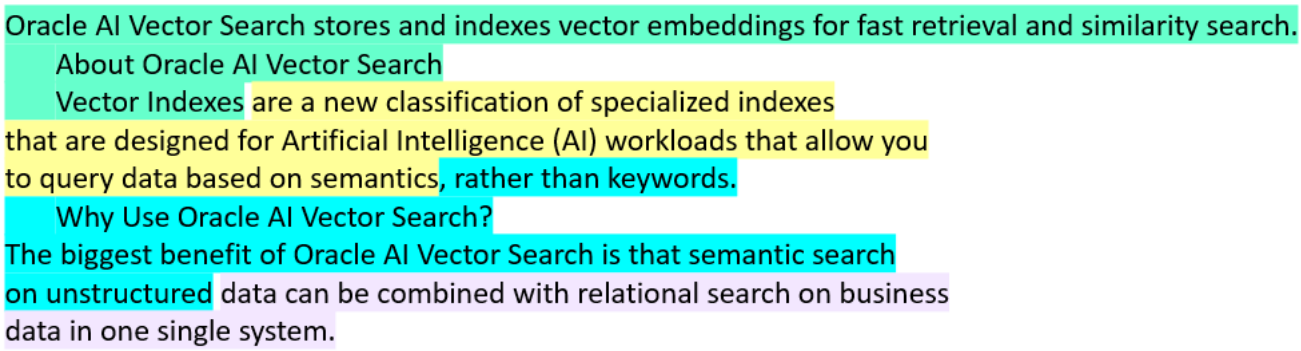
```
CHUNK_OFFSET CHUNK_LENGTH CHUNK_TEXT
--------------------------------------------------------------------------
--------------------
1         104          Oracle AI Vector Search stores and indexes vector
embeddings for fast retrieval and similarity search.

109         230          About Oracle AI Vector Search
                          Vector Indexes are a new classification of
specialized indexes that are designed for Artificial
                          Intelligence (AI) workloads that allow you to query
data based on semantics, rather than keywords.
```

```
343         204          Why Use Oracle AI Vector Search?
                          The biggest benefit of Oracle AI Vector Search is
that semantic search on unstructured data can be
                          combined with relational search on business data in
one single system.
```

Example 4-7 BY vocabulary MAX 40 OVERLAP 0 SPLIT BY none

In this example, the text is split into four chunks at an absolute maximum vocabulary token of 40, which contrasts with the three chunks produced in Example 4-4. This is because vocabulary tokens include pieces of words, so the chunk text is generally smaller than simple word splitting.



Syntax:

```
SELECT C.*
FROM documentation_tab D, VECTOR_CHUNKS(D.text BY vocabulary doc_vocab MAX 40
OVERLAP 0
                          SPLIT BY none LANGUAGE american NORMALIZE none) C;
```

Output:

```
CHUNK_OFFSET CHUNK_LENGTH CHUNK_TEXT
--------------------------------------------------------------------------------
--------------------
1         157          Oracle AI Vector Search stores and indexes vector
embeddings for fast retrieval and similarity search.

                          About Oracle AI Vector Search
                          Vector Indexes
158         156          are a new classification of specialized indexes that
are designed for Artificial Intelligence (AI) workloads that allow you to
query data based on semantics

314          150          , rather than keywords.

                          Why Use Oracle AI Vector Search?
                          The biggest benefit of Oracle AI Vector Search is
that semantic search on unstructured
```

```
464         83          data can be combined with relational search on
business data in one single system.
```

Example 4-8 BY vocabulary MAX 40 OVERLAP 0 SPLIT BY newline

In this example, the text is split into five chunks with newlines, using an absolute maximum vocabulary token of 40, which contrasts with Example 4-5.

Vocabulary tokens include pieces of words, so the chunk text is generally smaller than simple word splitting. This example produces five chunks rather than three in Example 4-5, with the middle passage split into two, and the final word unable to fit into the fourth chunk.

Oracle AI Vector Search stores and indexes vector embeddings for fast retrieval and similarity search.
About Oracle AI Vector Search
Vector Indexes are a new classification of specialized indexes
that are designed for Artificial Intelligence (AI) workloads that allow you
to query data based on semantics, rather than keywords.
Why Use Oracle AI Vector Search?
The biggest benefit of Oracle AI Vector Search is that semantic search
on unstructured data can be combined with relational search on business
data in one single system.

Syntax:

```
SELECT C.*
FROM documentation_tab D, VECTOR_CHUNKS(D.text BY vocabulary doc_vocab MAX 40
OVERLAP 0
                          SPLIT BY newline LANGUAGE american NORMALIZE none)
C;
```

Output:

```
CHUNK_OFFSET CHUNK_LENGTH CHUNK_TEXT
--------------------------------------------------------------------------------
--------------------
1         138          Oracle AI Vector Search stores and indexes vector
embeddings for fast retrieval and similarity search.

                         About Oracle AI Vector Search

143         148          Vector Indexes are a new classification of
specialized indexes that are designed for Artificial Intelligence (AI)
workloads that allow you to query

291         85           data based on semantics, rather than keywords.

                          Why Use Oracle AI Vector Search?

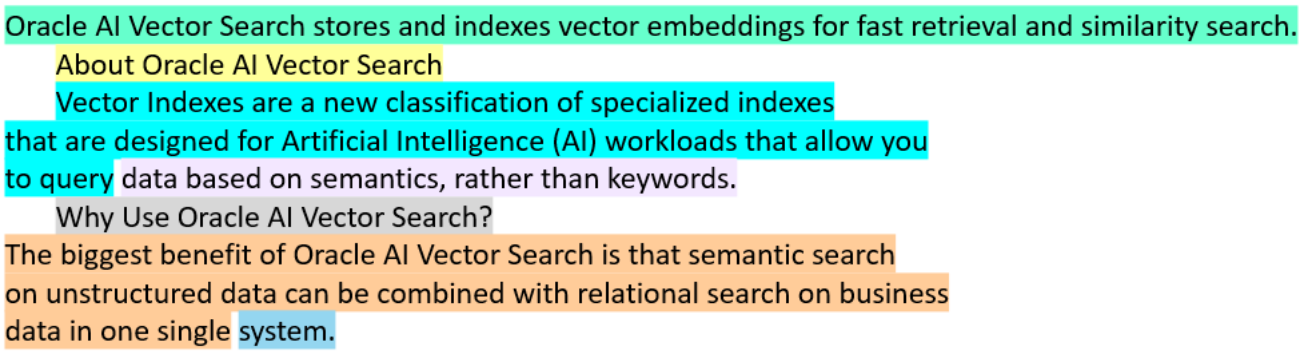
377         162          The biggest benefit of Oracle AI Vector Search is
that semantic search on unstructured data can be combined with relational
search on business data in one single
```

```
539         8           system.
```

Example 4-9 BY vocabulary MAX 40 OVERLAP 0 SPLIT BY recursively

In this example, the text is split into seven chunks recursively using blank lines, new lines, and spaces and an absolute maximum vocabulary token of 40, which contrasts with the three chunks produced in Example 4-6.

Vocabulary tokens include pieces of words, so the chunk text is generally smaller than simple word splitting. This example produces seven chunks with the middle passage split into three and the final passage split into three.



Syntax:

```
SELECT C.*
FROM documentation_tab D, VECTOR_CHUNKS(D.text BY vocabulary doc_vocab MAX 40
OVERLAP 0
                          SPLIT BY recursively LANGUAGE american NORMALIZE
none) C;
```

Output:

```
CHUNK_OFFSET CHUNK_LENGTH CHUNK_TEXT
--------------------------------------------------------------------------------
--------------------
1         104          Oracle AI Vector Search stores and indexes vector
embeddings for fast retrieval and similarity search.

109         30           About Oracle AI Vector Search

143         148          Vector Indexes are a new classification of
specialized indexes that are designed for Artificial Intelligence (AI)
workloads that allow you to query

291         48           data based on semantics, rather than keywords.

343         33           Why Use Oracle AI Vector Search?

377         162          The biggest benefit of Oracle AI Vector Search is
that semantic search on unstructured data can be combined with relational
```

```
search on business data in one single

539         8            system.
```

Example 4-10 BY words MAX 40 OVERLAP 5 SPLIT BY none

This example is similar to Example 4-4, except that an overlap of `5` is used here.

The first chunk ends at the maximum 40 words (after `workloads`). The second chunk overlaps with the last 5 words including parentheses of the first chunk, and ends after `unstructured`. The overlapping words are underlined below. The third chunk overlaps with the last 5 words, which are also underlined.

Oracle AI Vector Search stores and indexes vector embeddings for fast retrieval and similarity search.
About Oracle AI Vector Search
Vector Indexes are a new classification of specialized indexes
that are designed for Artificial **Intelligence (AI) workloads** that allow you
to query data based on semantics, rather than keywords.
Why Use Oracle AI Vector Search?
The biggest benefit of Oracle AI Vector Search is **that semantic search**
on unstructured data can be combined with relational search on business
data in one single system.

Syntax:

```
SELECT C.*
FROM documentation_tab D, VECTOR_CHUNKS(D.text BY words MAX 40 OVERLAP 5
                          SPLIT BY none LANGUAGE american NORMALIZE none) C;
```

Output:

```
CHUNK_OFFSET CHUNK_LENGTH CHUNK_TEXT
--------------------------------------------------------------------------------
------------------
1         266         Oracle AI Vector Search stores and indexes vector
embeddings for fast retrieval and similarity search.

                         About Oracle AI Vector Search
                         Vector Indexes are a new classification of
specialized indexes that are designed for Artificial Intelligence (AI)
workloads

239         225         Intelligence (AI) workloads that allow you to query
data based on semantics, rather than keywords.

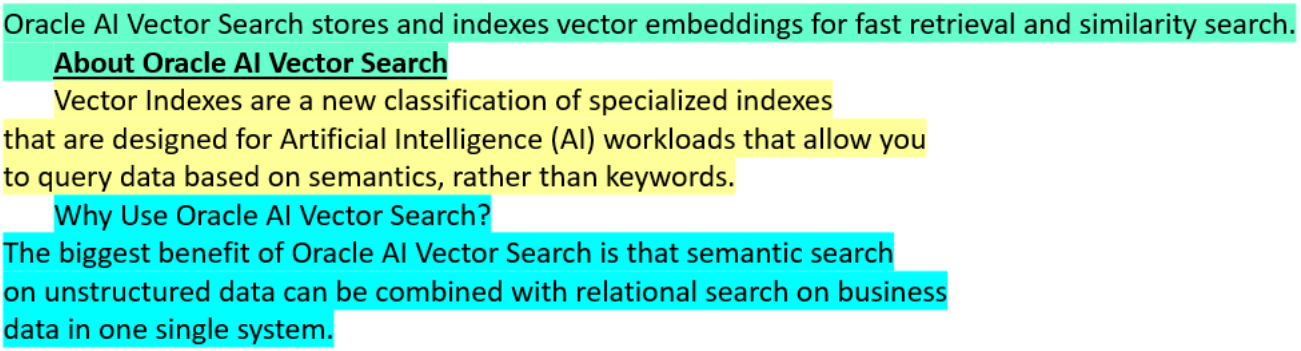
                         Why Use Oracle AI Vector Search?
                         The biggest benefit of Oracle AI Vector Search is
that semantic search on unstructured

427         120         that semantic search on unstructured data can be
combined with relational search on business data in one single system.
```

Example 4-11 BY words MAX 40 OVERLAP 5 SPLIT BY newline

This example is similar to Example 4-5, except that an overlap of 5 is used here. The overlapping portion of a chunk must obey the same split condition, in this case must begin on a new line.

The first chunk ends at the second line, as the third line would exceed the maximum 40 words. The second chunk starts with the second line of 5 words of the first chunk (underlined below) and ends at the third line. The third chunk has no overlap because the preceding line exceeds the maximum of 5.



Syntax:

```
SELECT C.*
FROM documentation_tab D, VECTOR_CHUNKS(D.text BY words MAX 40 OVERLAP 5
                           SPLIT BY newline LANGUAGE american NORMALIZE none)
C;
```

Output:

```
CHUNK_OFFSET CHUNK_LENGTH CHUNK_TEXT
--------------------------------------------------------------------------------
--------------------
1         138          Oracle AI Vector Search stores and indexes vector
embeddings for fast retrieval and similarity search.

                          About Oracle AI Vector Search

109         230          About Oracle AI Vector Search
                          Vector Indexes are a new classification of
specialized indexes that are designed for Artificial
                          Intelligence (AI) workloads that allow you to query
data based on semantics, rather than keywords.

343         204          Why Use Oracle AI Vector Search?
                          The biggest benefit of Oracle AI Vector Search is
that semantic search on unstructured data can be
                          combined with relational search on business data in
one single system.
```

Example 4-12 BY words MAX 40 OVERLAP 5 SPLIT BY recursively

This example is similar to Example 4-6, except that an overlap of `5` is used here. The overlapping portion of a chunk must obey the same split condition, in this case must begin at either a blank line, new line, or space.

The text after the second blank line exceeds the maximum words, so the first chunk ends at the first blank line. The second chunk overlaps with 5 words (beginning on a space; underlined below) and includes the second line, but excludes the third line of 33 words. The third chunk overlaps 5 words and ends on the second blank line. The fourth chunk consumes the rest of the input.

Oracle AI Vector Search stores and indexes vector embeddings for fast **retrieval and similarity search.**

About Oracle AI Vector Search

Vector Indexes are a new classification of specialized indexes that are designed for Artificial Intelligence (AI) workloads that allow you to query data based on semantics, **rather than keywords.**

Why Use Oracle AI Vector Search?

The biggest benefit of Oracle AI Vector Search is that semantic search on unstructured data can be combined with relational search on business data in one single system.

Syntax:

```
SELECT C.*
FROM documentation_tab D, VECTOR_CHUNKS(D.text BY words MAX 40 OVERLAP 5
                          SPLIT BY recursively LANGUAGE american NORMALIZE
none) C;
```

Output:

```
CHUNK_OFFSET CHUNK_LENGTH CHUNK_TEXT
--------------------------------------------------------------------------------
--------------------
1         104         Oracle AI Vector Search stores and indexes vector
embeddings for fast retrieval and similarity search.

71           68          retrieval and similarity search.

                         About Oracle AI Vector Search

109         230         About Oracle AI Vector Search
                         Vector Indexes are a new classification of
specialized indexes that are designed for Artificial
                         Intelligence (AI) workloads that allow you to query
data based on semantics, rather than keywords.

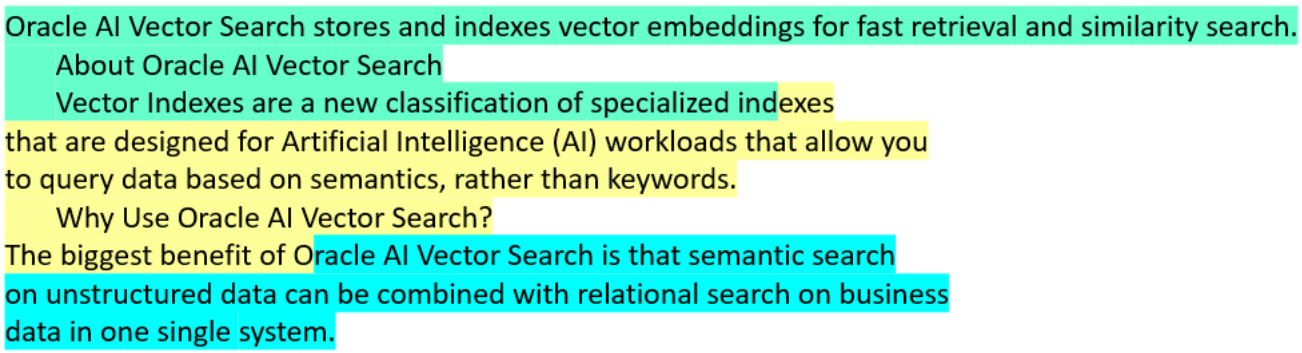
316          231         rather than keywords.

                         Why Use Oracle AI Vector Search?
                         The biggest benefit of Oracle AI Vector Search is
```

```
that semantic search on unstructured data can be
                          combined with relational search on business data in
one single system.
```

Example 4-13 BY chars MAX 200 OVERLAP 0 SPLIT BY none NORMALIZE none

This example is the same as the first one (Example 4-1), to contrast with the next example (Example 4-14) with normalization.



Syntax:

```
SELECT C.*
FROM documentation_tab D, VECTOR_CHUNKS(D.text BY chars MAX 200 OVERLAP 0
                           SPLIT BY none LANGUAGE american NORMALIZE none) C;
```

Output:

```
CHUNK_OFFSET CHUNK_LENGTH CHUNK_TEXT
--------------------------------------------------------------------------------
--------------------
1         200          Oracle AI Vector Search stores and indexes vector
embeddings for fast retrieval and similarity search.

                           About Oracle AI Vector Search
                           Vector Indexes are a new classification of
specialized ind

201         200           exes that are designed for Artificial Intelligence
(AI) workloads that allow you to query data based
                           on semantics, rather than keywords.

                           Why Use Oracle AI Vector Search?
                           The biggest benefit of O

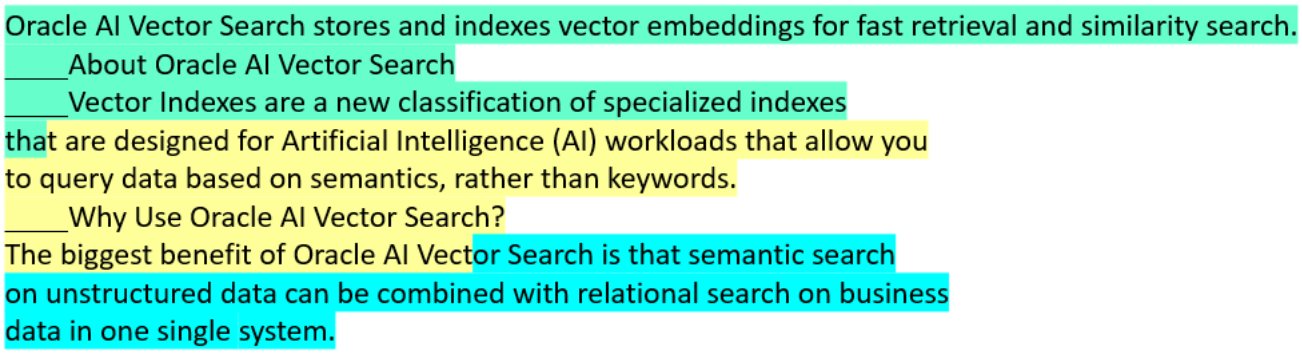
401         146           racle AI Vector Search is that semantic search on
unstructured data can be combined with relational
                           search on business data in one single system.
```

Example 4-14 BY chars MAX 200 OVERLAP 0 SPLIT BY none NORMALIZE whitespace

This example enables `whitespace` normalization, which collapses redundant white space to produce more content within a chunk maximum.

The first chunk extends 8 more characters due to the two indented lines of 4 spaces each (marked with underscores _ below). The second chunk extends 4 more characters due to the one indented line of 4 total spaces. The third chunk has the remaining input.

This example shows that the chunk length (normally in bytes) can differ from the chunk text's size. The `CHUNK_OFFSET` and `CHUNK_LENGTH` represent the original source location of the chunk.



Syntax:

```
SELECT C.*
FROM documentation_tab D, VECTOR_CHUNKS(D.text BY chars MAX 200 OVERLAP 0
                          SPLIT BY none LANGUAGE american NORMALIZE
whitespace) C;
```

Output:

```
CHUNK_OFFSET CHUNK_LENGTH CHUNK_TEXT
--------------------------------------------------------------------------------
--------------------
1         208          Oracle AI Vector Search stores and indexes vector
embeddings for fast retrieval and similarity search.

                          About Oracle AI Vector Search
                          Vector Indexes are a new classification of
specialized indexes tha

209         205          t are designed for Artificial Intelligence (AI)
workloads that allow you to query data based on sema
                          ntics, rather than keywords.

                          Why Use Oracle AI Vector Search?
                          The biggest benefit of Oracle AI Vect

414          133          or Search is that semantic search on unstructured
data can be combined with relational search on business data in one single
system.
```

Example 4-15 BY words MAX 40 OVERLAP 0 SPLIT BY sentence LANGUAGE American

This example uses end-of-sentence splitting which uses language-specific data and heuristics (such as sentence punctuations, contextual rules, or common abbreviations) to determine likely sentence boundaries. Three chunks are produced, each ending at the periods.

You can use this technique to keep your text intact for chunks that contain many split sentences. Otherwise, the text may lose semantic context and may not be useful for queries that target specific information.

Oracle AI Vector Search stores and indexes vector embeddings for fast retrieval and similarity search.

About Oracle AI Vector Search

Vector Indexes are a new classification of specialized indexes that are designed for Artificial Intelligence (AI) workloads that allow you to query data based on semantics, rather than keywords.

Why Use Oracle AI Vector Search?

The biggest benefit of Oracle AI Vector Search is that semantic search on unstructured data can be combined with relational search on business data in one single system.

Syntax:

```
SELECT C.*
FROM documentation_tab D, VECTOR_CHUNKS(D.text BY words MAX 40 OVERLAP 0
                          SPLIT BY sentence LANGUAGE american NORMALIZE
NONE) C;
```

Output:

```
CHUNK_OFFSET CHUNK_LENGTH CHUNK_TEXT
--------------------------------------------------------------------------------
-------------------
1            102          Oracle AI Vector Search stores and indexes vector
embeddings for fast retrieval and similarity search.

109          228          About Oracle AI Vector Search
                          Vector Indexes are a new classification of
specialized indexes that are designed for Artificial
                          Intelligence (AI) workloads that allow you to query
data based on semantics, rather than keywords.

343         203          Why Use Oracle AI Vector Search?
                          The biggest benefit of Oracle AI Vector Search is
that semantic search on unstructured data can be
                          combined with relational search on business data in
one single system.
```

Example 4-16 BY words MAX 40 OVERLAP 0 SPLIT BY sentence LANGUAGE Simplified Chinese

In continuation with the preceding example, this example uses a Simplified Chinese text as the input to specify language-specific sentence chunking.

The output contains four chunks, each ending at the periods:

```
使用 My Oracle Support 之前，您的用户概要信息中必须至少具有一个客户服务号。客户服务号是
标识您所在组织的唯一参考号。使用 My Oracle Support 门户向您的概要信息中添加一个客户服务
号。有关详细信息，请参阅 My Oracle Support 帮助的"如何将客户服务号添加到概要信息？
```

For the purpose of clarity, in this example, `documentation_tab` is a `CLOB` inserted with the following `ChineseDoc.txt` document:

使用 My Oracle Support 之前，您的用□概要信息中必□至少具有一个客□服□号。客□服□号是□□您所在□□的唯一参考号。使用 My Oracle Support □□向您的概要信息中添加一个客□服□号。有关□□信息，□参□ My Oracle Support 帮助的"如何将客□服□号添加到概要信息？

Perform the chunking operation as follows:

```
-- create a relational table

DROP TABLE IF EXISTS documentation_tab;
CREATE TABLE documentation_tab (
    id   NUMBER,
    text CLOB);

-- create a local directory and store the document into the table

CREATE OR REPLACE DIRECTORY VEC_DUMP AS '/my_local_dir/';
CREATE OR REPLACE PROCEDURE my_clob_from_file(
    p_dir in varchar2,
    p_file in varchar2,
    p_id in number
  ) AS
  dest_loc CLOB;
  v_bfile bfile := null;
  v_lang_context number := dbms_lob.default_lang_ctx;
  v_dest_offset integer := 1;
  v_src_offset integer := 1;
  v_warning number;
BEGIN
        insert into documentation_tab values(p_id,empty_clob()) returning text
        into dest_loc;

                v_bfile := BFileName(p_dir, p_file);

                dbms_lob.open(v_bfile, dbms_lob.lob_readonly);
                dbms_lob.loadClobFromFile(
                                dest_loc,
                                v_bfile,
                                dbms_lob.lobmaxsize,
                                v_dest_offset,
                                v_src_offset,
                                873,
                                v_lang_context,
                                v_warning);
```

```
        dbms_lob.close(v_bfile);
END my_clob_from_file;
/

show errors;

-- transform clob into chunks

exec my_clob_from_file('VEC_DUMP', 'ChineseDoc.txt', 1);

SELECT rownum as id, C.chunk_offset pos, C.chunk_length as siz,
       REPLACE(SUBSTR(C.chunk_text,1,15),CHR(10),'_') as beg,
       '...' as rng,
       REPLACE(SUBSTR(C.chunk_text,-15),CHR(10),'_') as end
FROM documentation_tab D, VECTOR_CHUNKS(to_char(D.text) BY words
                                MAX 40
                                OVERLAP 0
                                SPLIT BY sentence
                                LANGUAGE "simplified chinese"
                                NORMALIZE none) C;
```

Output:

```
ID   POS  SIZ  BEG                          RNG END
---- ---- ---- ---------------------------  --- ------------------------
 1    1   103  使用 My Oracle Su             ... 中必⌧至少具有一个客⌧服⌧号。
 2  104    60  客⌧服⌧号是⌧⌧您所在⌧⌧的唯    ... 是⌧⌧您所在⌧⌧的唯一参考号。
 3  164    85  使用 My Oracle Su             ... 概要信息中添加一个客⌧服⌧号。
 4  249   109  有关⌧⌧信息，⌧参⌧ My O        ... 何将客⌧服⌧号添加到概要信息？
```

Related Topics

- VECTOR_CHUNKS
  Use `VECTOR_CHUNKS` to split plain text into smaller chunks to generate vector embeddings that can be used with vector indexes or hybrid vector indexes.
- UTL_TO_CHUNKS
  Use the `DBMS_VECTOR_CHAIN.UTL_TO_CHUNKS` chainable utility function to split a large plain text document into smaller chunks of text.

Create and Use Custom Vocabulary

Create and use your own vocabulary of tokens when chunking data.

Here, you use the chunker helper function `CREATE_VOCABULARY` from the `DBMS_VECTOR_CHAIN` package to load custom vocabulary. This vocabulary file contains a list of tokens, recognized by your vector embedding model's tokenizer.

1. Connect as a local user and prepare your data dump directory.
   a. Log in to SQL*Plus as the `SYS` user, connecting as `SYSDBA`:

```
conn sys/password as sysdba


CREATE TABLESPACE tbs1
DATAFILE 'tbs5.dbf' SIZE 20G AUTOEXTEND ON
```

```
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;


SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 10000
SET LONG 10000
```

b. Create a local user (`docuser`) and grant necessary privileges:

```
drop user docuser cascade;


create user docuser identified by docuser DEFAULT TABLESPACE tbs1 quota
unlimited on tbs1;


grant DB_DEVELOPER_ROLE to docuser;
```

c. Create a local directory (`VEC_DUMP`) to store your vocabulary file. Grant necessary privileges:

```
create or replace directory VEC_DUMP as '/my_local_dir/';


grant read, write on directory VEC_DUMP to docuser;

commit;
```

d. Transfer the vocabulary file for your required model to the `VEC_DUMP` directory.

   For example, if using the WordPiece tokenization, you can download and transfer the `vocab.txt` vocabulary file for "bert-base-uncased".

e. Connect as the local user (`docuser`):

```
conn docuser/password
```

2. Create a relational table (`doc_vocabtab`) to store your vocabulary tokens in it:

```
CREATE TABLE doc_vocabtab(token nvarchar2(64))
  ORGANIZATION EXTERNAL
  (default directory VEC_DUMP
   ACCESS PARAMETERS (RECORDS DELIMITED BY NEWLINE)
   location ('bert-vocabulary-uncased.txt'));
```

3. Create a vocabulary (`doc_vocab`) by calling `DBMS_VECTOR_CHAIN.CREATE_VOCABULARY`:

```
DECLARE
  vocab_params clob := '{
                        "table_name"      : "doc_vocabtab",
```

```
                                "column_name"     : "token",
                                "vocabulary_name" : "doc_vocab",
                                "format"          : "bert",
                                "cased"           : false
                               }';

BEGIN
  dbms_vector_chain.create_vocabulary(json(vocab_params));
END;
/
```

After loading the token vocabulary, you can now use the `BY VOCABULARY` chunking mode (with `VECTOR_CHUNKS` or `UTL_TO_CHUNKS`) to split data by counting the number of tokens.

Related Topics

- CREATE_VOCABULARY
  Use the `DBMS_VECTOR_CHAIN.CREATE_VOCABULARY` chunker helper procedure to load your own token vocabulary file into the database.
- VECTOR_CHUNKS
  Use `VECTOR_CHUNKS` to split plain text into smaller chunks to generate vector embeddings that can be used with vector indexes or hybrid vector indexes.
- UTL_TO_CHUNKS
  Use the `DBMS_VECTOR_CHAIN.UTL_TO_CHUNKS` chainable utility function to split a large plain text document into smaller chunks of text.

Create and Use Custom Language Data

Create and use your own language-specific conditions (such as common abbreviations) when chunking data.

Here, you use the chunker helper function `CREATE_LANG_DATA` from the `DBMS_VECTOR_CHAIN` package to load the data file for Simplified Chinese. This data file contains abbreviation tokens for your chosen language.

1. Connect as a local user and prepare your data dump directory.
   a. Log in to SQL*Plus as the `SYS` user, connecting as `SYSDBA`:

```
conn sys/password as sysdba


CREATE TABLESPACE tbs1
DATAFILE 'tbs5.dbf' SIZE 20G AUTOEXTEND ON
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;


SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 10000
SET LONG 10000
```

b. Create a local user (`docuser`) and grant necessary privileges:

```
drop user docuser cascade;


create user docuser identified by docuser DEFAULT TABLESPACE tbs1 quota
unlimited on tbs1;


grant DB_DEVELOPER_ROLE to docuser;
```

c. Create a local directory (`VEC_DUMP`) to store your language data file. Grant necessary privileges:

```
create or replace directory VEC_DUMP as '/my_local_dir/';


grant read, write on directory VEC_DUMP to docuser;

commit;
```

d. Transfer the data file for your required language to the `VEC_DUMP` directory. For example, `dreoszhs.txt` for Simplified Chinese.

   To know the data file location for your language, see Supported Languages and Data File Locations.

e. Connect as the local user (`docuser`):

```
conn docuser/password
```

2. Create a relational table (`doc_langtab`) to store your abbreviation tokens in it:

```
CREATE TABLE doc_langtab(token nvarchar2(64))
  ORGANIZATION EXTERNAL
  (default directory VEC_DUMP
   ACCESS PARAMETERS (RECORDS DELIMITED BY NEWLINE CHARACTERSET AL32UTF8)
   location ('dreoszhs.txt'));
```

3. Create language data (`doc_lang_data`) by calling `DBMS_VECTOR_CHAIN.CREATE_LANG_DATA`:

```
DECLARE
  lang_params clob := '{
                        "table_name"      : "doc_langtab",
                        "column_name"     : "token",
                        "language"        : "simplified chinese",
                        "preference_name" : "doc_lang_data"
                      }';
BEGIN
  dbms_vector_chain.create_lang_data(json(lang_params));
END;
/
```

After loading the language data, you can now use language-specific chunking by specifying the `LANGUAGE` chunking parameter with `VECTOR_CHUNKS` or `UTL_TO_CHUNKS`.

Related Topics

- CREATE_LANG_DATA
  Use the `DBMS_VECTOR_CHAIN.CREATE_LANG_DATA` chunker helper procedure to load your own language data file into the database.
- VECTOR_CHUNKS
  Use `VECTOR_CHUNKS` to split plain text into smaller chunks to generate vector embeddings that can be used with vector indexes or hybrid vector indexes.
- UTL_TO_CHUNKS
  Use the `DBMS_VECTOR_CHAIN.UTL_TO_CHUNKS` chainable utility function to split a large plain text document into smaller chunks of text.

5

Store Vector Embeddings

Store vector embeddings and associated unstructured data with your relational business data in Oracle Database.

- Create Tables Using the VECTOR Data Type
  You can declare a table's column as a `VECTOR` data type.
- Insert Vectors in a Database Table Using the INSERT Statement
  Once you create a table with a `VECTOR` data type column, you can directly insert vectors into the table using the `INSERT` statement.
- Load Vector Data Using SQL*Loader
  Use these examples to understand how you can load character and binary vector data.
- Unload and Load Vectors Using Oracle Data Pump
  Starting with Oracle Database 23ai, Oracle Data Pump enables you to use multiple components to load and unload vectors to databases.

Create Tables Using the VECTOR Data Type

You can declare a table's column as a `VECTOR` data type.

The following command shows a simple example:

```
CREATE TABLE my_vectors (id NUMBER, embedding VECTOR);
```

In this example, you don't have to specify the number of dimensions or their format, which are both optional. If you don't specify any of them, you can enter vectors of different dimensions with different formats. This is a simplification to help you get started with using vectors in Oracle Database.

> **Note:**
>
> Such vectors typically arise from different embedding models. Vectors from different models (providing a different semantic landscape) are not comparable for use in similarity search.

Here's a more complex example that imposes more constraints on what you can store:

```
CREATE TABLE my_vectors (id NUMBER, embedding VECTOR(768, INT8)) ;
```

In this example, each vector that is stored:

- Must have 768 dimensions, and
- Each dimension will be formatted as an `INT8`.

The number of dimensions must be strictly greater than zero with a maximum of 65535 for non-`BINARY` vectors and 65528 for `BINARY` vectors. If you attempt to use larger values, the following error is raised:

```
ORA-51801: Invalid VECTOR type specification: Invalid dimension
count ('...'). Valid values can either be * (i.e. flexible) or an
integer between 1 and 65535.
```

`BINARY` vectors must have a dimension that is a multiple of 8. If the specified dimension is not a multiple of 8, the following error is raised:

```
ORA-51813: Vector of BINARY format should have a dimension count
that is a multiple of 8.
```

The possible dimension formats are:

- `INT8` (8-bit integers)
- `FLOAT32` (32-bit IEEE floating-point numbers)
- `FLOAT64` (64-bit IEEE floating-point numbers)
- `BINARY` (packed `UINT8` bytes where each dimension is a single bit)

Oracle Database automatically casts the values as needed.

Here is some simple code that shows you how to define and insert a `VECTOR` in a table:

```
DROP TABLE my_vect_tab PURGE;
CREATE TABLE my_vect_tab (v01 VECTOR(3, INT8));
INSERT INTO my_vect_tab VALUES ('[10, 20, 30]');

SELECT * FROM my_vect_tab;

V01
----------
[10,20,30]
```

You use a textual form to represent a vector in SQL statements. The `DENSE` textual form as shown in the preceding `INSERT` statement is a basic example: coordinate 1 has a value of 10, coordinate 2 has a value of 20, and coordinate 3 has a value of 30. You separate each coordinate with a comma and you enclose the list with square brackets.

So far, the vector types shown are, by default, `DENSE` vectors where each dimension is physically stored. All the definitions seen are equivalent to the following form:

```
VECTOR(number_of_dimensions, dimension_element_format, DENSE) or
VECTOR(number_of_dimensions, dimension_element_format, *)
```

However, you also have the option to create `SPARSE` vectors. In contrast to `DENSE` vectors, a sparse vector is a vector whose dimension values are expected to be mostly zero. When using `SPARSE` vectors, only the non-zero values are physically stored. You define sparse vectors using the following form:

```
VECTOR(number_of_dimensions, dimension_element_format, SPARSE)
```

Note:

- It is not supported to have `SPARSE` storage in one row and `DENSE` storage in another row for the same vector column as the coding of the two representations are very different.
- Sparse vectors are not supported with `BINARY` format.
- `VECTOR(…, …, *)` is always interpreted as `DENSE`.
- `SPARSE` vectors cannot be used in the creation of vector indexes.

The following table guides you through the possible declaration format for a `VECTOR` data type with a `DENSE` storage format:

Possible Declaration Format	Explanation
`VECTOR`	Vectors can have an arbitrary number of dimensions and formats.

`VECTOR(*, *)`
`VECTOR(*, *, *)`
`VECTOR(*, *, DENSE)`
	Vectors can have an arbitrary number of dimensions and formats.
	`VECTOR`,
	`VECTOR(*, *)`,
	`VECTOR(*,*,*)`, and
	`VECTOR(*, *, DENSE)` are equivalent.

`VECTOR`(number_of_dimensions) `VECTOR`(number_of_dimensions, *) `VECTOR`(number_of_dimensions, *, *) `VECTOR`(number_of_dimensions, *, `DENSE`)	Vectors must all have the specified number of dimensions or an error is thrown. Every vector will have its dimensions stored without format modification. `VECTOR`(number_of_dimensions), `VECTOR`(number_of_dimensions, *), `VECTOR`(number_of_dimensions, *, *), and `VECTOR`(number_of_dimensions, *, `DENSE`) are equivalent.
`VECTOR`(*, dimension_element_format) `VECTOR`(*, dimension_element_format, *) `VECTOR`(*, dimension_element_format, `DENSE`)	Vectors can have an arbitrary number of dimensions, but their format will be up-converted or down-converted to the specified dimension_element_format (`INT8`, `FLOAT32`, `FLOAT64`, or `BINARY`). `VECTOR`(*, dimension_element_format), `VECTOR`(*, dimension_element_format, *), and `VECTOR`(*, dimension_element_format, `DENSE`) are equivalent.

The following table guides you through the possible declaration format for a `VECTOR` data type for sparse vectors.

Possible Declaration Format	Explanation
`VECTOR(*, *, SPARSE)`	Vectors can have an arbitrary number of dimensions and formats besides `BINARY`.

Possible Declaration Format	Explanation
`VECTOR(number_of_dimensions, *, SPARSE)`	Vectors must all have the specified number of dimensions or an error is thrown. Every vector will have its dimensions stored without format modification.
`VECTOR(*, dimension_element_format, SPARSE)`	Vectors can have an arbitrary number of dimensions but their format will be up-converted or down-converted to the specified `dimension_element_format` (`INT8`, `FLOAT32`, or `FLOAT64`)

The following SQL*Plus code example shows how the system interprets various vector definitions:

```
CREATE TABLE my_vect_tab (
     v1  VECTOR(3, FLOAT32),
     v2  VECTOR(2, FLOAT64),
     v3  VECTOR(1, INT8),
     v4  VECTOR(1024, BINARY),
     v5  VECTOR(1, *),
     v6  VECTOR(*, FLOAT32),
     v7  VECTOR(*, *),
     v8  VECTOR,
     v9  VECTOR(10),
     v10 VECTOR(*, *, DENSE),
     v11 VECTOR(1024, FLOAT32, DENSE),
     v12 VECTOR(1000, INT8, SPARSE),
     v13 VECTOR(*, INT8, SPARSE),
     v14 VECTOR(*, *, SPARSE),
     v15 VECTOR(2048, FLOAT32, *)
   );

Table created.

DESC my_vect_tab;
 Name                          Null?    Type
 ----------------------------- -------- ----------------------------
 V1                                     VECTOR(3 , FLOAT32, DENSE)
 V2                                     VECTOR(2 , FLOAT64, DENSE)
 V3                                     VECTOR(1 , INT8, DENSE)
 V4                                     VECTOR(1024, BINARY, DENSE)
 V5                                     VECTOR(1 , *, DENSE)
 V6                                     VECTOR(* , FLOAT32, DENSE)
 V7                                     VECTOR(* , *, DENSE)
 V8                                     VECTOR(* , *, DENSE)
 v9                                     VECTOR(10, *, DENSE)
 v10                                    VECTOR(*, *, DENSE)
 v11                                    VECTOR(1024, FLOAT32, DENSE)
 v12                                    VECTOR(1000, INT8, SPARSE)
 v13                                    VECTOR(*, INT8, SPARSE)
 v14                                    VECTOR(*, *, SPARSE)
 v15                                    VECTOR(2048, FLOAT32, DENSE)
```

A vector can be `NULL` but its dimensions cannot (for example, you cannot have a `VECTOR` with a `NULL` dimension such as `[1.1, NULL, 2.2]`).

> **Note:**
>
> Vectors (`DENSE` and `SPARSE`) are internally stored as Securefile BLOBs and most popular embedding model vector sizes are between 1.5KB and 12KB in size. You can use the following formula to determine the size of your vectors on disk:
>
> - For `DENSE` vectors:
>   *number of vectors * number of dimensions * size of your vector dimension type*
>   (for example, a `FLOAT32` is equivalent to `BINARY_FLOAT` and is 4 bytes in size).
> - For `SPARSE` vectors:
>   *number of vectors * ((average number of sparse dimensions * 4 bytes) + (number of sparse dimensions * size of your vector dimension type))*.

Restrictions

You currently cannot define `VECTOR` columns in/as:

- IOTs (neither as Primary Key nor as non-Key column)
- Clusters/Cluster Tables
- Global Temp Tables
- (Sub)Partitioning Key
- Primary Key
- Foreign Key
- Unique Constraint
- Check Constraint
- Default Value
- Modify Column
- Manual Segment Space Management (MSSM) tablespace (only SYS user can create VECTORs as Basicfiles in MSSM tablespace)
- Continuous Query Notification (CQN) queries
- Non-vector indexes such as B-tree, Bitmap, Reverse Key, Text, Spatial indexes, etc

Oracle Database does not support the following SQL constructs with `VECTOR` columns:

- Distinct, Count Distinct
- Order By, Group By
- Join condition
- Comparison operators (e.g. >, <, =) etc

Oracle's Globally Distributed Database has the following restrictions on vector data types:

- Sharding keys: A distributed database only supports sharding keys on non-vector columns. The vector data can be distributed across shards using a primary key on any other non-vector column identified as a sharding key.
- Raft replication: A distributed database using the Raft replication method for high availability does not support vector columns.

Topics

- Vectors in External Tables
  External tables can be created with columns of type `VECTOR`, allowing vector embeddings represented in text or binary format stored in external files to be rendered as the `VECTOR` data type in Oracle Database.
- Vectors in Distributed Database Tables
  There is no new SQL syntax or keyword when creating sharded tables and duplicated tables with vector columns in a Globally Distributed Database; however, there are some requirements and restrictions to consider.
- BINARY Vectors
  In addition to `FLOAT32` (the default format for comma-separated string representations of vectors), `FLOAT64`, and `INT8` dimension formats, you can also use the `BINARY` dimension format.
- SPARSE Vectors
  The storage format of a vector can be specified as `SPARSE` or `DENSE`. Sparse vectors are vectors that typically have a large number of dimensions but with very few non-zero dimension values, while Dense vectors are vectors where every dimension stores a value, including zero-values.

Vectors in External Tables

External tables can be created with columns of type `VECTOR`, allowing vector embeddings represented in text or binary format stored in external files to be rendered as the `VECTOR` data type in Oracle Database.

The ability to store `VECTOR` data in external tables can be beneficial when considering the vast amount of data that is involved in AI workflows. Using external tables provides a convenient option to store vector embeddings and contextual information outside of the database while still using the database as a semantic search engine.

The following types of external files support `VECTOR` columns in external tables:

- CSV
- Parquet
- Avro
- ORC
- Dmp

External tables with `VECTOR` columns can be accessed by using the drivers `ORACLE_LOADER`, `ORACLE_DATAPUMP`, and `ORACLE_BIGDATA`. Once the chosen driver has loaded the file data into the database, you can interact with the external table rows in any SQL operation (supported by your preferred SQL interface), such as joins, SQL functions, aggregation, and so on.

Vectors of any dimension format and storage format are supported. If a vector is `SPARSE`, the data must be provided as text as opposed to an array or list format.

Columns of type `VECTOR` can be included in both explicitly created external tables as well as inline external tables created as part of a `SELECT` statement. The benefit of this approach is that there is no need to predefine a static table to access the vectors in the external table before loading them into the database. The external tables mapping is persisted only while the external table is in use by the SQL query. For an example, see Querying an Inline External Table, which shows how the external table mappings are created as part of the SQL query

operation. Once the query has completed, the external table mapping is discarded from the database.

Additionally, the `row_limiting_clause` can be used in `SELECT` statements that reference external tables. Internal and external tables can be referenced in the same query. You can use the `CREATE TABLE AS SELECT` statement to create an internal table by selecting from an external table with `VECTOR` column(s). Similarly, you can use the `INSERT INTO SELECT` statement to insert values into an internal table from an external table called in the `SELECT` subquery.

> **Note:**
>
> - External tables are not currently supported in multi-vector similarity searches.
> - HNSW and IVF indexes cannot currently be created on `VECTOR` columns stored in external tables.

Vector embeddings in external tables can be accessed for use in similarity searches in the same way as you would use an internal table, as in the following query:

```
SELECT id, embedding
FROM external_table
ORDER BY VECTOR_DISTANCE(embedding, '[1,1]', COSINE)
FETCH APPROX FIRST 3 ROWS ONLY WITH TARGET ACCURACY 90;
```

The following examples demonstrate the syntax used to create external tables with `VECTOR` columns depending on the access driver:

- Using `ORACLE_LOADER`:

```
CREATE TABLE ext_vec_tab1(
  v1 VECTOR,
  v2 VECTOR
)
  ORGANIZATION EXTERNAL
  (
    TYPE ORACLE_LOADER
    DEFAULT DIRECTORY my_dir
    ACCESS PARAMETERS
    (
      RECORDS DELIMITED BY NEWLINE
      FIELDS TERMINATED BY ":"
      MISSING FIELD VALUES ARE NULL
    )
    LOCATION('my_ext_vec_embeddings.csv')
  )
REJECT LIMIT UNLIMITED;
```

- Using `ORACLE_DATAPUMP`:

```
-- First create the table with the loader
CREATE TABLE dp_ext_tab(
  country_code         VARCHAR2(5),
```

```
  country_name         VARCHAR2(50),
  country_language     VARCHAR2(50),
  country_vector       VECTOR(*,*)
)
  ORGANIZATION EXTERNAL
  (
    TYPE ORACLE_LOADER
    DEFAULT DIRECTORY my_dir
    ACCESS PARAMETERS
    (
      RECORDS DELIMITED BY NEWLINE
      FIELDS TERMINATED BY ":"
      MISSING FIELD VALUES ARE NULL
      (
        country_code         CHAR(5),
        country_name         CHAR(50),
        country_language     CHAR(50),
        country_vector       CHAR(10000)
      )
    )
    LOCATION ('ext_vectorcountries.dat')
    )
  PARALLEL 5
  REJECT LIMIT UNLIMITED;

-- Then generate the dmp file
CREATE TABLE ext_export_table
ORGANIZATION EXTERNAL
(
  TYPE ORACLE_DATAPUMP
  DEFAULT DIRECTORY my_dir
  LOCATION ('ext.dmp')
)
  AS SELECT * FROM dp_ext_tab;

-- Finally, create an external table with the datapump driver
CREATE TABLE dp_ext_tab_final
(
  country_code         VARCHAR2(5),
  country_name         VARCHAR2(50),
  country_language     VARCHAR2(50),
  country_vector       VECTOR(3, INT8)
)
  ORGANIZATION EXTERNAL
  (
    TYPE ORACLE_DATAPUMP
    DEFAULT DIRECTORY my_dir
    LOCATION ('ext.dmp')
  )
  PARALLEL 5
  REJECT LIMIT UNLIMITED;
```

- Using `ORACLE_BIGDATA`:

```
CREATE TABLE bd_ext_tab
(
```

```
  COL1 vector(5,INT8),
  COL2 vector(5,INT8),
  COL3 vector(5,INT8),
  COL4 vector(5,INT8)
)
  ORGANIZATION external
  (
    TYPE ORACLE_BIGDATA
    DEFAULT DIRECTORY my_dir
    ACCESS PARAMETERS
  (
    com.oracle.bigdata.credential.name\=OCI_CRED
    com.oracle.bigdata.credential.schema\=PDB_ADMIN
    com.oracle.bigdata.fileformat=parquet
    com.oracle.bigdata.debug=true
  )
  LOCATION ( 'https://swiftobjectstorage.<region>.oraclecloud.com/v1/
<namespace>/<filepath>/basic_vec_data.parquet' )
  )
REJECT LIMIT UNLIMITED PARALLEL 2;
```

- Querying an Inline External Table
  In this example, vectors in an external table of type `ORACLE_BIGDATA` are queried as part of a vector search.
- Performing a Semantic Similarity Search Using External Table
  See a SQL example plan that illustrates how you can use external tables as the data set for semantic similarity searches

> **See Also:**
>
> - *Oracle Database Utilities* for more information about external tables

Querying an Inline External Table

In this example, vectors in an external table of type `ORACLE_BIGDATA` are queried as part of a vector search.

```
select * from external (
    (
      COL1 vector,
      COL2 vector,
      COL3 vector,
      COL4 vector
    )
    TYPE ORACLE_BIGDATA
    DEFAULT DIRECTORY DEF_DIR1
    ACCESS PARAMETERS
    (
      com.oracle.bigdata.credential.name\=OCI_CRED
      com.oracle.bigdata.credential.schema\=PDB_ADMIN
      com.oracle.bigdata.fileformat=parquet
```

```
      com.oracle.bigdata.debug=true
    )
    location ( 'https://swiftobjectstorage.us-phoenix-1.oraclecloud.com/v1/
myvdataproject/BIGDATA_PARQUET/vector_data/basic_vector_data.parquet' )
    REJECT LIMIT UNLIMITED
  ) tkexobd_bd_vector_inline;
```

Performing a Semantic Similarity Search Using External Table

See a SQL example plan that illustrates how you can use external tables as the data set for semantic similarity searches

The following is an example of an explain plan for `select id`, embedding from `ext_table_3`, and using `order by vector_distance('[1,1]', embedding, cosine)` to return approximately only the first three rows of data with a target accuracy of 90 percent:

```
SQL> select * from table(dbms_xplan.display('plan_table', null, 'advanced
predicate'));

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--
Plan hash value: 1784440045

--------------------------------------------------------------------------------
--
--------------------

| Id  | Operation                     | Name        | Rows  | Bytes |TempSpc|
Co
st (%CPU)| Time     |

--------------------------------------------------------------------------------
--
--------------------


PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--
|   0 | SELECT STATEMENT              |             |     3 | 48945 |
|  1
466K  (2)| 00:00:58 |

|*  1 |  COUNT STOPKEY                |             |       |       |       |
         |          |

|   2 |   VIEW                        |             |   102K|  1588M|
|  1
466K  (2)| 00:00:58 |

|*  3 |    SORT ORDER BY STOPKEY      |             |   102K|  1589M|
798M|  1
466K  (2)| 00:00:58 |
```

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--

|   4 |     EXTERNAL TABLE ACCESS FULL| EXT_TABLE_3 |   102K|  1589M|       |
362   (7)| 00:00:01 |

--------------------------------------------------------------------------------
--
---------------------


Query Block Name / Object Alias (identified by operation id):
-------------------------------------------------------------


PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--
   1 - SEL$2
   2 - SEL$1 / "from$_subquery$_002"@"SEL$2"
   3 - SEL$1
   4 - SEL$1 / "EXT_TABLE_3"@"SEL$1"

Outline Data
-------------

  /*+
      BEGIN_OUTLINE_DATA
      FULL(@"SEL$1" "EXT_TABLE_3"@"SEL$1")

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--
      NO_ACCESS(@"SEL$2" "from$_subquery$_002"@"SEL$2")
      OUTLINE_LEAF(@"SEL$2")
      OUTLINE_LEAF(@"SEL$1")
      ALL_ROWS
      OPT_PARAM('_fix_control' '6670551:0')
      OPT_PARAM('_optimizer_cost_model' 'fixed')
      DB_VERSION('26.1.0')
      OPTIMIZER_FEATURES_ENABLE('26.1.0')
      IGNORE_OPTIM_EMBEDDED_HINTS
      END_OUTLINE_DATA
  */

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter(ROWNUM<=3)
   3 - filter(ROWNUM<=3)
```

```
Column Projection Information (identified by operation id):
-----------------------------------------------------------

   1 - "from$_subquery$_002"."ID"[NUMBER,22],
"from$_subquery$_002"."EMBEDDING"[

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------
--
VECTOR,32600]

   2 - "from$_subquery$_002"."ID"[NUMBER,22],
"from$_subquery$_002"."EMBEDDING"[
VECTOR,32600]

   3 - (#keys=1) VECTOR_DISTANCE(VECTOR('[1,1]', *, *, * /*+
USEBLOBPCW_QVCGMD
*/ ),

       "EMBEDDING" /*+ LOB_BY_VALUE */ , COSINE)[BINARY_DOUBLE,8],
"ID"[NUMBER,2
2], "EMBEDDING" /*+


PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------
--
       LOB_BY_VALUE */ [VECTOR,32600]
   4 - "ID"[NUMBER,22], "EMBEDDING" /*+ LOB_BY_VALUE */ [VECTOR,32600],
       VECTOR_DISTANCE(VECTOR('[1,1]', *, *, * /*+  USEBLOBPCW_QVCGMD */ ),
"EMB
EDDING" /*+

       LOB_BY_VALUE */ , COSINE)[BINARY_DOUBLE,8]

Query Block Registry:
---------------------

  SEL$1 (PARSER) [FINAL]

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------
--
  SEL$2 (PARSER) [FINAL]
```

Vectors in Distributed Database Tables

There is no new SQL syntax or keyword when creating sharded tables and duplicated tables with vector columns in a Globally Distributed Database; however, there are some requirements and restrictions to consider.

User Permissions

Only an all-shards user can create sharded and duplicated tables. You must connect to the shard catalog as an all-shards user. Connecting to the shard catalog as an all-shards user automatically enables `SHARD DDL`, and the DDL to create the tables is propagated to all the shards in the distributed database.

Creating Sharded Tables with a Vector Column

- Sharded tables must be created on the catalog database with `SHARD DDL` enabled.
- A vector column cannot be part of the sharding key or the partitionset key.
- The `CREATE SHARDED TABLE` command is propagated to all of the shards by the shard coordinator.

The syntax to create a sharded table with a vector column is same as the syntax to create a non-sharded table with a vector column. The only difference is to include the `SHARDED` keyword in the `CREATE TABLE` statement.

```
CREATE SHARDED TABLE REALTORS(
     REALTOR_ID NUMBER PRIMARY KEY,
     NAME VARCHAR2(20),
     IMAGE VECTOR,
     ZIPCODE VARCHAR2(40))
PARTITION BY CONSISTENT HASH(REALTOR_ID)
TABLESPACE SET TS1;
```

Creating Duplicated Tables with a Vector Column

- Duplicated tables must be created on the shard catalog database with `SHARD DDL` enabled.

The syntax to create a duplicated table with a vector column is same as the syntax to create a non-sharded table with a vector column. The only difference is to include the `DUPLICATED` keyword in the `CREATE TABLE` statement.

```
CREATE DUPLICATED TABLE PRODUCT_DESCRIPTIONS
     (
     PRODUCT_ID          NUMBER(6,0) NOT NULL,
     ORDER_ID            NUMBER(6,0) NOT NULL,
     LANGUAGE_ID         VARCHAR2(6 BYTE),
     TRANSLATED_NAME     NVARCHAR2(50),
     TRANSLATED_DESCRIPTION NVARCHAR2(2000),
     VECT4 VECTOR,
     VECT5 VECTOR,
     CONSTRAINT  PRODUCT_DESCRIPTIONS_PK primary key (PRODUCT_ID)
     ) tablespace products
     STORAGE (INITIAL 1M NEXT 1M);
```

BINARY Vectors

In addition to `FLOAT32` (the default format for comma-separated string representations of vectors), `FLOAT64`, and `INT8` dimension formats, you can also use the `BINARY` dimension format.

A `BINARY` vector represents each dimension as a single bit (0 or 1). The following statement is an example of declaring a 1024 dimension vector using the `BINARY` format:

```
VECTOR(1024, BINARY)
```

The main advantages of using the `BINARY` format are:

- The storage footprint of vectors can be reduced by a factor of 32 compared to the default `FLOAT32` format.
- Distance computations between two vectors are up to 40 times faster.

The downside of using the `BINARY` format is the potential loss of accuracy. However, the loss is often not very substantial. `BINARY` vector accuracy is often greater than 90% compared to that of `FLOAT32` vectors. Several third-party providers have added embedding models that have the ability to generate binary embeddings, including Cohere, Hugging Face, and Jina AI.

`BINARY` vectors are stored as packed `UINT8` bytes (unsigned integer). This means that a single byte represents exactly 8 `BINARY` dimensions and no less.

Note:

- The default distance metric for `BINARY` vectors is `HAMMING`.
- Conversions between `BINARY` vectors and any other dimension format are not currently supported.
- A column can be declared as `VECTOR(*, BINARY)`. In this case, '`*`' means that vectors can have an arbitrary number of dimensions. However, because the maximum possible number of dimensions supported is 65535 for other formats, you cannot exceed a `UINT8` array of size 8191 for a `BINARY` vector, which represents 8191 * 8 = 65528 dimensions, the greatest multiple of 8 less than 65535.
- Oracle does not currently support using OML4Py to export `BINARY` models to ONNX format and import them in the Oracle Database.
- Oracle Database server does not currently provide quantization techniques. When applicable, this should be handled by the client.

The following is an example of a Cohere `INT8` embedding and a `UBINARY` embedding. Note when considering this example that Oracle Database server works with the data provided in the `VECTOR` constructor as it is received. Any quantization logic necessary for processing the data into binary format should be handled by the client. The example is provided to help in understanding the concept.

```
INT8 Embedding of 1024 dimensions from Cohere embed-english-v3.0:
[25, 11, -99, -114, 13, -17, -59, 44, 65, 33, -50, -2, 28, -16, -6, -20, -33,
```

```
49, -59, -50, 0, -82, -67, 10, 82, -2, -126, -28, -32, -69, -13, 120, 54, 4,
-71, 24, 4, -37, -57, 34, 16, -7, 27, -74, -12, 13, 1, -24, 65, -24, 28, 46,
25, -33, -25, 36, 3, -47, 12, -49, -17, 11, 53, 70, -18, 10, -8, 4, 0, -33,
10, -3, 27, -24, -35, -24, 23, -32, 0, -4, -21, -7, -29, -48, -7, -28, -25,
-8, 54, -7, 14, -8, 39, 78, 0, -13, 26, 2, 40, 27, -35, -26, 5, -23, 15, 72,
-4, -5, 33, 14, 18, 11, 0, -6, 6, -16, -53, 56, -35, 15, -1, -8, 83, 28, -2,
27, -34, -60, 36, 4, 14, 21, -69, 17, -22, 0, 16, -77, 29, 27, 26, 0, 81, 15,
-90, 7, 22, -2, -26, -39, -31, -10, 2, 32, -30, 40, -71, 29, 2, 36, -72, -6,
42, -16, -16, 6, 40, 30, 1, -31, -42, 31, 56, 18, 0, 9, 27, 59, 11, 38, 28,
-30, 73, -10, -56, 6, 17, 87, 15, 1, 49, -33, -68, 0, 10, -49, 18, -10, 8,
12,
52, -31, 7, -37, -25, -53, 9, -5, 72, 14, -37, -41, 30, -54, -60, 30, -62,
20,
3, 7, 64, -7, 48, 16, 19, 1, -43, -18, -91, -6, -113, 104, 42, 61, -24, -15,
20, -9, 4, 36, 27, 46, -30, -39, 43, -14, 53, -36, -4, 35, 74, 37, 1, -19,
62,
12, -13, 8, -11, 21, -4, 96, 29, 17, -99, 2, -67, -32, -55, -8, 55, 16, -29,
28, 47, 47, -77, 0, -24, 1, 1, 38, 28, -11, 2, -55, 4, 18, 42, 99, 98, 1, 17,
18, -21, 4, 89, 66, -32, 17, 56, 14, -2, -45, 19, -30, 26, 14, 34, -36, 5,
74,
50, 33, 47, -37, 34, 61, -8, -62, 46, 56, -55, 0, 33, 5, -72, -29, -48, 21,
40,
22, 3, 39, -1, 10, 32, -47, 28, 19, 92, -5, -13, 2, 12, -21, -33, -9, 31, -2,
-25, -20, -14, 1, 53, -34, -26, 17, 72, -35, -36, -26, -86, -20, 55, -4, -53,
-14, 47, 26, 82, -3, -41, -18, -40, -94, 87, 3, -17, 38, 54, 17, 62, -23, 61,
20, -4, 18, 37, 21, -37, -10, -43, -32, -40, -29, 43, 75, -44, -3, 47, 9,
-10,
29, -26, 55, 35, -17, 43, 37, -8, 19, 0, -32, -49, 43, -27, 16, -81, 34, 56,
15, -33, -13, -30, -13, -28, 54, -61, -90, -45, -101, -52, -101, 5, 22, 7,
72,
-30, 31, 27, 42, -47, -6, -30, -30, 42, 13, -23, 63, -84, -20, -17, 61, -40,
35, 37, 21, -8, 110, 108, 26, -49, -1, -31, 8, 10, 7, 29, -67, -29, 72, 15,
11,
4, -34, 12, 28, -48, -21, -81, 38, -29, 26, 4, 10, 29, -11, 26, -78, -51,
-52,
27, -92, -23, -5, -11, 31, 18, -33, -49, 7, -51, -35, -57, -14, 121, -8, 29,
25,
70, -19, 29, 48, -41, 48, -18, 19, -18, -13, 46, 27, 47, 42, 1, -33, 20, -27,
8,
-31, 31, 1, 0, 11, -4, 32, -65, -7, 9, -11, 15, 3, -34, 42, -15, -71, -5, 3,
8,
-8, 22, -7, -70, 10, 21, -127, -114, 13, -11, 46, -13, -10, -10, 29, -59, 43,
-1, -17, -21, 8, -15, 12, 1, -73, -26, -5, 6, 37, 23, 46, 73, 14, -74, 84,
-2,
-22, -6, 5, -7, -26, 28, -39, -23, -22, 14, 38, 0, -2, 41, 27, -65, 30, 3,
-23,
53, 86, 35, -32, -48, -15, 32, 21, -26, -48, -26, 32, 32, 4, -70, -72, -62,
-28,
-14, -86, -10, -63, 44, -68, -41, 27, -52, 33, -56, -30, 5, 84, -54, 16, -22,
-20, 16, 34, 14, -25, 8, -14, -13, -28, -40, 16, 41, -5, -88, -35, 55, -82,
55,
74, -55, -12, 58, 57, -83, -26, 55, 32, -6, 42, -14, 35, -5, -36, 84, -40,
-29,
7, -20, -17, 23, -20, -49, -48, 22, 49, -30, 35, 48, 5, 34, 17, 13, 30, 33,
-38,
-37, 10, -52, -24, 67, -15, -12, -3, -11, -46, -7, 32, 10, -46, 3, 18, -7,
```

```
-26,
0, -40, 23, -46, 89, 37, 3, -29, -51, -32, 49, -51, 9, 16, -47, -26, 14, 10,
14,
-13, 11, 16, -18, 54, -24, 18, -14, -51, -89, -24, 20, 12, 2, 62, 13, 53,
-22,
2, 22, -14, 29, -9, 51, -42, -97, 28, 49, -4, -93, -17, -26, 46, 47, 33, -33,
25, 81, -29, 5, 17, 24, 54, -10, -14, -2, 29, 17, -4, -47, 56, 4, 9, 30, -87,
39, -16, 39, 67, -13, 37, 13, 67, 50, -16, -55, 8, 24, -50, -1, -36, -51,
-20,
-58, 11, -28, -22, -26, 16, 7, -17, 39, -9, -21, -9, -8, -18, 37, -47, -19,
36,
-8, 6, -39, 58, -26, -37, 11, 86, 33, 67, -35, 25, -11, -7, -22, 20, 14, 8,
8,
7, -30, -58, 37, -1, 16, -13, 89, -6, 81, -46, -37, -7, 9, -23, -11, -41,
-13,
18, -17, -4, -42, 0, 91, -128, 33, -18, -88, -84, -11, -62, 79, -34, -39, 54,
-17, -14, 15, 79, -33, -4, 30, 5, 8, -55, -9, -38, 10, -41, 37, -5, 2, 62, 3,
-5, -42, 17, -50, 14, -58, -16, 26, -20, -49, 52, 73, -42, 9, 7, -50, 14,
-11,
39, 0, -45, -90, -30, -16, -19, -6, -1, 43, -7, -47, -4, 40, -6, 5, 2, 2,
-20,
-40, 39, 10, -16, 64, -11, -36, -5, 37, -16, 49, 24, -20, 17, 27, -21, -49,
-49,
-38, -19, -31, -2, 15, 52, -68, -14, 20, 38, 10, -48, -2, -52, -60, -55, -30,
37, -32, -80, 1, -1, -12, -45, 15, 29, 8, -46, -42, -28, -38, 11, 4, 19, 2,
67,
-44, -5, -28, 21, 17, -16, -34, 16, -6, 10, -11, 15, 2, 33, -25, -13, 8, -7,
2,
-22, 21, -41, 10, -29, -36, 46, 19, -41, 36, -39, 10, -23, -13, -2, -53, 39,
-25, -4]
```

```
UBINARY Embedding from the same model (1024 dimensions = 128 packed UINT8
bytes)
[201, 200, 65, 129, 217, 166, 185, 167, 90, 138, 0, 172, 242, 207, 165, 52,
245,
187, 96, 215, 39, 159, 250, 126, 107, 162, 201, 123, 193, 203, 202, 123, 87,
67,
113, 235, 253, 220, 187, 236, 220, 125, 185, 136, 102, 8, 224, 222, 220, 12,
214,
217, 92, 16, 61, 195, 69, 220, 121, 236, 94, 136, 100, 46, 212, 250, 189, 45,
26,
101, 20, 88, 253, 18, 51, 110, 49, 192, 37, 52, 232, 98, 204, 212, 146, 55,
249,
32, 108, 174, 44, 237, 67, 246, 166, 29, 188, 103, 173, 230, 4, 104, 37, 79,
71,
202, 162, 16, 160, 147, 56, 174, 82, 109, 96, 34, 230, 139, 96, 51, 129, 35,
135,
198, 87, 42, 154, 132]
```

The `BINARY` vector is generated through a binary quantization mechanism using the following rule:

- If the `INT8` dimension value > 0, the `BINARY` dimension value is 1
- If the `INT8` dimension value <= 0, the `BINARY` dimension format is 0

Consider the first 8 `INT8` dimensions from the preceding example:

```
[25, 11, -99, -114, 13, -17, -59, 44]
```

In `BINARY`, this translates to:

```
[1, 1, 0, 0, 1, 0, 0, 1]
```

Representing this as a `UINT8` byte makes it 201, which is the first byte value of the packed `UINT8` representation. So, each `BINARY` vector can therefore be inserted as a `UINT8` array whose size is: *number of `BINARY` vector dimensions/8*.

> **Note:**
>
> `BINARY` vectors are only supported with a number of dimensions that is a multiple of 8.

The following is an example of an invalid declaration of a `BINARY` vector column, due to the fact that the vector dimension, 12, is not divisible by 8:

```
CREATE TABLE vectab (id NUMBER, data VECTOR(12, BINARY));
```

Result:

```
CREATE TABLE vectab (id NUMBER, data VECTOR(12, BINARY))
                                                       *
ERROR at line 1:
ORA-51813: Vector of BINARY format should have a dimension count that is a
multiple of 8.
```

The following statements are an example of a valid table creation with a `BINARY` vector column and a valid insert (string representation):

```
CREATE TABLE vectab(id NUMBER, data VECTOR(16, BINARY));
INSERT INTO vectab VALUES (1, '[201, 15]');
SELECT data FROM vectab;
```

Result:

```
DATA
---------
[201,15]
```

These next statements are examples of invalid inserts (string representation):

```
SQL> INSERT INTO vectab VALUES (1, '[201]');
INSERT INTO vectab VALUES (1, '[201]')
                              *
ERROR at line 1:
```

```
ORA-51803: Vector dimension count must match the dimension count specified in
the column definition (actual: 8, required: 16).

SQL> INSERT INTO vectab VALUES (1, '[201, 15, 123]');
INSERT INTO vectab VALUES (1, '[201, 15, 123]')
                              *
ERROR at line 1:
ORA-51803: Vector dimension count must match the dimension count specified in
the column definition (actual: 24, required: 16).

SQL> INSERT INTO vectab VALUES (1, '[256, 15]');
INSERT INTO vectab VALUES (1, '[256, 15]')
                              *
ERROR at line 1:
ORA-51806: Vector column is not properly formatted (dimension value 1 is
outside the allowed precision range).
```

SPARSE Vectors

The storage format of a vector can be specified as `SPARSE` or `DENSE`. Sparse vectors are vectors that typically have a large number of dimensions but with very few non-zero dimension values, while Dense vectors are vectors where every dimension stores a value, including zero-values.

Sparse vectors can be generated by Sparse Encoding models such as SPLADE or BM25. Generally speaking, sparse models such as SPLADE outperform dense models, such as BERT and All-MiniLM, in keyword awareness search. They are also widely used for Hybrid Vector Search by combining sparse and dense vectors.

Conceptually, a sparse vector can be thought of as a vector where every dimension corresponds to a keyword in a certain vocabulary. For a given document, the sparse vector contains non-zero dimension values representing the number of occurrences for the keywords within that document. For example, BERT has a vocabulary size of 30,522 and several sparse encoders generate vectors of this dimensionality.

Representing a dense vector with 30,522 dimensions with only 100 non-zero `FLOAT32` dimension values would still require *30,522 * 4 = ~120KB* of storage. Such a format takes up a lot of space for no reason as most of the dimension values are 0. This would cause a huge performance deficit compared to the `SPARSE` representation of such vectors.

That is why when using `SPARSE` vectors, only the non-zero dimension values are physically stored.

Here is an example of creating and inserting a `SPARSE` vector:

```
DROP TABLE my_sparse_tab PURGE;
CREATE TABLE my_sparse_tab (v01 VECTOR(5, INT8, SPARSE));

INSERT INTO my_sparse_tab VALUES('[5,[2,4],[10,20]]');
INSERT INTO my_sparse_tab VALUES('[[2,4],[10,20]]');

SELECT * FROM my_sparse_tab;

V01
--------------------
```

```
[5,[2,4],[10,20]]
[5,[2,4],[10,20]]
```

You can see the difference between the `SPARSE` textual form within the `INSERT` statement with the one used for a `DENSE` vector. The `SPARSE` textual form looks like:

```
'[Total Dimension Count, [Dimension Index Array], [Dimension Value Array]]'
```

The example uses the number of dimensions in total (5 here but it is optional to specify it in this case as it is defined in the column's declaration), then gives the list of coordinates that have non-zero values, then the list of the corresponding values. In this example, coordinate 2 has the value 10 and coordinate 4 has the value 20. Coordinates 1, 3, and 5 have the value 0.

It is not permitted to use a `DENSE` textual form for `SPARSE` vectors and vice versa. However, it is possible to use vector functions to transform one into the other as illustrated in the following sample code using the table `my_sparse_tab` (created in the previous snippet):

The following `INSERT` statement fails:

```
INSERT INTO my_sparse_tab VALUES('[0, 10, 0, 20, 0]');

Error starting at line : 1 in command -
INSERT INTO my_sparse_tab VALUES ('[0,10,0,20,0]')
Error at Command Line : 1 Column : 33
Error report -
SQL Error: ORA-51833: Textual input conversion between sparse and dense
vector is not
supported.
```

However, this insertion works:

```
INSERT INTO my_sparse_tab VALUES (TO_VECTOR('[0,0,10,0,20]', 5, INT8, DENSE));

SELECT * FROM my_sparse_tab;

V01
____________________
[5,[2,4],[10,20]]
[5,[2,4],[10,20]]
[5,[3,5],[10,20]]
```

You can also transform a `SPARSE` vector into a `DENSE` textual form if needed and vice versa:

```
SELECT FROM_VECTOR(v01 RETURNING CLOB FORMAT DENSE)
FROM my_sparse_tab
WHERE ROWNUM<2;

FROM_VECTOR(V01RETURNINGCLOBFORMATDENSE)
_______________________________________
[0,10,0,20,0]
```

> **Note:**
>
> The `RETURNING` clause used in the preceding example can also return a `VARCHAR2` or a `BLOB`.

Insert Vectors in a Database Table Using the INSERT Statement

Once you create a table with a `VECTOR` data type column, you can directly insert vectors into the table using the `INSERT` statement.

The following examples assume you have already created vectors and know their values.

Here is a simple example:

```
DROP TABLE galaxies PURGE;
CREATE TABLE galaxies (id NUMBER, name VARCHAR2(50), doc VARCHAR2(500),
embedding VECTOR);

INSERT INTO galaxies VALUES (1, 'M31', 'Messier 31 is a barred spiral galaxy
in the Andromeda constellation which has a lot of barred spiral galaxies.',
'[0,2,2,0,0]');
INSERT INTO galaxies VALUES (2, 'M33', 'Messier 33 is a spiral galaxy in the
Triangulum constellation.', '[0,0,1,0,0]');
INSERT INTO galaxies VALUES (3, 'M58', 'Messier 58 is an intermediate barred
spiral galaxy in the Virgo constellation.', '[1,1,1,0,0]');
INSERT INTO galaxies VALUES (4, 'M63', 'Messier 63 is a spiral galaxy in the
Canes Venatici constellation.', '[0,0,1,0,0]');
INSERT INTO galaxies VALUES (5, 'M77', 'Messier 77 is a barred spiral galaxy
in the Cetus constellation.', '[0,1,1,0,0]');
INSERT INTO galaxies VALUES (6, 'M91', 'Messier 91 is a barred spiral galaxy
in the Coma Berenices constellation.', '[0,1,1,0,0]');
INSERT INTO galaxies VALUES (7, 'M49', 'Messier 49 is a giant elliptical
galaxy in the Virgo constellation.', '[0,0,0,1,1]');
INSERT INTO galaxies VALUES (8, 'M60', 'Messier 60 is an elliptical galaxy in
the Virgo constellation.', '[0,0,0,0,1]');
INSERT INTO galaxies VALUES (9, 'NGC1073', 'NGC 1073 is a barred spiral
galaxy in Cetus constellation.', '[0,1,1,0,0]');
COMMIT;
```

Here is a more sophisticated example:

```
DROP TABLE doc_queries PURGE;
CREATE TABLE doc_queries (id NUMBER, query VARCHAR2(500), embedding VECTOR);

DECLARE
  e CLOB;
BEGIN
e:=
'[-7.73346797E-002,1.09683955E-002,4.68435362E-002,2.57333983E-002,6.95586428E
-00'||
'2,-2.43412293E-002,-7.25011379E-002,6.66433945E-002,3.78751606E-002,-2.223544
75E'||
'-002,3.02388351E-002,9.36625451E-002,-1.65204913E-003,3.50606232E-003,-5.4773
```

```
859'||
'7E-002,-7.5879097E-002,-2.72218436E-002,7.01764375E-002,-1.32512336E-003,3.14
728'||
'022E-002,-1.39147148E-001,-7.52705336E-002,2.62449421E-002,1.91645715E-002,4.
055'||
'73137E-002,5.83701171E-002,-3.26474942E-002,2.0509012E-002,-3.81141738E-003,-
7.1'||
'8656182E-002,-1.95893757E-002,-2.56917924E-002,-6.57705888E-002,-4.39117625E-
002'||
',-6.82357177E-002,1.26592368E-001,-3.46683599E-002,1.07687116E-001,-3.9695449
2E-'||
'002,-9.06721968E-003,-2.4109887E-002,-1.29214963E-002,-4.82468568E-002,-3.872
307'||
'76E-002,5.13443872E-002,-1.40985977E-002,-1.87066793E-002,-1.11725368E-002,9.
367'||
'76772E-002,-6.39425665E-002,3.13162468E-002,8.61801133E-002,-5.5481784E-002,4
.13'||
'125418E-002,2.0447813E-002,5.03717586E-002,-1.73418857E-002,3.94522659E-002,-
7.2'||
'6833269E-002,3.13266069E-002,1.2377765E-002,7.64856935E-002,-3.77447419E-002,
-6.'||
'41075056E-003,1.1455299E-001,1.75497644E-002,4.64923214E-003,1.89623125E-002,
9.1'||
'3506597E-002,-8.22509527E-002,-1.28537193E-002,1.495138E-002,-3.22528258E-002
,-4'||
'.71280375E-003,-3.15563753E-003,2.20409594E-002,7.77796134E-002,-1.927099E-00
2,-'||
'1.24283969E-001,4.69769612E-002,1.78121701E-002,1.67152807E-002,-3.83916795E-
002'||
',-1.51029453E-002,2.10864041E-002,6.86845928E-002,-7.4719809E-002,1.17681816E
-00'||
'3,3.93113159E-002,6.04066066E-002,8.55340436E-002,3.68878953E-002,2.41579115E
-00'||
'2,-5.92489541E-002,-1.21883564E-002,-1.77226216E-002,-1.96259264E-002,8.51236
377'||
'E-003,1.43039867E-001,2.62829307E-002,2.96348184E-002,1.92485824E-002,7.66567
141'||
'E-002,-1.18600562E-001,3.01779062E-002,-5.88010997E-002,7.07774982E-002,-6.60
426'||
'617E-002,6.44619241E-002,1.29240509E-002,-2.51785964E-002,2.20869959E-004,-2.
514'||
'38171E-002,5.52265197E-002,8.65883753E-002,-1.83726232E-002,-8.13263431E-002,
1.1'||
'6624301E-002,1.63392909E-002,-3.54643688E-002,2.05128491E-002,4.67337575E-003
,1.'||
'20488718E-001,-4.89500947E-002,-3.80397178E-002,6.06209273E-003,-1.37961926E-
002'||
',4.68355882E-031,3.35873142E-002,6.20040558E-002,2.13472452E-002,-1.87379227E
-00'||
'3,-5.83158981E-004,-4.04266678E-002,2.40761992E-002,-1.93725452E-002,9.376372
4E-'||
'002,-3.02913114E-002,7.67844869E-003,6.11112304E-002,6.02455214E-002,-6.38855
845'||
'E-002,-8.03523697E-003,2.08786246E-003,-7.45898336E-002,8.74964818E-002,-5.02
371'||
'937E-002,-4.99385223E-003,3.37120108E-002,8.99377018E-002,1.09540671E-001,5.8
```

```
501'||
'102E-002,1.71627291E-002,-3.26152593E-002,8.36912021E-002,5.05600758E-002,-9.
737'||
'63615E-002,-1.40264994E-002,-2.07926836E-002,-4.20163684E-002,-5.97197041E-00
2,1'||
'.32461395E-002,2.26361351E-003,8.1473738E-002,-4.29272018E-002,-3.86809185E-0
02,'||
'-8.24682564E-002,-3.89646105E-002,1.9992901E-002,2.07321253E-002,-1.74706057E
-00'||
'2,4.50415723E-003,4.43851873E-002,-9.86309871E-002,-7.68082142E-002,-4.538143
05E'||
'-003,-8.90906602E-002,-4.54972908E-002,-5.71065396E-002,2.10020249E-003,1.224
947'||
'07E-002,-6.70659095E-002,-6.52298108E-002,3.92126441E-002,4.33384106E-002,4.3
899'||
'6181E-002,5.78813367E-002,2.95345876E-002,4.68395352E-002,9.15119275E-002,-9.
629'||
'58392E-003,-5.96637605E-003,1.58674959E-002,-6.74034096E-003,-6.00510836E-002
,2.'||
'67188111E-003,-1.10706768E-003,-6.34015873E-002,-4.80389707E-002,6.84534572E-
003'||
',-1.1547043E-002,-3.44865513E-003,1.18979132E-002,-4.31232266E-002,-5.9022788
E-0'||
'02,4.87607308E-002,3.95954074E-003,-7.95252472E-002,-1.82770658E-002,1.182642
49E'||
'-002,-3.79164703E-002,3.87993976E-002,1.09805465E-002,2.29136664E-002,-7.2278
082'||
'4E-002,-5.31538352E-002,6.38669729E-002,-2.47980515E-003,-9.6999377E-002,-3.7
566'||
'7699E-002,4.06541862E-002,-1.69874367E-003,5.58868013E-002,-1.80723771E-033,-
6.6'||
'5985467E-003,-4.45010923E-002,1.77929532E-002,-4.8369132E-002,-1.49722975E-00
2,-'||
'3.97582203E-002,-7.05247298E-002,3.89178023E-002,-8.26886389E-003,-3.91006246
E-0'||
'02,-7.02963024E-002,-3.91333885E-002,1.76661201E-002,-5.09723537E-002,2.37749
107'||
'E-002,-1.83419678E-002,-1.2693027E-002,-1.14232123E-001,-6.68751821E-002,7.52
167'||
'869E-003,-9.94713791E-003,6.03599809E-002,6.61353692E-002,3.70595567E-002,-2.
019'||
'52495E-002,-2.40410417E-002,-3.36526595E-002,6.20064288E-002,5.50279953E-002,
-2.'||
'68641673E-002,4.35859226E-002,-4.57317568E-002,2.76936609E-002,7.88119733E-00
2,-'||
'4.78852056E-002,1.08523415E-002,-6.43479973E-002,2.0192951E-002,-2.09538229E-
002'||
',-2.2202393E-002,-1.0728148E-003,-3.09607089E-002,-1.67067181E-002,-6.0357227
9E-'||
'002,-1.58187654E-002,3.45828459E-002,-3.45360823E-002,-4.4002533E-003,1.77463
517'||
'E-002,6.68234832E-004,6.14458732E-002,-5.07084019E-002,-1.21073434E-002,4.195
981'||
'85E-002,3.69152687E-002,1.09461844E-002,1.83413982E-001,-3.89185362E-002,-5.1
846'||
'0497E-002,-8.71620141E-003,-1.17692262E-001,4.04785499E-002,1.07505821E-001,1
```

```
.41'||
'624091E-002,-2.57720836E-002,2.6652012E-002,-4.50568087E-002,-3.34110335E-002
,-1'||
'.11387551E-001,-1.29796984E-003,-6.51671961E-002,5.36890551E-002,1.0702607E-0
01,'||
'-2.34011523E-002,3.97406481E-002,-1.01149324E-002,-9.95831117E-002,-4.4019784
8E-'||
'002,6.88989647E-003,4.85475454E-003,-3.94048765E-002,-3.6099229E-002,-5.40755
13E'||
'-002,8.58292207E-002,1.0697281E-002,-4.70785573E-002,-2.96272673E-002,-9.4919
120'||
'9E-003,1.57316476E-002,-5.4926388E-002,6.49022609E-002,2.55531631E-002,-1.839
057'||
'17E-002,4.06873561E-002,4.74951901E-002,-1.22502812E-033,-4.6441108E-002,3.74
079'||
'868E-002,9.14599106E-004,6.09740615E-002,-7.67391697E-002,-6.32521287E-002,-2
.17'||
'353106E-002,2.45231949E-003,1.50869079E-002,-4.96984981E-002,-3.40828523E-002
,8.'||
'09691194E-003,3.31339166E-002,5.41345142E-002,-1.16213948E-001,-2.49572527E-0
02,'||
'5.00682592E-002,5.90037219E-002,-2.89178211E-002,8.01460445E-003,-3.41945067E
-00'||
'2,-8.60121697E-002,-6.20261126E-004,2.26721354E-002,1.28968194E-001,2.8765536
8E-'||
'002,-2.20255274E-002,2.7228903E-002,-1.12029864E-002,-3.20301466E-002,4.98079
099'||
'E-002,2.89051589E-002,2.413591E-002,3.64605561E-002,6.26017479E-003,6.5463289
6E-'||
'002,1.11282602E-001,-3.60428065E-004,1.95987038E-002,6.16615731E-003,5.935930
46E'||
'-002,1.50377362E-003,2.95319762E-002,2.56325547E-002,-1.72190219E-002,-6.5816
819'||
'7E-002,-4.08149995E-002,2.7983617E-002,-6.80195764E-002,-3.52494679E-002,-2.9
840'||
'0577E-002,-3.04043181E-002,-1.9352382E-002,5.49411364E-002,8.74160081E-002,5.
614'||
'25127E-002,-5.60747795E-002,-3.43311466E-002,9.83581021E-002,2.01142877E-002,
1.3'||
'193069E-002,-3.22583504E-002,8.54402035E-002,-2.20514946E-002]';

INSERT INTO doc_queries VALUES (13, 'different methods of backup and
recovery', e);
COMMIT;
END;
/
```

You can also create a table that holds `BINARY` vectors:

```
DROP TABLE my_bin_tab PURGE;
CREATE TABLE my_bin_tab(id NUMBER, data VECTOR(16, BINARY));

INSERT INTO my_bin_tab VALUES (1, '[201, 15]');
```

Here's an additional example demonstrating the insertion of `SPARSE` vectors into a table:

```
DROP TABLE my_sparse_tab PURGE;
CREATE TABLE my_sparse_tab(v01 VECTOR(5, INT8, SPARSE));

INSERT INTO my_sparse_tab VALUES ('[5,[2,4],[10,20]]');
INSERT INTO my_sparse_tab VALUES ('[[2,4],[10,20]]');
```

You can also generate vectors by calling services outside the database or generate vectors directly from within the database after you have imported pretrained embedding models.

If you have already loaded an embedding model into the database, called, let's say, `MyModel`, you can insert vectors using the `VECTOR_EMBEDDING` function. For example:

```
INSERT INTO doc_queries VALUES (13, 'different methods of backup and
recovery',
  vector_embedding(MyModel using 'different methods of backup and recovery'
as data));”
```

See Also:

- Import Pretrained Models in ONNX Format
- Convert Text String to Embedding Within Oracle Database

Load Vector Data Using SQL*Loader

Use these examples to understand how you can load character and binary vector data.

SQL*Loader supports loading `VECTOR` columns from character data and binary floating point array `fvec` files. The format for `fvec` files is that each binary 32-bit floating point array is preceded by a four (4) byte value, which is the number of elements in the vector. There can be multiple vectors in the file, possibly with different dimensions. Export and import of a table with vector datatype columns is supported in all the modes (FULL, SCHEMA, TABLES) using all the available methods (access_method=direct_path, access_method=external_table, access_method=automatic) for unloading/loading data.

- Load Character Vector Data Using SQL*Loader Example
  In this example, you can see how to use SQL*Loader to load vector data into a five-dimension vector space.
- Load Binary Vector Data Using SQL*Loader Example
  In this example, you can see how to use SQL*Loader to load binary vector data files.

Load Character Vector Data Using SQL*Loader Example

In this example, you can see how to use SQL*Loader to load vector data into a five-dimension vector space.

Let's imagine we have the following text documents classifying galaxies by their types:

- **DOC1**: "Messier 31 is a barred spiral galaxy in the Andromeda constellation which has a lot of barred spiral galaxies."
- **DOC2**: "Messier 33 is a spiral galaxy in the Triangulum constellation."
- **DOC3**: "Messier 58 is an intermediate barred spiral galaxy in the Virgo constellation."
- **DOC4**: "Messier 63 is a spiral galaxy in the Canes Venatici constellation."
- **DOC5**: "Messier 77 is a barred spiral galaxy in the Cetus constellation."
- **DOC6**: "Messier 91 is a barred spiral galaxy in the Coma Berenices constellation."
- **DOC7**: "NGC 1073 is a barred spiral galaxy in Cetus constellation."
- **DOC8**: "Messier 49 is a giant elliptical galaxy in the Virgo constellation."
- **DOC9**: "Messier 60 is an elliptical galaxy in the Virgo constellation."

You can create vectors representing the preceding galaxy's classes using the following five-dimension vector space based on the count of important words appearing in each document:

Table 5-1 Five dimension vector space

Galaxy Classes	Intermediate	Barred	Spiral	Giant	Elliptical
M31	0	2	2	0	0
M33	0	0	1	0	0
M58	1	1	1	0	0
M63	0	0	1	0	0
M77	0	1	1	0	0
M91	0	1	1	0	0
M49	0	0	0	1	1
M60	0	0	0	0	1
NGC1073	0	1	1	0	0

This naturally gives you the following vectors:

- **M31**: `[0,2,2,0,0]`
- **M33**: `[0,0,1,0,0]`
- **M58**: `[1,1,1,0,0]`
- **M63**: `[0,0,1,0,0]`
- **M77**: `[0,1,1,0,0]`
- **M91**: `[0,1,1,0,0]`
- **M49**: `[0,0,0,1,1]`
- **M60**: `[0,0,0,0,1]`
- **NGC1073**: `[0,1,1,0,0]`

You can use SQL*Loader to load this data into the `GALAXIES` database table defined as:

```
drop table galaxies purge;
create table galaxies (id number, name varchar2(50), doc varchar2(500),
embedding vector);
```

Based on the data described previously, you can create the following `galaxies_vec.csv` file:

```
1:M31:Messier 31 is a barred spiral galaxy in the Andromeda constellation
which has a lot of barred spiral galaxies.:[0,2,2,0,0]:
2:M33:Messier 33 is a spiral galaxy in the Triangulum constellation.:
[0,0,1,0,0]:
3:M58:Messier 58 is an intermediate barred spiral galaxy in the Virgo
constellation.:[1,1,1,0,0]:
4:M63:Messier 63 is a spiral galaxy in the Canes Venatici constellation.:
[0,0,1,0,0]:
5:M77:Messier 77 is a barred spiral galaxy in the Cetus constellation.:
[0,1,1,0,0]:
6:M91:Messier 91 is a barred spiral galaxy in the Coma Berenices
constellation.:[0,1,1,0,0]:
7:M49:Messier 49 is a giant elliptical galaxy in the Virgo constellation.:
[0,0,0,1,1]:
8:M60:Messier 60 is an elliptical galaxy in the Virgo constellation.:
[0,0,0,0,1]:
9:NGC1073:NGC 1073 is a barred spiral galaxy in Cetus constellation.:
[0,1,1,0,0]:
```

Here is a possible SQL*Loader control file `galaxies_vec.ctl`:

```
recoverable
LOAD DATA
infile 'galaxies_vec.csv'
INTO TABLE galaxies
fields terminated by ':'
trailing nullcols
(
id,
name char (50),
doc  char (500),
embedding char (32000)
)
```

> **Note:**
>
> You cannot use comma-delimited vectors (vectors separated by commas) as the field terminator in your CSV file. You must use another deliminator. In these examples the deliminator is a colon (`:`).

After you have created the two files `galaxies_vec.csv` and `galaxies_vec.ctl`, you can run the following sequence of instructions directly from your favorite SQL command line tool:

```
host sqlldr vector/vector@CDB1_PDB1 control=galaxies_vec.ctl
log=galaxies_vec.log

SQL*Loader: Release 23.0.0.0.0 - Development on Thu Jan 11 19:46:21 2024
Version 23.4.0.23.00

Copyright (c) 1982, 2024, Oracle and/or its affiliates.  All rights reserved.
```

```
Path used:      Conventional
Commit point reached - logical record count 10

Table GALAXIES2:
  9 Rows successfully loaded.

Check the log file:
  galaxies_vec.log
for more information about the load.

SQL>

select * from galaxies;

 ID NAME    DOC
EMBEDDING
--- ------  ------------------------------------------------------------
--------------------------------
  1 M31     Messier 31 is a barred spiral galaxy in the Andromeda ...
[0,2.0E+000,2.0E+000,0,0]
  2 M33     Messier 33 is a spiral galaxy in the Triangulum ...
[0,0,1.0E+000,0,0]
  3 M58     Messier 58 is an intermediate barred spiral galaxy ...
[1.0E+000,1.0E+000,1.0E+000,0,0]
  4 M63     Messier 63 is a spiral galaxy in the Canes Venatici ...
[0,0,1.0E+000,0,0]
  5 M77     Messier 77 is a barred spiral galaxy in the Cetus ...
[0,1.0E+000,1.0E+000,0,0]
  6 M91     Messier 91 is a barred spiral galaxy in the Coma ...
[0,1.0E+000,1.0E+000,0,0]
  7 M49     Messier 49 is a giant elliptical galaxy in the Virgo ...
[0,0,0,1.0E+000,1.0E+000]
  8 M60     Messier 60 is an elliptical galaxy in the Virgo ...
[0,0,0,0,1.0E+000]
  9 NGC1073 NGC 1073 is a barred spiral galaxy in Cetus ...
[0,1.0E+000,1.0E+000,0,0]

9 rows selected.

SQL>
```

Here is the resulting log file for this load (`galaxies_vec.log`):

```
cat galaxies_vec.log

SQL*Loader: Release 23.0.0.0.0 - Development on Thu Jan 11 19:46:21 2024
Version 23.4.0.23.00

Copyright (c) 1982, 2024, Oracle and/or its affiliates.  All rights reserved.

Control File:   galaxies_vec.ctl
Data File:      galaxies_vec.csv
  Bad File:     galaxies_vec.bad
  Discard File:  none specified
```

```
 (Allow all discards)

Number to load: ALL
Number to skip: 0
Errors allowed: 50
Bind array:     250 rows, maximum of 1048576 bytes
Continuation:    none specified
Path used:      Conventional

Table GALAXIES, loaded from every logical record.
Insert option in effect for this table: INSERT
TRAILING NULLCOLS option in effect

Column Name   Position   Len  Term Encl Datatype
----------- ---------- ----- ---- ---- ----------
ID               FIRST     *   :       CHARACTER
NAME              NEXT    50   :       CHARACTER
DOC               NEXT   500   :       CHARACTER
EMBEDDING         NEXT 32000   :       CHARACTER

value used for ROWS parameter changed from 250 to 31


Table GALAXIES2:
  9 Rows successfully loaded.
  0 Rows not loaded due to data errors.
  0 Rows not loaded because all WHEN clauses were failed.


Space allocated for bind array:                1017234 bytes(31 rows)
Read   buffer bytes: 1048576

Total logical records skipped:          0
Total logical records read:             9
Total logical records rejected:         0
Total logical records discarded:        1

Run began on Thu Jan 11 19:46:21 2024
Run ended on Thu Jan 11 19:46:24 2024

Elapsed time was:     00:00:02.43
CPU time was:         00:00:00.03
$
```

> **Note:**
>
> This example uses `embedding char (32000)` vectors. For very large vectors, you can use the `LOBFILE` feature

Related Topics

- Loading LOB Data from LOBFILEs

Load Binary Vector Data Using SQL*Loader Example

In this example, you can see how to use SQL*Loader to load binary vector data files.

The vectors in a binary (`fvec`) file are stored in raw 32-bit Little Endian format.

Each vector takes 4+*d**4 bytes for the `.fvecs` file where the first 4 bytes indicate the dimensionality (*d*) of the vector (that is, the number of dimensions in the vector) followed by *d**4 bytes representing the vector data, as described in the following table:

Table 5-2 Fields for Vector Dimensions and Components

Field	Field Type	Description
d	int	The vector dimension
components	array of floats	The vector components

For binary `fvec` files, they must be defined as follows:

- You must specify `LOBFILE`.
- You must specify the syntax format `fvecs` to indicate that the dafafile contains binary dimensions.
- You must specify that the datafile contains raw binary data (`raw`).

The following is an example of a control file used to load `VECTOR` columns from binary floating point arrays using the galaxies vector example described in Understand Hierarchical Navigable Small World Indexes, but in this case importing `fvecs` data, using the control file syntax `format "fvecs"`:

> **Note:**
>
> SQL*Loader supports loading `VECTOR` columns from character data and binary floating point array `fvec` files. The format for `fvec` files is that each binary 32-bit floating point array is preceded by a four (4) byte value, which is the number of elements in the vector. There can be multiple vectors in the file, possibly with different dimensions.

```
LOAD DATA
infile 'galaxies_vec.csv'
INTO TABLE galaxies
fields terminated by ':'
trailing nullcols
(
id,
name char (50),
doc  char (500),
embedding lobfile (constant '/u01/data/vector/embedding.fvecs' format
"fvecs")  raw
)
```

The data contained in `galaxies_vec.csv` in this case does not have the vector data. Instead, the vector data will be read from the secondary `LOBFILE` in the `/u01/data/vector` directory (`/u01/data/vector/embedding.fvecs`), which contains the same information in `float32` floating point binary numbers, but is in `fvecs` format:

```
1:M31:Messier 31 is a barred spiral galaxy in the Andromeda constellation
which has a lot of barred spiral galaxies.:
2:M33:Messier 33 is a spiral galaxy in the Triangulum constellation.:
3:M58:Messier 58 is an intermediate barred spiral galaxy in the Virgo
constellation.:
4:M63:Messier 63 is a spiral galaxy in the Canes Venatici constellation.:
5:M77:Messier 77 is a barred spiral galaxy in the Cetus constellation.:
6:M91:Messier 91 is a barred spiral galaxy in the Coma Berenices
constellation.:
7:M49:Messier 49 is a giant elliptical galaxy in the Virgo constellation.:
8:M60:Messier 60 is an elliptical galaxy in the Virgo constellation.:
9:NGC1073:NGC 1073 is a barred spiral galaxy in Cetus constellation.:
```

Unload and Load Vectors Using Oracle Data Pump

Starting with Oracle Database 23ai, Oracle Data Pump enables you to use multiple components to load and unload vectors to databases.

Oracle Data Pump technology enables very high-speed movement of data and metadata from one database to another. Oracle Data Pump is made up of three distinct components: Command-line clients, expdp and impdp; the DBMS_DATAPUMP PL/SQL package (also known as the Data Pump API); and the DBMS_METADATA PL/SQL package (also known as the Metadata API).

Unloading and Loading a table with vector datatype columns is supported in all modes (`FULL`, `SCHEMA`, `TABLES`) using all the available access methods (`DIRECT_PATH`, `EXTERNAL_TABLE`, `AUTOMATIC`, `INSERT_AS_SELECT`).

Examples Vector Export and Import Syntax

```
expdp <username>/<password>@<Database-instance-TNS-alias>  dumpfile=<dumpfile-
name>.dmp directory=<directory-name> full=y metrics=y
access_method=direct_path

expdp <username>/<password>@<Database-instance-TNS-alias>  dumpfile=<dumpfile-
name>.dmp directory=<directory-name> schemas=<schema-name> metrics=y
access_method=external_table

expdp <username>/<password>@<Database-instance-TNS-alias>  dumpfile=<dumpfile-
name>.dmp directory=<directory-name> tables=<schema-name>.<table-name>
metrics=y access_method=direct_path

impdp <username>/<password>@<Database-instance-TNS-alias>  dumpfile=<dumpfile-
name>.dmp directory=<directory-name> metrics=y access_method=direct_path
```

Note:

- `TABLE_EXISTS_ACTION=APPEND | TRUNCATE` can only be used with the `EXTERNAL_TABLE` access method.
- `TABLE_EXISTS_ACTION=APPEND | TRUNCATE` can load `VECTOR` column data into a `VARCHAR2` column if the conversion can fit into that `VARCHAR2`.
- `TABLE_EXISTS_ACTION=APPEND | TRUNCATE` can only load a `VECTOR` column with the source `VECTOR` data dimasion that matches that loaded `VECTOR` column's dimension. If the dimension does not match, then an error is raised.
- `TABLE_EXISTS_ACTION=REPLACE` supports any access method.
- It is not possible to use a the transportable tablespace mode with vector indexes. However, this mode supports tables with the `VECTOR` datatype.

Related Topics

- Overview of Oracle Data Pump
- DBMS_DATAPUMP
- DBMS_METADATA

6

Create Vector Indexes and Hybrid Vector Indexes

Depending on whether to use similarity search or hybrid search, you can create either a vector index or a hybrid vector index.

Vector indexes are a class of specialized indexing data structures that are designed to accelerate similarity searches using high-dimensional vectors. They use techniques such as clustering, partitioning, and neighbor graphs to group vectors representing similar items, which drastically reduces the search space, thereby making the search process extremely efficient. Create vector indexes on your vector embeddings to use these indexes for running similarity searches over huge vector spaces.

Hybrid vector indexes leverage the existing Oracle Text indexing data structures and vector indexing data structures. Such indexes can provide more relevant search results by integrating the keyword matching capabilities of text search with the semantic precision of vector search. Create hybrid vector indexes directly on your input data or on your vector embeddings to use these indexes for performing a combination of full-text search and similarity search.

- Size the Vector Pool
  To allow vector index creation, you must enable a new memory area stored in the SGA called the **Vector Pool**.
- Manage the Different Categories of Vector Indexes
- Manage Hybrid Vector Indexes
  Learn how to manage a hybrid vector index, which is a single index for searching by *similarity* and *keywords*, to enhance the accuracy of your search results.
- Vector Indexes in a Globally Distributed Database
  Inverted File Flat (IVF) index and Hierarchical Navigable Small World (HNSW) index are supported on sharded tables in a distributed database; however there are some considerations.

Size the Vector Pool

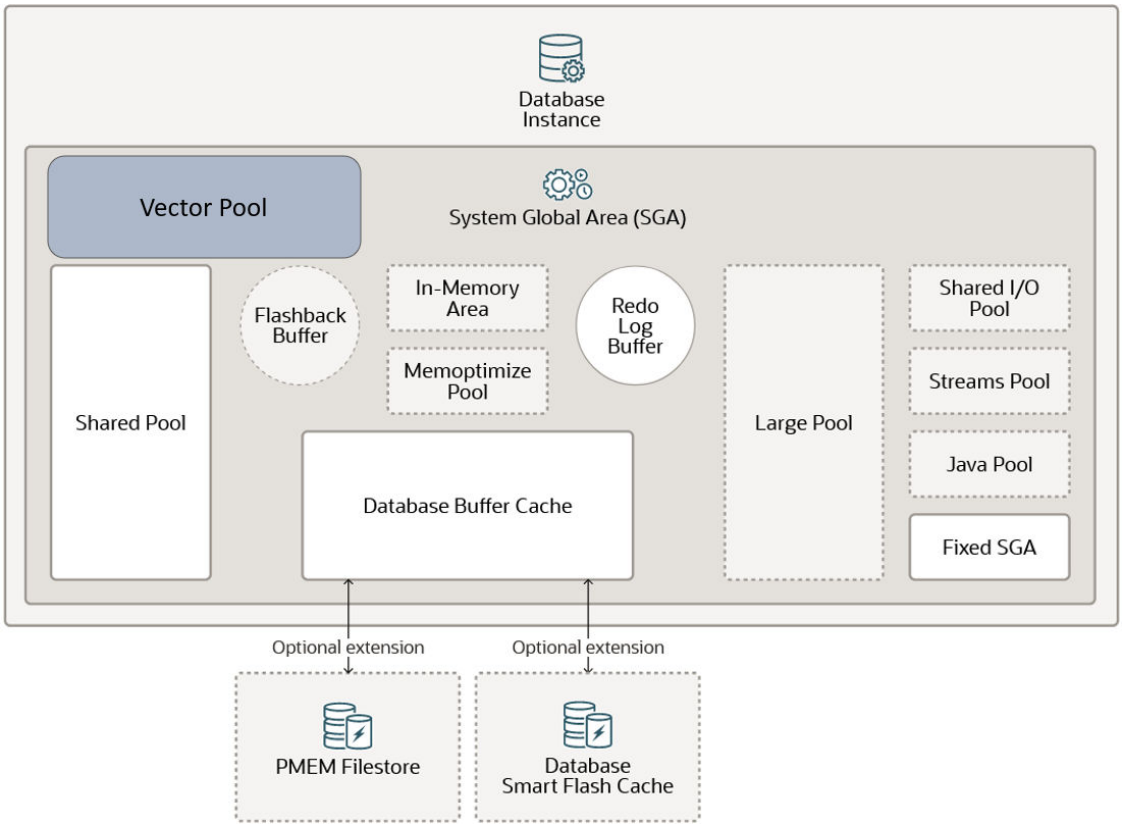
To allow vector index creation, you must enable a new memory area stored in the SGA called the **Vector Pool**.

The Vector Pool is a memory allocated in SGA to store Hierarchical Navigable Small World (HNSW) vector indexes and all associated metadata. It is also used to speed up Inverted File Flat (IVF) index creation as well as DML operations on base tables with IVF indexes.

> **Note:**
>
> IVF centroid vectors are stored in the shared pool if they don't fit in the vector pool.

Enabling a Vector Pool is illustrated in the following diagram:

Figure 6-1 Vector Pool



To size the Vector Pool in an on-premises environment, use the `VECTOR_MEMORY_SIZE` initialization parameter. You can dynamically modify this parameter at the following levels:

- At the CDB level `VECTOR_MEMORY_SIZE` specifies the current size of the Vector Pool. Reducing the parameter value will fail if there is current vector usage.
- At the PDB level `VECTOR_MEMORY_SIZE` specifies the maximum Vector Pool usage allowed by a PDB. Reducing the parameter value will be allowed even if current vector usage exceeds the new quota.

You can change the value of a parameter in a parameter file in the following ways:

- By editing an initialization parameter file. In most cases, the new value takes effect the next time you start an instance of the database.
- By issuing an `ALTER SYSTEM SET ... SCOPE=SPFILE` statement to update a server parameter file.
- By issuing an `ALTER SYSTEM RESET` statement to clear an initialization parameter value and set it back to its default value.

Here is an example of how to change the value for `VECTOR_MEMORY_SIZE` at the PDB level if you are using an SPFILE:

```
SQL> show con_name
CON_NAME
------------------------------
MYPDB1
```

```
SQL> show user
USER is "SYS"

SQL> show parameter vector_memory_size
NAME                 TYPE        VALUE
-------------------- ----------- -----
vector_memory_size big integer 500M

SQL> SELECT ISPDB_MODIFIABLE
  2  FROM   V$SYSTEM_PARAMETER
  3* WHERE  NAME='vector_memory_size';

ISPDB_MODIFIABLE
____________________
TRUE

SQL> ALTER SYSTEM SET vector_memory_size=1G SCOPE=BOTH;

System altered.

SQL> show parameter vector_memory_size

NAME                  TYPE        VALUE
--------------------- ----------- -------
vector_memory_size  big integer 1G
SQL>
```

For more information about changing initialization parameter values, see Managing Initialization Parameters Using a Server Parameter File.

If `VECTOR_MEMORY_SIZE` is set to 1 and the `sga_target` is greater than 0 at CDB initialization, HNSW index creation will automatically grow the vector memory pool to satisfy the new index. The maximum PDB `VECTOR_MEMORY_SIZE` value is limited to 70% of the PDB `sga_target`. Dropping an HNSW index shrinks the vector memory pool accordingly.

In this configuration, where the vector memory pool grows automatically, the PDB `VECTOR_MEMORY_SIZE` value will default to 0 and cannot be changed using the `ALTER SYSTEM` command. Changes in vector pool size as a result of automatic growth are not persisted to the spfile, so when the database is restarted, the vector pool size is reset.

You can query the `V$VECTOR_MEMORY_POOL` view to monitor the Vector Pool.

> **Note:**
>
> To roughly determine the memory size needed to store an HNSW index, use the following formula: *1.3 * number of vectors * number of dimensions * size of your vector dimension type* (for example, a `FLOAT32` is equivalent to `BINARY_FLOAT` and is 4 bytes in size).

> **See Also:**
>
> - Vector Memory Pool Views

Vector Pool Size with Autonomous Database Serverless Services

When using Autonomous Database Serverless services (ADB-S), you cannot explicitly set any SGA-related memory parameters. This includes direct modification of the Vector Pool size.

With ADB-S, the Vector Pool can dynamically grow and shrink:

- An increase to the Vector Pool size is automatically triggered by HNSW index creation and when reopening a PDB containing HNSW indexes.
- A reduction to the Vector Pool size is automatically triggered by an HNSW index drop, PDB close, and CPU reduction.

> **Note:**
>
> Vector Pool size is limited to a maximum of 70% of the PDB SGA size. The SGA size can be retrieved by using the following query:
>
> ```
> SELECT value FROM V$PARAMETER WHERE name='sga_target';
> ```

> **Note:**
>
> Changing instance configuration can cause a previously created HNSW index to be evicted due to insufficient Vector Pool memory. For example, this could be the case if one instance with 15 OCPUs is split into two instances, each with 8 OCPUs.
>
> You can check how much memory is allocated to the Vector Pool after HNSW index creation using a `SELECT` statement similar to the following:
>
> ```
> SELECT sum(alloc_bytes) FROM V$VECTOR_MEMORY_POOL;
> ```

> **Note:**
>
> Until an HNSW index is created, `V$VECTOR_MEMORY_POOL` shows `0` in the `ALLOC_BYTES` column.

Manage the Different Categories of Vector Indexes

There are two ways to make vector searches faster:

- Reduce the search scope by clustering vectors (nearest neighbors) into structures based on certain attributes and restricting the search to closest clusters.
- Reduce the vector size by reducing the number of bits representing vectors values.

Oracle AI Vector Search supports the following categories of vector indexing methods based on approximate nearest-neighbors (ANN) search:

- In-Memory Neighbor Graph Vector Index
- Neighbor Partition Vector Index

The distance function used to create and search the index should be the one recommended by the embedding model used to create the vectors. You can specify this distance function at the time of index creation or when you perform a similarity search using the `VECTOR_DISTANCE()` function. If you use a different distance function than the one used to create the index, an exact match is triggered because you cannot use the index in this case.

> **Note:**
>
> - Oracle AI Vector Search indexes supports the same distance metrics as the `VECTOR_DISTANCE()` function. `COSINE` is the default metric if you do not specify any metric at the time of index creation or during a similarity search using the `VECTOR_DISTANCE()` function.
> - You should always define the distance metric in an index based on the distance metric used by the embedding model you are using.

- In-Memory Neighbor Graph Vector Index
  Hierarchical Navigable Small World (HNSW) is the only type of In-Memory Neighbor Graph vector index supported. HNSW graphs are very efficient indexes for vector approximate similarity search. HNSW graphs are structured using principles from small world networks along with layered hierarchical organization.
- Neighbor Partition Vector Index
  Inverted File Flat (IVF) index is the only type of Neighbor Partition vector index supported. Inverted File Flat Index (IVF Flat or simply IVF) is a partitioned-based index that lets you balance high-search quality with reasonable speed.
- Guidelines for Using Vector Indexes
  Use these guidelines to create and use Hierarchical Navigable Small World (HNSW) or Inverted File Flat (IVF) vector indexes.
- Index Accuracy Report
  The index accuracy reporting feature lets you determine the accuracy of your vector indexes.
- Vector Index Status, Checkpoint, and Advisor Procedures
  Review these high-level details on the `GET_INDEX_STATUS`, `ENABLE_CHECKPOINT`, `DISABLE_CHECKPOINT`, and `INDEX_VECTOR_MEMORY_ADVISOR` procedures that are available with the `DBMS_VECTOR` PL/SQL package.

In-Memory Neighbor Graph Vector Index

Hierarchical Navigable Small World (HNSW) is the only type of In-Memory Neighbor Graph vector index supported. HNSW graphs are very efficient indexes for vector approximate similarity search. HNSW graphs are structured using principles from small world networks along with layered hierarchical organization.

- About In-Memory Neighbor Graph Vector Index
  The default type of index created for an In-Memory Neighbor Graph vector index is Hierarchical Navigable Small World.
- Hierarchical Navigable Small World Index Syntax and Parameters
  Review the syntax and examples for Hierarchical Navigable Small World vector indexes.

About In-Memory Neighbor Graph Vector Index

The default type of index created for an In-Memory Neighbor Graph vector index is Hierarchical Navigable Small World.

- Understand Hierarchical Navigable Small World Indexes
  Use these examples to understand how to create HNSW indexes for vector approximate similarity searches.
- Understand Transaction Support for Tables with HNSW Indexes
  Hierarchical Navigable Small World (HNSW) index graphs are static memory-only structures. Transaction maintenance on tables with HNSW indexes is done by using two main structures: private journal and shared journal.
- Understand HNSW Index Population Mechanisms in Oracle RAC or Single Instance
  Learn how Hierarchical Navigable Small World (HNSW) indexes are populated during index creation, index repopulation, or instance startup in an Oracle Real Application Clusters (Oracle RAC) or a non-RAC environment.

Understand Hierarchical Navigable Small World Indexes

Use these examples to understand how to create HNSW indexes for vector approximate similarity searches.

With Navigable Small World (NSW), the idea is to build a proximity graph where each vector in the graph connects to several others based on three characteristics:

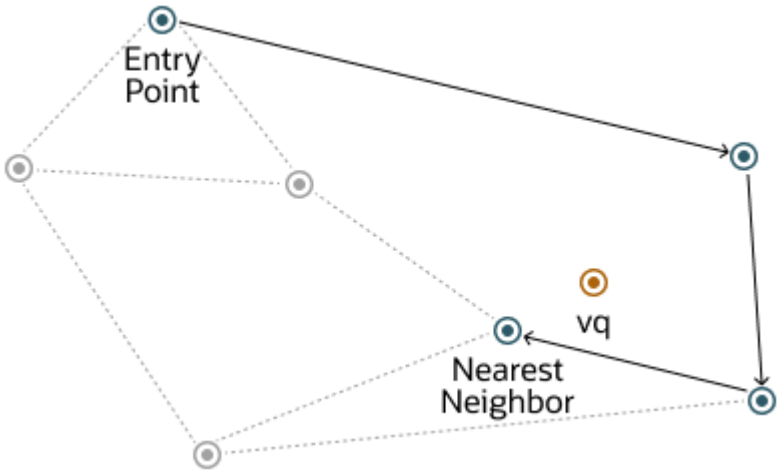
- The distance between vectors
- The maximum number of closest vector candidates considered at each step of the search during insertion `(EFCONSTRUCTION)`
- Within the maximum number of connections (`NEIGHBORS`) permitted per vector

If the combination of the above two thresholds is too high, then you may end up with a densely connected graph, which can slow down the search process. On the other hand, if the combination of those thresholds is too low, then the graph may become too sparse and/or disconnected, which makes it challenging to find a path between certain vectors during the search.

Navigable Small World (NSW) graph traversal for vector search begins with a predefined entry point in the graph, accessing a cluster of closely related vectors. The search algorithm employs two key lists: Candidates, a dynamically updated list of vectors that we encounter while traversing the graph, and Results, which contains the vectors closest to the query vector found thus far. As the search progresses, the algorithm navigates through the graph, continually refining the Candidates by exploring and evaluating vectors that might be closer than those in the Results. The process concludes once there are no vectors in the Candidates closer than the farthest in the Results, indicating a local minimum has been reached and the closest vectors to the query vector have been identified.

This is illustrated in the following graphic:

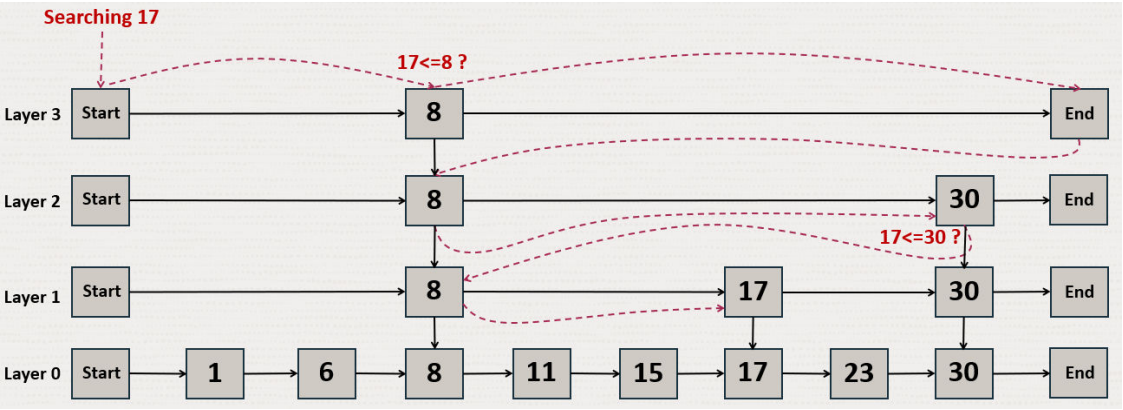
Figure 6-2 Navigable Small World Graph



The described method demonstrates robust performance up to a certain scale of vector insertion into the graph. Beyond this threshold, the Hierarchical Navigable Small World (HNSW) approach enhances the NSW model by introducing a multi-layered hierarchy, akin to the structure observed in Probabilistic Skip Lists. This hierarchical architecture is implemented by distributing the graph's connections across several layers, organizing them in a manner where each subsequent layer contains a subset of the links from the layer below. This stratification ensures that the top layers capture long-distance links, effectively serving as express pathways across the graph, while the lower layers focus on shorter links, facilitating fine-grained, local navigation. As a result, searches begin at the higher layers to quickly approximate the region of the target vector, progressively moving to lower layers for a more precise search, significantly improving search efficiency and accuracy by leveraging shorter links (smaller distances) between vectors as one moves from the top layer to the bottom.

To better understand how this works for HNSW, let's look at how this hierarchy is used for the Probability Skip List structure:

Figure 6-3 Probability Skip List Structure



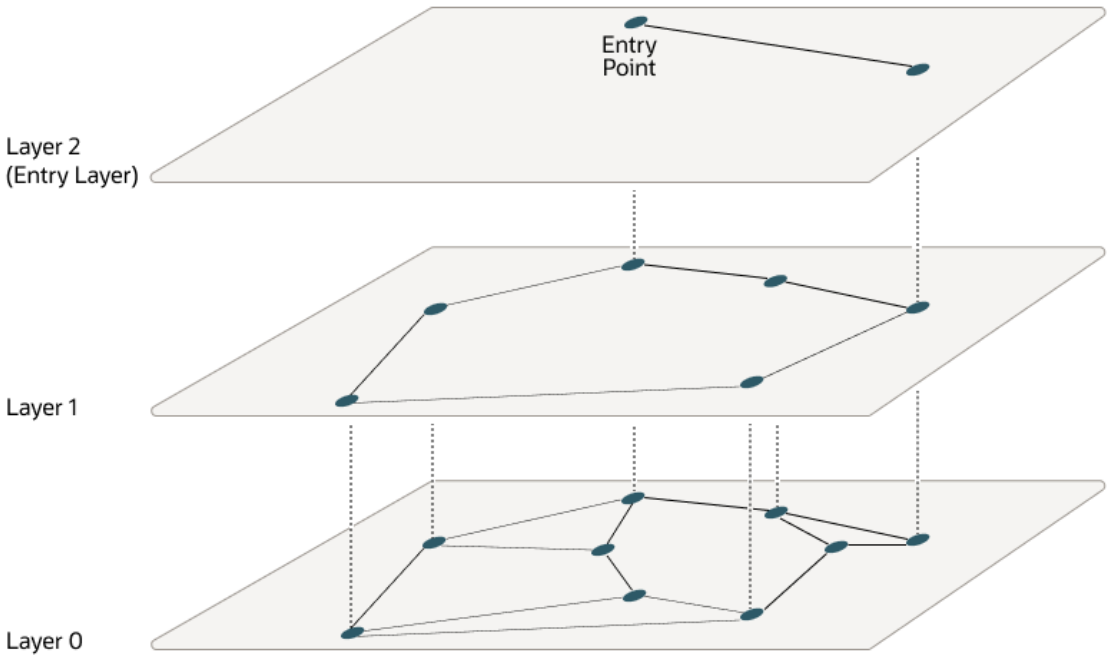
The Probability Skip List structure uses multiple layers of linked lists where the above layers are skipping more numbers than the lower ones. In this example, you are trying to search for number 17. You start with the top layer and jump to the next element until you either find 17, reach the end of the list, or you find a number that is greater than 17. When you reach the end

of a list or you find a number greater than 17, then you start in the previous layer from the latest number less than 17.

HNSW uses the same principle with NSW layers where you find greater distances between vectors in the higher layers. This is illustrated by the following diagrams in a 2D space:

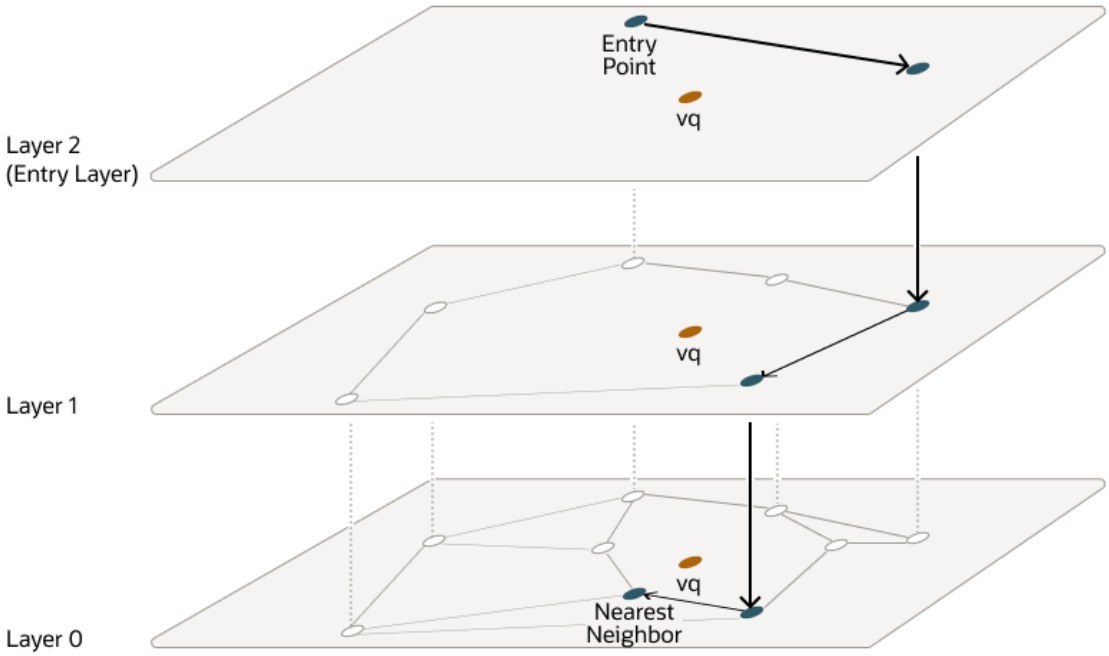
At the top layer are the longest edges and at the bottom layer are the shortest ones.

Figure 6-4 Hierarchical Navigable Small World Graphs



Starting from the top layer, the search in one layer starts at the entry vector. Then for each node, if there is a neighbor that is closer to the query vector than the current node, it jumps to that neighbor. The algorithm keeps doing this until it finds a local minimum for the query vector. When a local minimum is found in one layer, the search goes to the next layer by using the same vector in that new layer and the search continues in that layer. This process repeats itself until the local minimum of the bottom layer is found, which contains all the vectors. At this point, the search is transformed into an approximate similarity search using the NSW algorithm around that latest local minimum found to extract the top `k` most similar vectors to your query vector. While the upper layers can have a maximum of connections for each vector set by the `NEIGHBORS` parameter, layer `0` can have twice as much. This process is illustrated in the following graphic:

Figure 6-5 Hierarchical Navigable Small World Graphs Search



Layers are implemented using in-memory graphs (not Oracle Inmemory graph). Each layer uses a separate in-memory graph. As already seen, when creating an HNSW index, you can fine tune the maximum number of connections per vector in the upper layers using the `NEIGHBORS` parameter as well as the maximum number of closest vector candidates considered at each step of the search during insertion using the `EFCONSTRUCTION` parameter, where EF stands for Enter Factor.

As explained earlier, when using Oracle AI Vector Search to run an approximate search query using HNSW indexes, you have the possibility to specify a target accuracy at which the approximate search should be performed.

In the case of an HNSW approximate search, you can specify a target accuracy percentage value to influence the number of candidates considered to probe the search. This is automatically calculated by the algorithm. A value of 100 will tend to impose a similar result as an exact search, although the system may still use the index and will not perform an exact search. The optimizer may choose to still use an index as it may be faster to do so given the predicates in the query. Instead of specifying a target accuracy percentage value, you can specify the `EFSEARCH` parameter to impose a certain maximum number of candidates to be considered while probing the index. The higher that number, the higher the accuracy.

> **Note:**
>
> - If you do not specify any target accuracy in your approximate search query, then you will inherit the one set when the index was created. You will see that at index creation, you can specify a target accuracy either using a percentage value or parameters values depending on the index type you are creating.
> - It is possible to specify a different target accuracy at index search compared to the one set at index creation. For HNSW indexes, you may look at more neighbors using the `EFSEARCH` parameter (higher than the `EFCONSTRUCTION` value specified at index creation) to get more accurate results. The target accuracy that you give during index creation decides the index creation parameters and also acts as the default accuracy values for vector index searches.

Understand Transaction Support for Tables with HNSW Indexes

Hierarchical Navigable Small World (HNSW) index graphs are static memory-only structures. Transaction maintenance on tables with HNSW indexes is done by using two main structures: private journal and shared journal.

A **private journal** is a per-transaction in-memory data structure that tracks vectors, added or deleted by a transaction (updates are in fact a delete followed by an insert). This is comparable to transaction journals that are used to maintain the in-memory column store data (explained in *Oracle Database In-Memory Guide*). These memory structures come out of the Vector Memory Pool and are used for read consistency purposes.

A **shared journal** contains the commit system change numbers (SCNs) and corresponding modified rows. This structure is an on-disk structure created at the time of vector index creation. At commit time, the changes recorded into a private journal are converted to rows and flushed into the shared journal. This structure is also used for read consistency purposes.

> **Note:**
>
> - For bulk DMLs (direct load using `INSERT /*+ APPEND */`), changes are tracked directly in the shared journal to avoid pressure on the vector memory pool.
> - At the time of full repopulation (until a new HNSW graph becomes available), if a query tries to access an older version of the HNSW graph that no longer exists, then the read consistency error `ORA 51815 "INMEMORY NEIGHBOR GRAPH HNSW vector index snapshot is too old."` is triggered.
>
>   For example, assume that an old HNSW graph exists at SCN 100. While a full repopulation builds a new graph at SCN 200 (which will create a new ROWID-to-VID mapping table), a query arriving at SCN 150 cannot access the new graph at SCN 200. This is because the query's execution plan is compiled with the old ROWID-to-VID mapping table corresponding to the old graph at SCN 100.

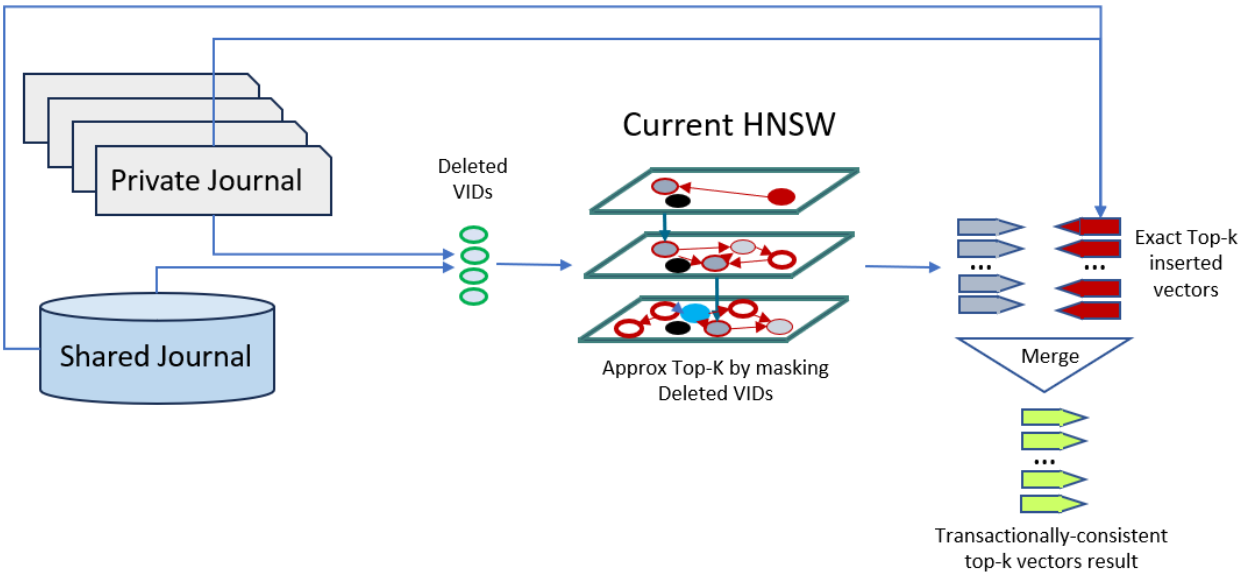
In addition to the previously defined structures used mostly for transaction consistency, a full checkpoint on-disk structure can also be maintained, if enabled, for faster reload of HNSW indexes after instance restart. A full checkpoint is automatically created upon index creation and full repopulation of the HNSW graph. HNSW checkpointing is enabled by default. To

disable or re-enable checkpoints, see Vector Index Status, Checkpoint, and Advisor Procedures.

Let us see how the previously defined structures are used in conjunction of each other to achieve overall better performance for transaction maintenance and read consistency on tables with HNSW indexes:

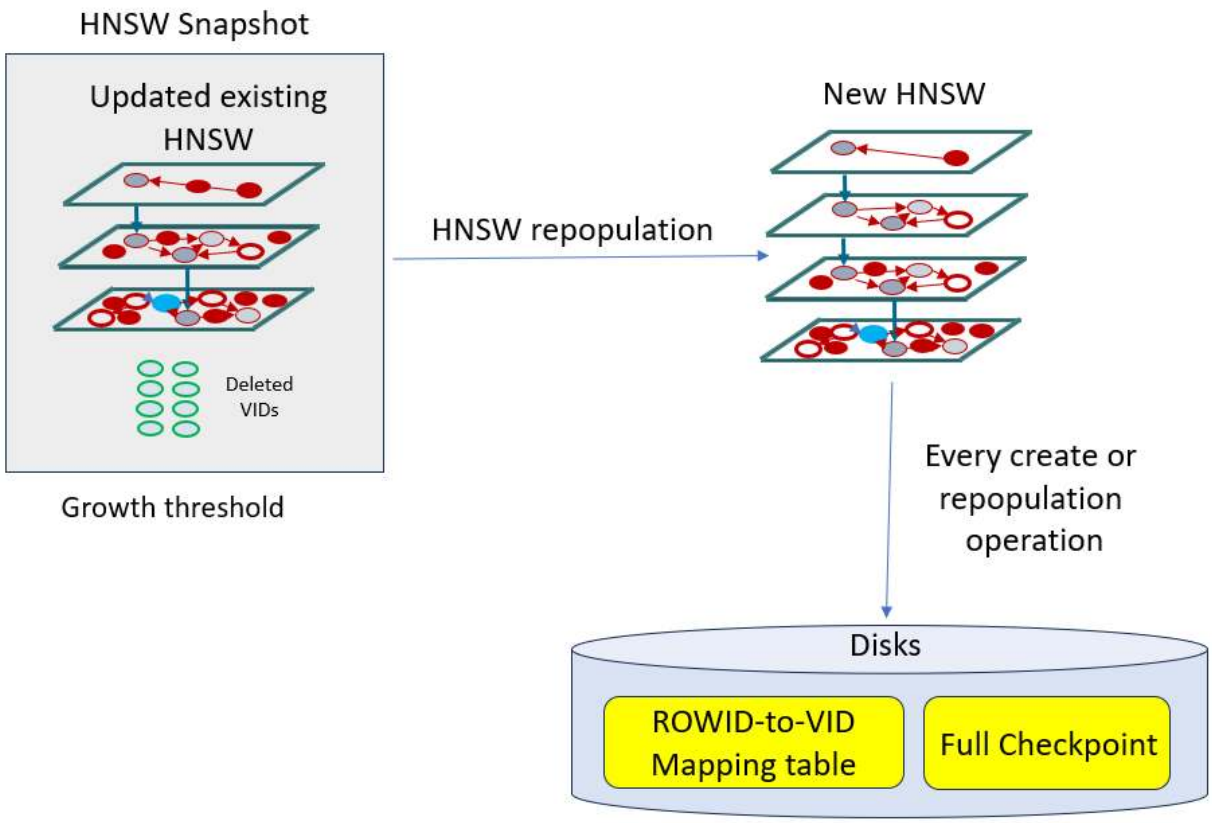
1. Following DMLs, read consistency is achieved by taking into account both the existing HNSW graph in memory as well as the query's private journal and shared journal to determine the list of transactionally-consistent set of deleted and inserted vectors. This consists of identifying exact lists of deleted vectors from the journals, running an approximate top-K search through the current version of the HNSW index by augmenting the filter to ignore deleted vectors, running an exact top-k search of newly inserted vectors in the journals, and merging the results of the two searches.

   This is illustrated by the following diagram:



2. As DMLs accumulate in the shared journal, exact searches for deleted and inserted vectors in the on-disk journal see their performance deteriorating. To minimize this impact, a full repopulation of the index is automatically triggered. The decision to repopulate the HNSW index in the background is based on a default threshold representing a certain percentage of the number of DMLs run against the index. In addition, each time the HNSW index is created or repopulated, a full checkpoint of the newly created graph and the new ROWID-to-VID mapping table are recreated on disks.

   This is illustrated by the following diagram:



> **Note:**
>
> During full repopulation, two copies of the graph are needed in memory. For example, if we have 10% new inserts, then the vector memory requirement for the repopulated graph will be approximately 10% more than the original HNSW graph size in memory. So at peak vector memory requirement during repopulation, the memory need will be 2.1 times before going down to 1.1 times after repopulation is finalized. This is done to ensure that read consistency can still be done using the previous version of the graph while the new one is being created.

Understand HNSW Index Population Mechanisms in Oracle RAC or Single Instance

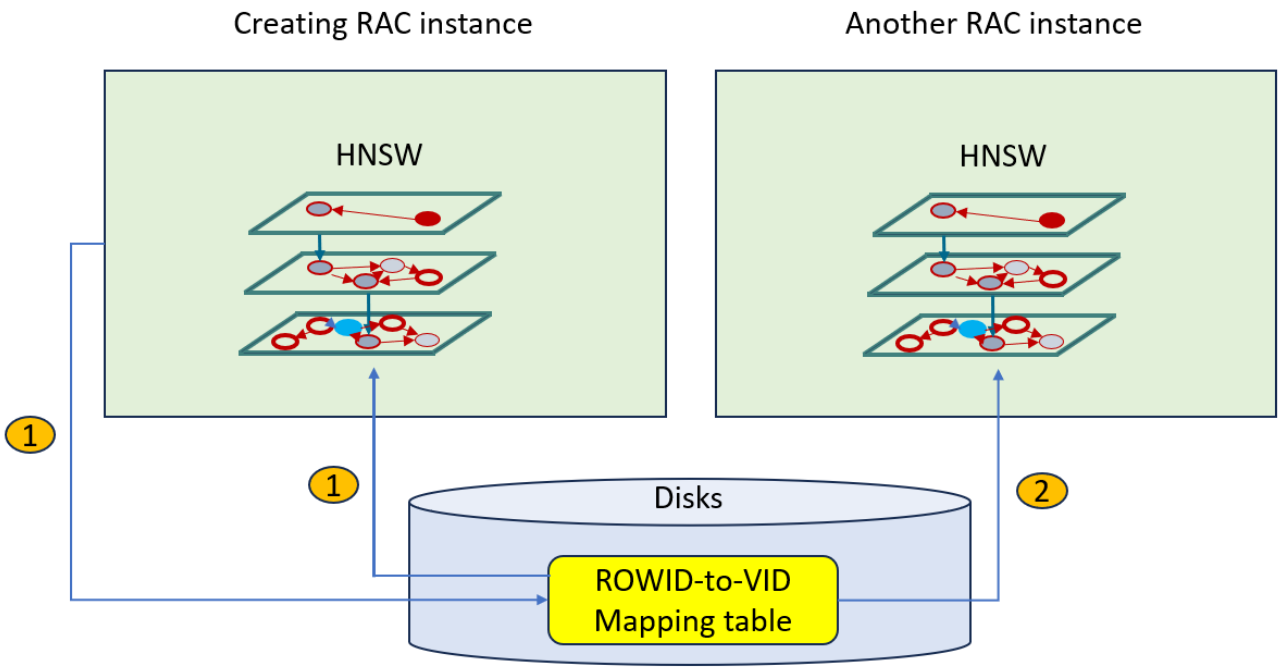
Learn how Hierarchical Navigable Small World (HNSW) indexes are populated during index creation, index repopulation, or instance startup in an Oracle Real Application Clusters (Oracle RAC) or a non-RAC environment.

HNSW Index Creation and Repopulation

The following diagram summarizes what happens when you create the HNSW index in an Oracle RAC environment:

1. The RAC instance that creates the HNSW index is responsible for creating the HNSW ROWID-to-VID mapping table on the disk. As explained in Optimizer Plans for HNSW Vector Indexes, this table is needed by the optimizer to run certain optimization plans.
2. By default, once the first HNSW graph is created, all other RAC instances are informed to start their own HNSW In-Memory Graph creation concurrently. This operation is called an

HNSW duplication operation. The duplication mechanism is using the HNSW ROWID-to-VID mapping table on disk to avoid many recalculations and benefit from the existence of the HNSW ROWID-to-VID mapping table.



> **Note:**
>
> Instances that do not have enough vector memory cannot participate in the parallel RAC-wide HNSW index population.

> **Note:**
>
> Although each RAC instance shares the same ROWID-to-VID mapping table, each instance may end up with a different HNSW In-Memory Graph. Therefore, you may get different results depending on which RAC instance the query lands on.

In addition to this initial index creation case, the same duplication mechanism is also used when the HNSW index needs to be fully repopulated. See Understand Transaction Support for Tables with HNSW Indexes for more information about why and when HNSW index full repopulation operation is triggered.

HNSW Full Checkpoints

A **full checkpoint** is a serialized version of the HNSW graph, stored on disk and containing all the vertices and edges of the HNSW multi-layered graph. A full checkpoint is self-contained and has roughly the same footprint as the corresponding HNSW in-memory graph. As explained in Understand Transaction Support for Tables with HNSW Indexes, a full checkpoint is created at both the index creation time and repopulation operation.

HNSW full checkpoints are used to reduce the HNSW graph creation time when a new instance joins an Oracle RAC cluster or when an instance is restarted. The main advantage of using the full checkpoint over using the ROWID-to-VID mapping table or creating a new graph is that the neighbors for a particular vector have already been computed and persisted in the full checkpoint.

> **Note:**
>
> Although an HNSW full checkpoint might not be completely up-to-date and missing some transactions (because it is not maintained for every DML), it will eventually catch up while read consistency is still ensured by reading the missing vectors from the on-disk shared journal. The shared journal is also automatically created by the HNSW index creating instance for transaction support on HNSW indexes. For more information on the shared journal, see Understand Transaction Support for Tables with HNSW Indexes.

When you create an HNSW index, the full checkpoint creation and maintenance is enabled by default.

> **Note:**
>
> The HNSW full checkpoint can only be maintained provided there is adequate space in the user's tablespace.

You can disable or re-enable full HNSW checkpoints by using the `DBMS_VECTOR` package:

- **Disable** means drop existing full checkpoint for a particular index and do not create new full checkpoints:

  ```
  DBMS_VECTOR.DISABLE_CHECKPOINT(<schema owning indexes> [, <index name>])
  ```

- **Enable** (default) means the next HNSW graph repopulation will create a full checkpoint for a particular HNSW index:

  ```
  DBMS_VECTOR.ENABLE_CHECKPOINT(<schema owning indexes> [, <index name>] [,
  <tablespace name>])
  ```

For more information, see the `ENABLE_CHECKPOINT` and `DISABLE_CHECKPOINT` procedures in Vector Index Status, Checkpoint, and Advisor Procedures.

You can query the catalog table `VECSYS.VECTOR$INDEX$CHECKPOINTS` to track information about full checkpoints at the database level. See VECSYS.VECTOR$INDEX$CHECKPOINTS.

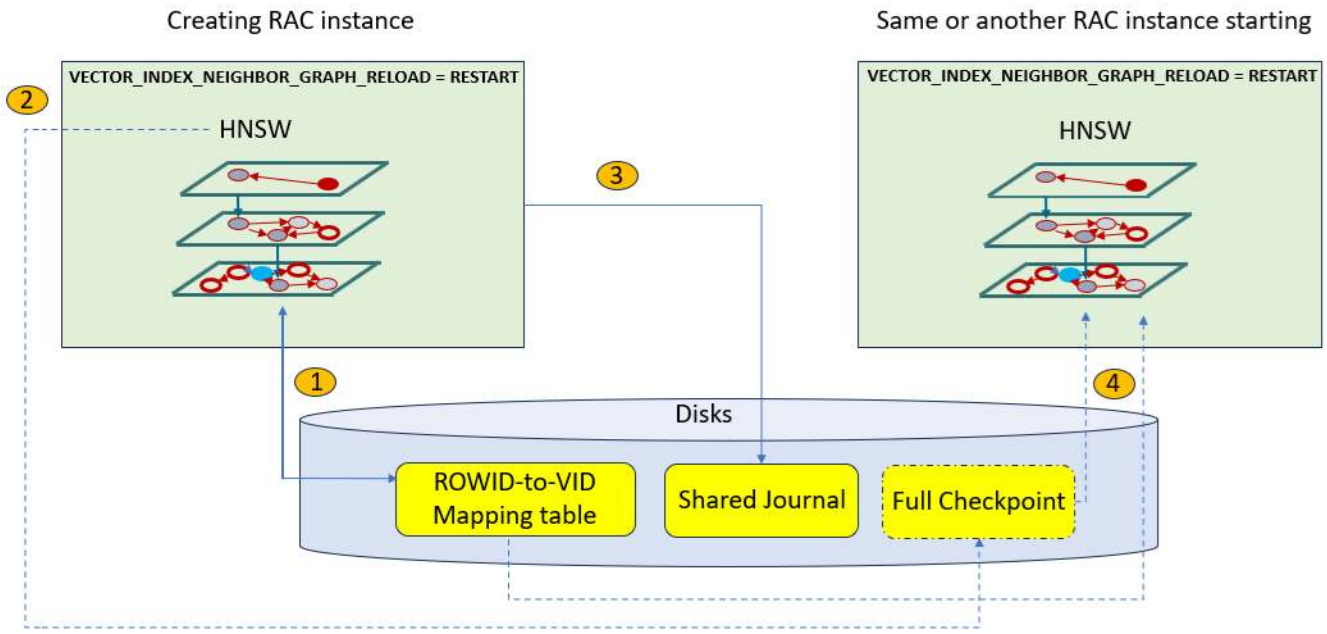
HNSW Index Reload at Instance Restart or New Node Joining Cluster

Because HNSW indexes are created in memory, if the Oracle Database instance goes down, then you lose the corresponding in-memory graphs representing your HNSW indexes. By default, when the instance starts again, a **reload** mechanism is triggered to recreate the HNSW graph in memory as quickly as possible. This reload mechanism is enabled by default for both Oracle RAC and non-RAC environments.

The way the reload mechanism is processed depends on the existence, or not, of a full HNSW graph checkpoint on disk.

The following diagram summarizes reload mechanism in a RAC environment:

1. At index creation time, the in-memory HNSW graph and the ROWID-to-VID mapping table on disks are created.
2. If enabled, a full checkpoint is also created on disks.
3. The shared journal structure is also created and used to handle transaction consistency for HNSW indexes.
4. If the `VECTOR_INDEX_NEIGHBOR_GRAPH_RELOAD` instance parameter is set to `RESTART` (default setting) at the time the instance joins the cluster or when restarting, and if a full checkpoint exists and is not deemed too old compared to the current instance SCN, then it is used by the starting instance to create its HNSW graph in memory. If these two conditions are not met, then the starting instance uses the duplication mechanism to create the HNSW graph in memory from scratch. If the `VECTOR_INDEX_NEIGHBOR_GRAPH_RELOAD` instance parameter is set to `OFF` at the time the instance joins the cluster or when restarting, then the HNSW graph is not reloaded.



> **Note:**
>
> A full checkpoint is used to reload the HNSW graph in memory for an instance if its creation SCN is not too old as compared to the current instance's SCN. If the SCN is too old, then that instance does a full repopulation of the index using the duplication mechanism (as previously described).

> **Note:**
>
> The `VECTOR_INDEX_NEIGHBOR_GRAPH_RELOAD` initialization parameter value governs the automatic reload or not of the HNSW indexes in-memory graphs.

Hierarchical Navigable Small World Index Syntax and Parameters

Review the syntax and examples for Hierarchical Navigable Small World vector indexes.

Syntax

```
CREATE VECTOR INDEX vector_index_name
ON table_name (vector_column)
[GLOBAL] ORGANIZATION INMEMORY [NEIGHBOR] GRAPH
[WITH] [DISTANCE [CUSTOM [<schema>.][<package>.]] metric name]
[WITH TARGET ACCURACY percentage_value]
[PARAMETERS (TYPE HNSW , {NEIGHBORS max_closest_vectors_connected
    | M max_closest_vectors_connected}
    , EFCONSTRUCTION max_candidates_to_consider)]
[PARALLEL degree_of_parallelism]
```

HNSW Parameters

`NEIGHBORS` and `M` are equivalent and represent the maximum number of neighbors a vector can have on any layer. The last vertex has one additional flexibility that it can have up to 2M neighbors.

`EFCONSTRUCTION` represents the maximum number of closest vector candidates considered at each step of the search during insertion.

The valid range for HNSW vector index parameters are:

- `ACCURACY`: > 0 and <= 100
- `DISTANCE`: `EUCLIDEAN`, `L2_SQUARED` (aka `EUCLIDEAN_SQUARED`), `COSINE`, `DOT`, `MANHATTAN`, `HAMMING`, `CUSTOM <custom metric name>`
- `TYPE`: `HNSW`
- `NEIGHBORS`: > 0 and <= 2048
- `EFCONSTRUCTION`: > 0 and <= 65535

Examples

```
CREATE VECTOR INDEX galaxies_hnsw_idx ON galaxies (embedding) ORGANIZATION
INMEMORY NEIGHBOR GRAPH
DISTANCE COSINE
WITH TARGET ACCURACY 95;

CREATE VECTOR INDEX galaxies_hnsw_idx ON galaxies (embedding) ORGANIZATION
INMEMORY NEIGHBOR GRAPH
DISTANCE COSINE
WITH TARGET ACCURACY 90 PARAMETERS (type HNSW, neighbors 40, efconstruction
500);
```

For detailed information, see CREATE VECTOR INDEX in *Oracle Database SQL Language Reference* .

Neighbor Partition Vector Index

Inverted File Flat (IVF) index is the only type of Neighbor Partition vector index supported. Inverted File Flat Index (IVF Flat or simply IVF) is a partitioned-based index that lets you balance high-search quality with reasonable speed.

- About Neighbor Partition Vector Index
  The default type of index created for a Neighbor Partition vector index is Inverted File Flat.
- Included Columns
  Included columns in vector indexes facilitate faster searches with attribute filters by incorporating non-vector columns within a Neighbor Partition Vector Index.
- Inverted File Flat Index Syntax and Parameters
  Review the syntax and examples for Inverted File Flat vector indexes.

About Neighbor Partition Vector Index

The default type of index created for a Neighbor Partition vector index is Inverted File Flat.

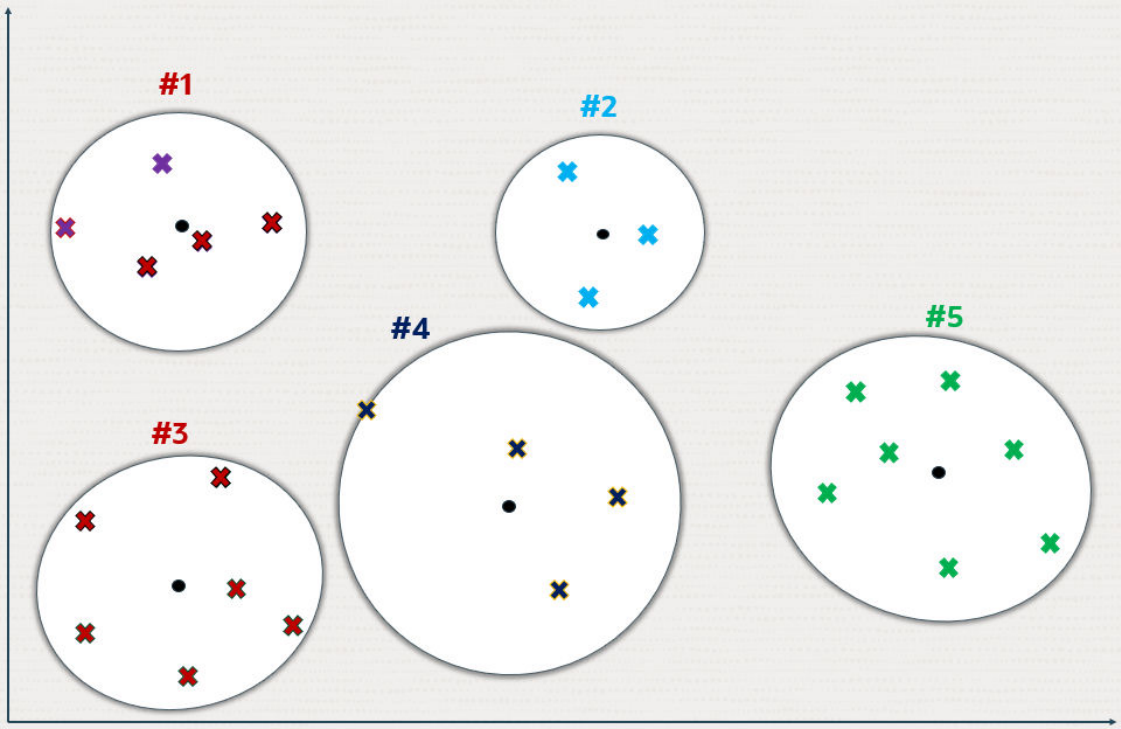
- Understand Inverted File Flat Vector Indexes
  The Inverted File Flat vector index is a technique designed to enhance search efficiency by narrowing the search area through the use of neighbor partitions or clusters.
- Inverted File Flat Vector Indexes Partitioning Schemes
  Inverted File Flat vector indexes support both global and local indexes on partitioned tables. By default, IVF indexes are globally partitioned by centroid.

Understand Inverted File Flat Vector Indexes

The Inverted File Flat vector index is a technique designed to enhance search efficiency by narrowing the search area through the use of neighbor partitions or clusters.

The following diagrams depict how partitions or clusters are created in an approximate search done using a 2D space representation. But this can be generalized to much higher dimensional spaces.

Figure 6-6 Inverted File Flat Index Using 2D



Crosses represent the vector data points in this space.

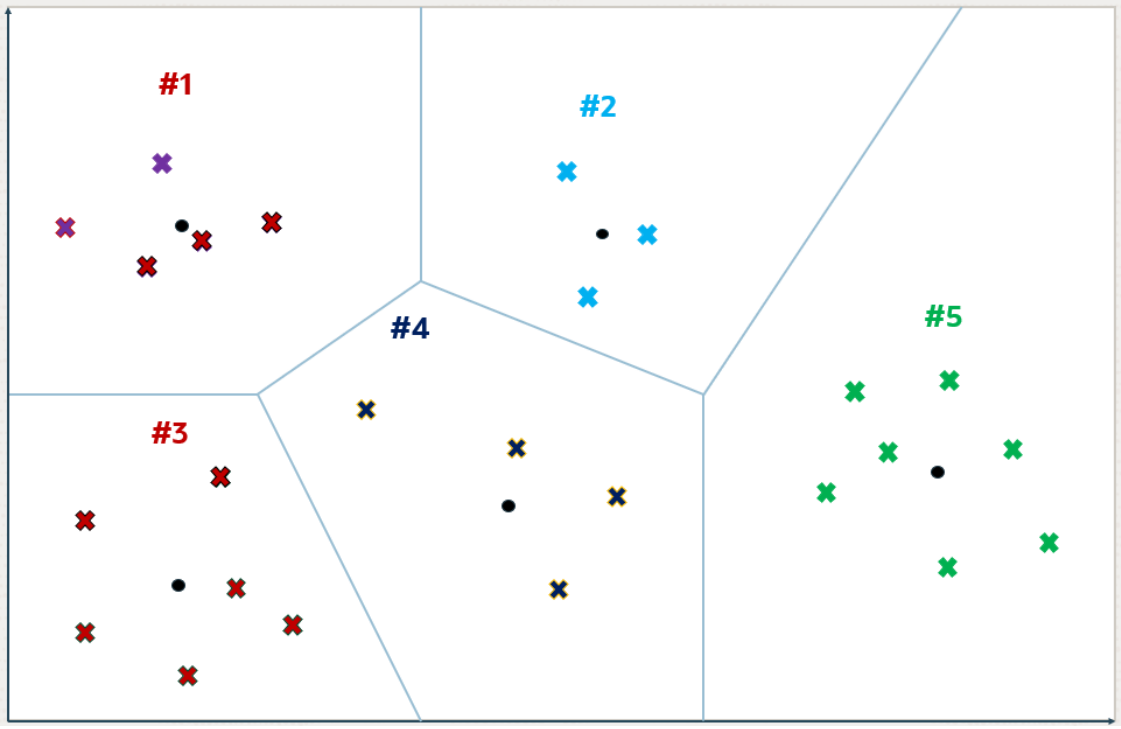
New data points, shown as small plain circles, are added to identify `k` partition centroids, where the number of centroids (`k`) is determined by the size of the dataset (`n`). Typically `k` is set to the square root of `n`, though it can be adjusted by specifying the `NEIGHBOR PARTITIONS` parameter during index creation.

Each centroid represents the average vector (center of gravity) of the corresponding partition.

The centroids are calculated by a training pass over the vectors whose goal is to minimize the total distance of each vector from the closest centroid.

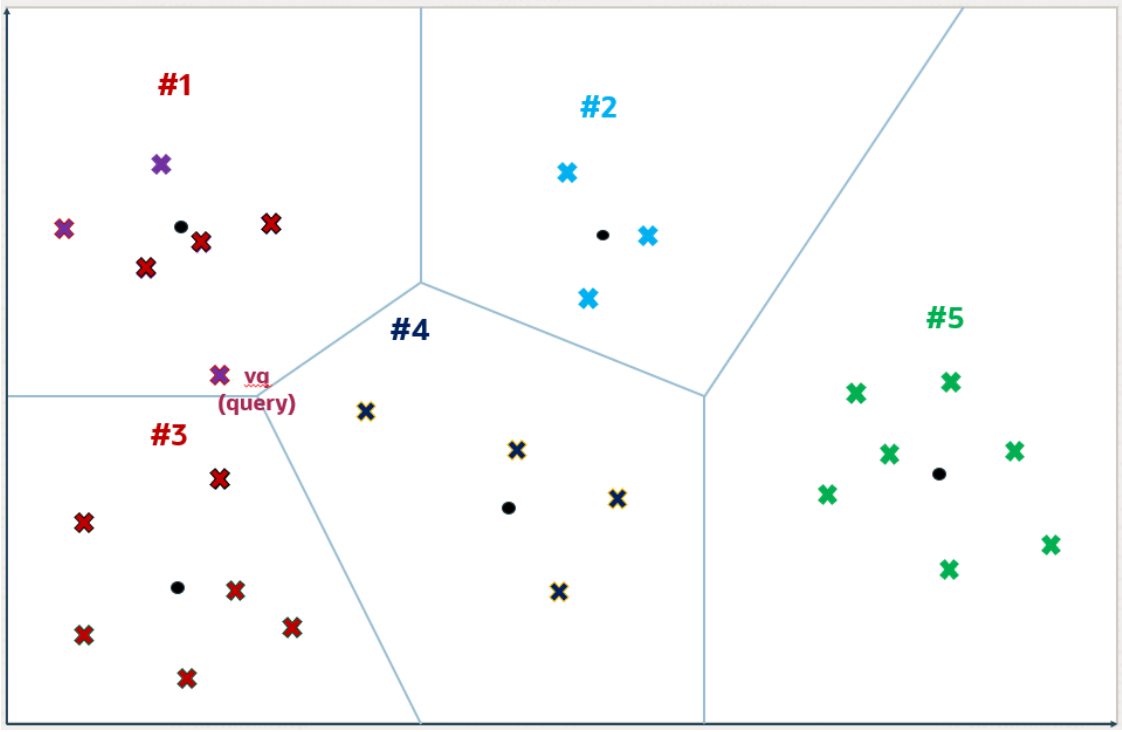
The centroids ends up partitioning the vector space into `k` partitions. This division is conceptually illustrated as expanding circles from the centroids that stop growing as they meet, forming distinct partitions.

Figure 6-7 Inverted File Flat Index



Except for those on the periphery, each vector falls within a specific partition associated with a centroid.

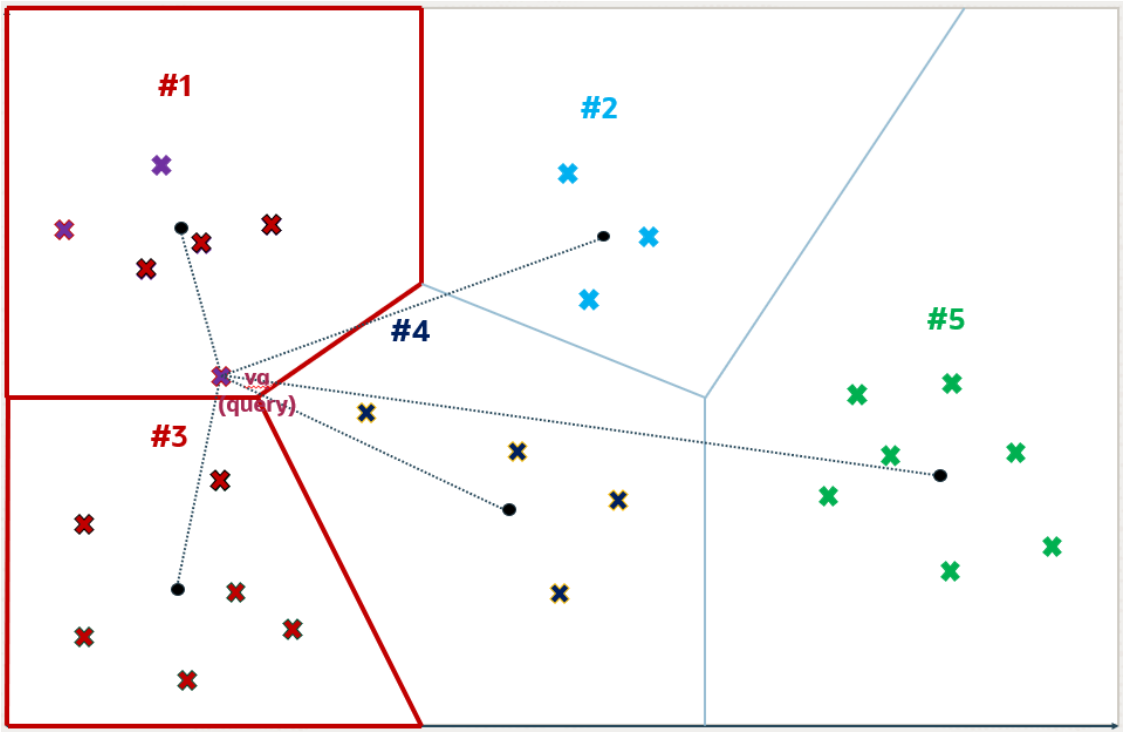
Figure 6-8 Inverted File Flat Index



For a query vector `vq`, the search algorithm identifies the nearest `i` centroids, where `i` defaults to the square root of `k` but can be adjusted for a specific query by setting the `NEIGHBOR PARTITION PROBES` parameter. This adjustment allows for a trade-off between search speed and accuracy.

Higher numbers for this parameter will result in higher accuracy. In this example, `i` is set to 2 and the two identified partitions are partitions number 1 and 3.

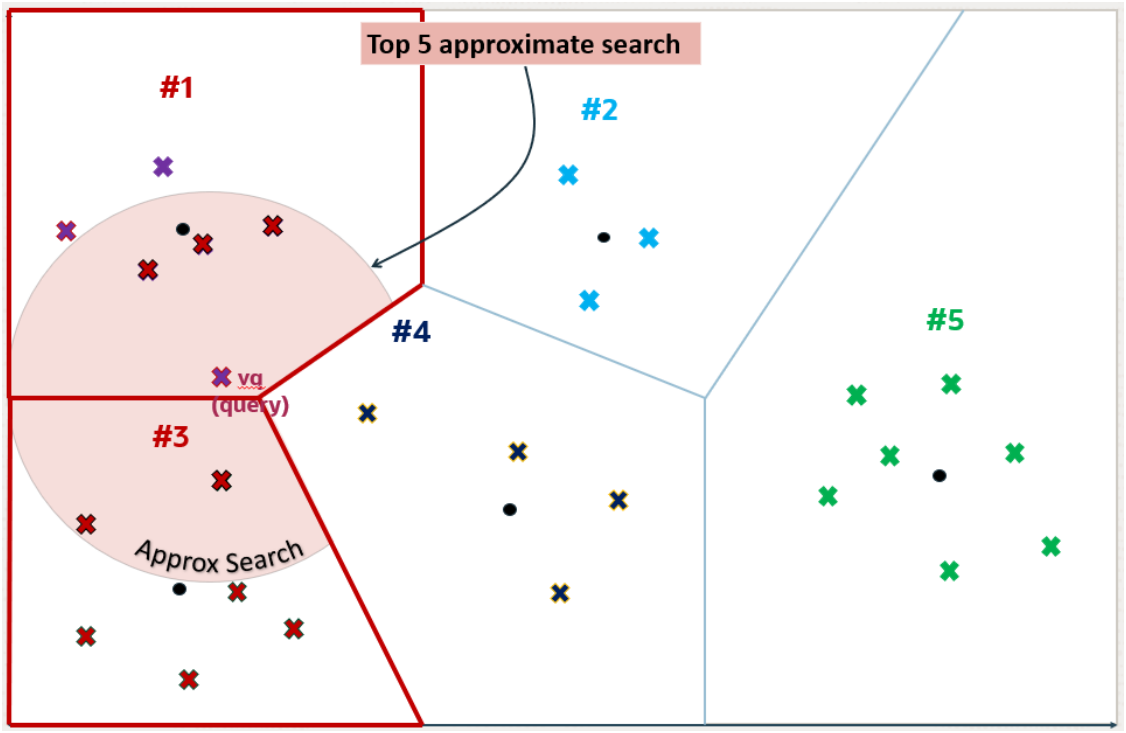
Figure 6-9 Inverted File Flat Index



Once the `i` partitions are determined, they are fully scanned to identify, in this example, the top 5 nearest vectors. This number 5 can be different from `k` and you specify this number in your query. The five nearest vectors to `vq` found in partitions number 1 and 3 are highlighted in the following diagram.

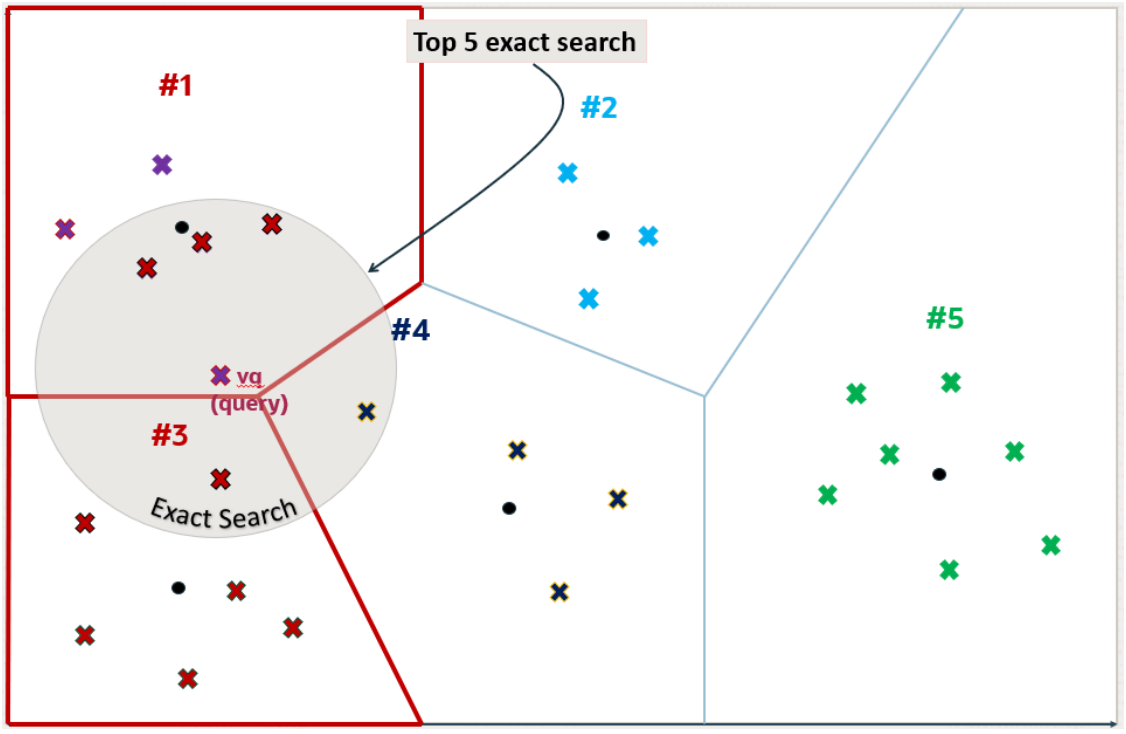
This method constitutes an approximate search as it limits the search to a subset of partitions, thereby accelerating the process but potentially missing closer vectors in unexamined partitions. This example illustrates that an approximate search might not yield the exact nearest vectors to `vq`, demonstrating the inherent trade-off between search efficiency and accuracy.

Figure 6-10 Inverted File Flat Index



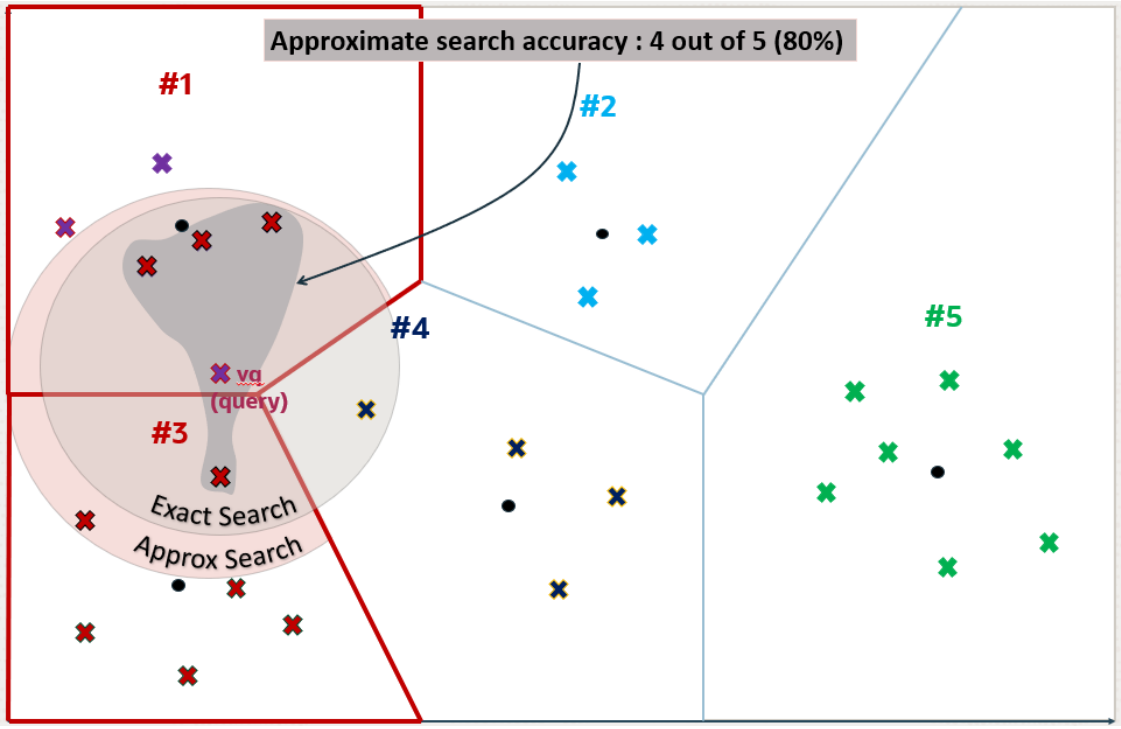
However, the five exact nearest vectors from `vq` are not the ones found by the approximate search. You can see that one of the vectors in partition number 4 is closer to `vq` than one of the retrieved vectors in partition number 3.

Figure 6-11 Inverted File Flat Index



You can now see why using vector index searches is not always an exact search and is called an approximate search instead. In this example, the approximate search accuracy is only 80% as it has retrieved only 4 out of 5 of the exact search vectors' result.

Figure 6-12 Inverted File Flat Index



When using Oracle AI Vector Search to run an approximate search query using vector indexes, you have the possibility to specify a target accuracy at which the approximate search should be performed.

In the case of an IVF approximate search, you can specify a target accuracy percentage value to influence the number of partitions used to probe the search. This is automatically calculated by the algorithm. A value of 100 will tend to impose an exact search, although the system may still use the index and will not perform an exact search. The optimizer may choose to still use an index as it may be faster to do so given the predicates in the query. Instead of specifying a target accuracy percentage value, you can specify the `NEIGHBOR PARTITION PROBES` parameter to impose a certain maximum number of partitions to be probed by the search. The higher that number, the higher the accuracy.

> **Note:**
>
> - If you do not specify any target accuracy in your approximate search query, then you will inherit the one set when the index was created. You will see that at index creation time, you can specify a target accuracy either using a percentage value or parameters values depending on the type of index you are creating.
> - It is possible to specify a different target accuracy at index search, compared to the one set at index creation. For IVF indexes, you may probe more centroid partitions using the `NEIGHBOR PARTITION PROBES` parameter to get more accurate results. The target accuracy that you provide during index creation decides the index creation parameters and also acts as the default accuracy value for vector index searches.

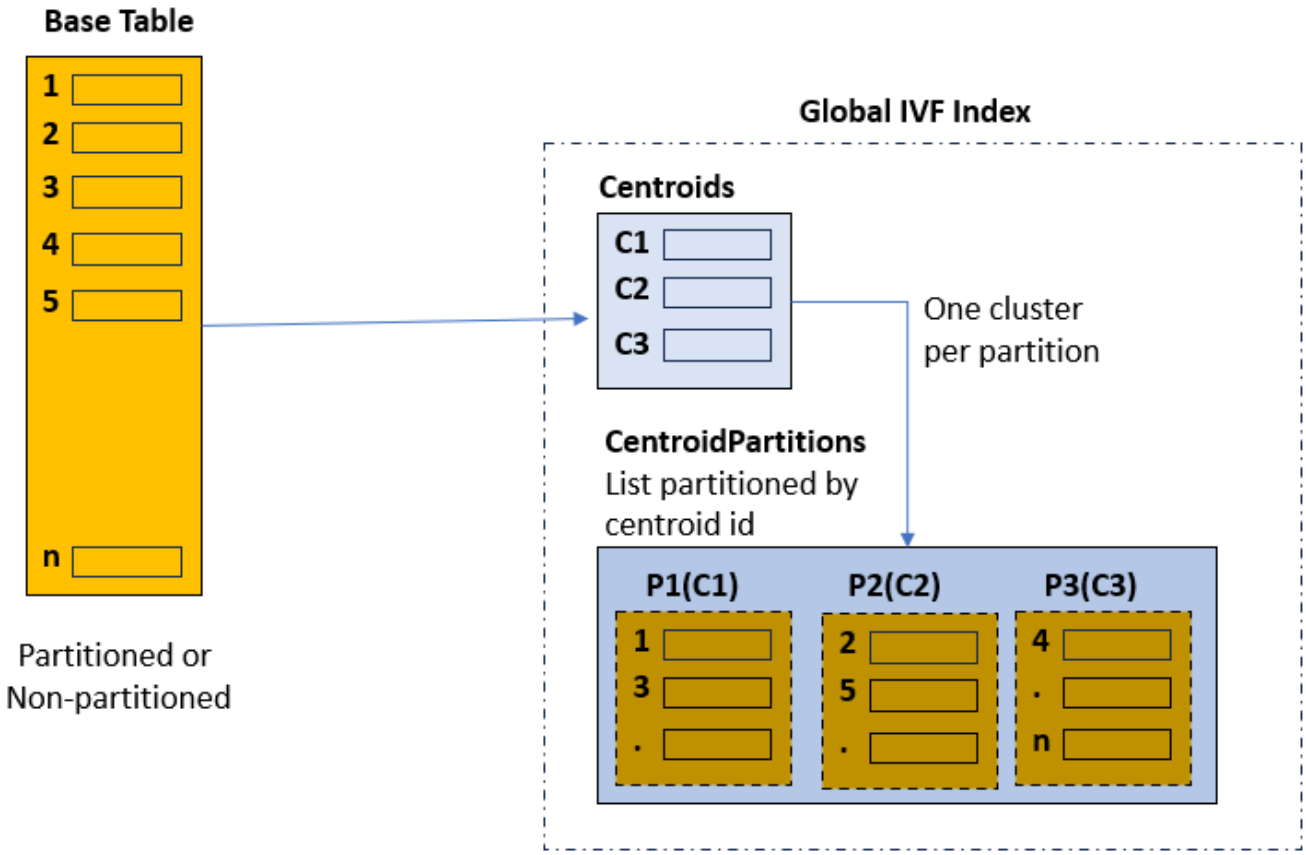
Inverted File Flat Vector Indexes Partitioning Schemes

Inverted File Flat vector indexes support both global and local indexes on partitioned tables. By default, IVF indexes are globally partitioned by centroid.

A **global IVF index** is composed of two tables:

- One called `VECTOR$<base table name>_IVF_IDX$<object info>$IVF_FLAT_CENTROIDS`, containing the list of identified centroid vectors and associated ids.
- The second called `VECTOR$<base table name>_IVF_IDX$<object info>$IVF_FLAT_CENTROID_PARTITIONS`, which is list-partitioned on the centroid ids. Each partition contains the base table vectors closely related (cluster) to the corresponding centroid id for that partition.

This is illustrated by the following diagram:



This structure is used to accelerate searches in the index by identifying first the centroid that is the closest to your query vector, and then use the corresponding centroid id to prune unnecessary partitions.

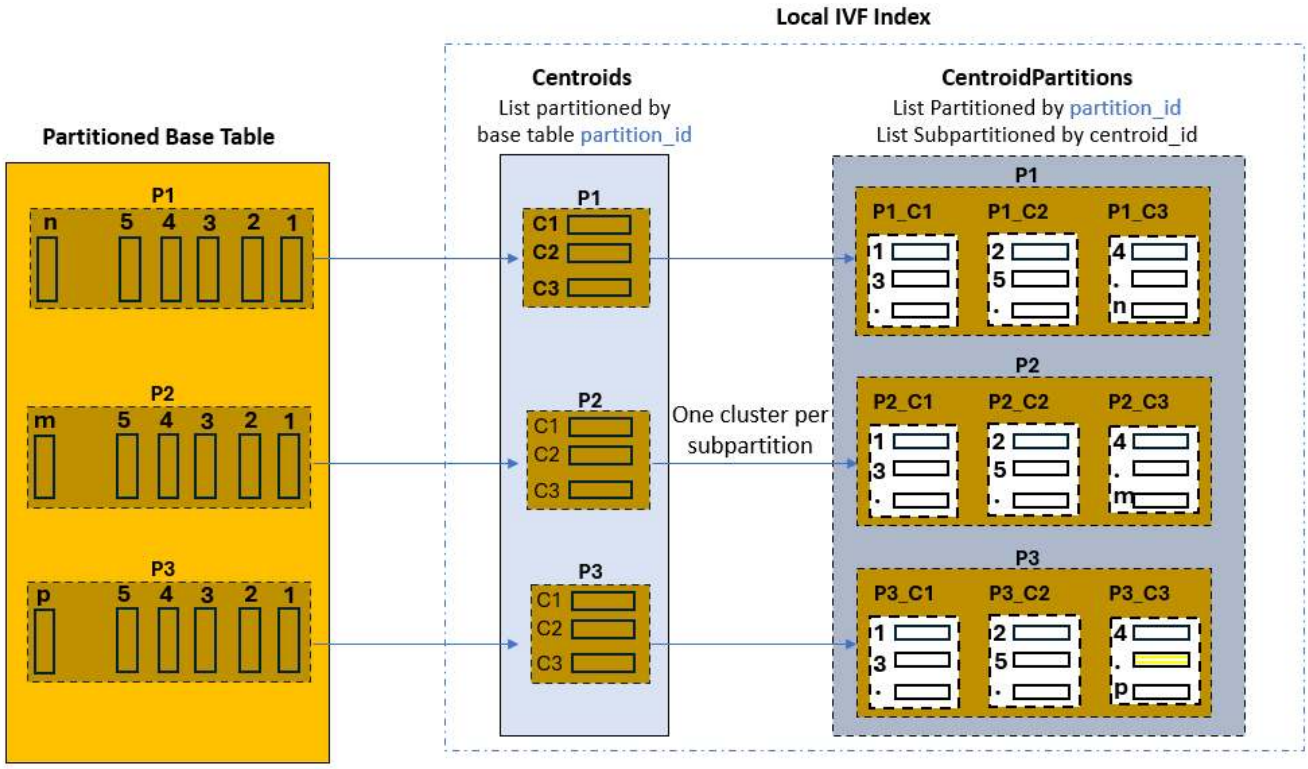
However, if the base table is partitioned on some relational data and your query is filtering on the base table partition key, then global IVF indexes are not optimal because they are completely independent of the base table partition key. For example, if you search the top-10 houses in California similar to a vectorized picture, the picture itself has most probably no relationship with the state of California. While your query benefits from the fact that your base table is partitioned by state, so you can search only the partition corresponding to California, the query still must look at pictures that may not be in California.

To further accelerate such type of queries, you have the possibility to create a **local IVF index**. The term **local** for an index refers to a one-to-one relationship between the base table partitions or subpartitions and the index partitions. The local IVF index creation and DML operations on the base tables with IVF indexes could be accelerated if **Vector Pool** is enabled. **Vector Pool** is a new memory area stored in the SGA. For more information related to **Vector Pool**, read this topic: Size the Vector Pool.

This is illustrated by the following graphic, where the base table has three partitions. The created local IVF index is still constituted by two internal tables:

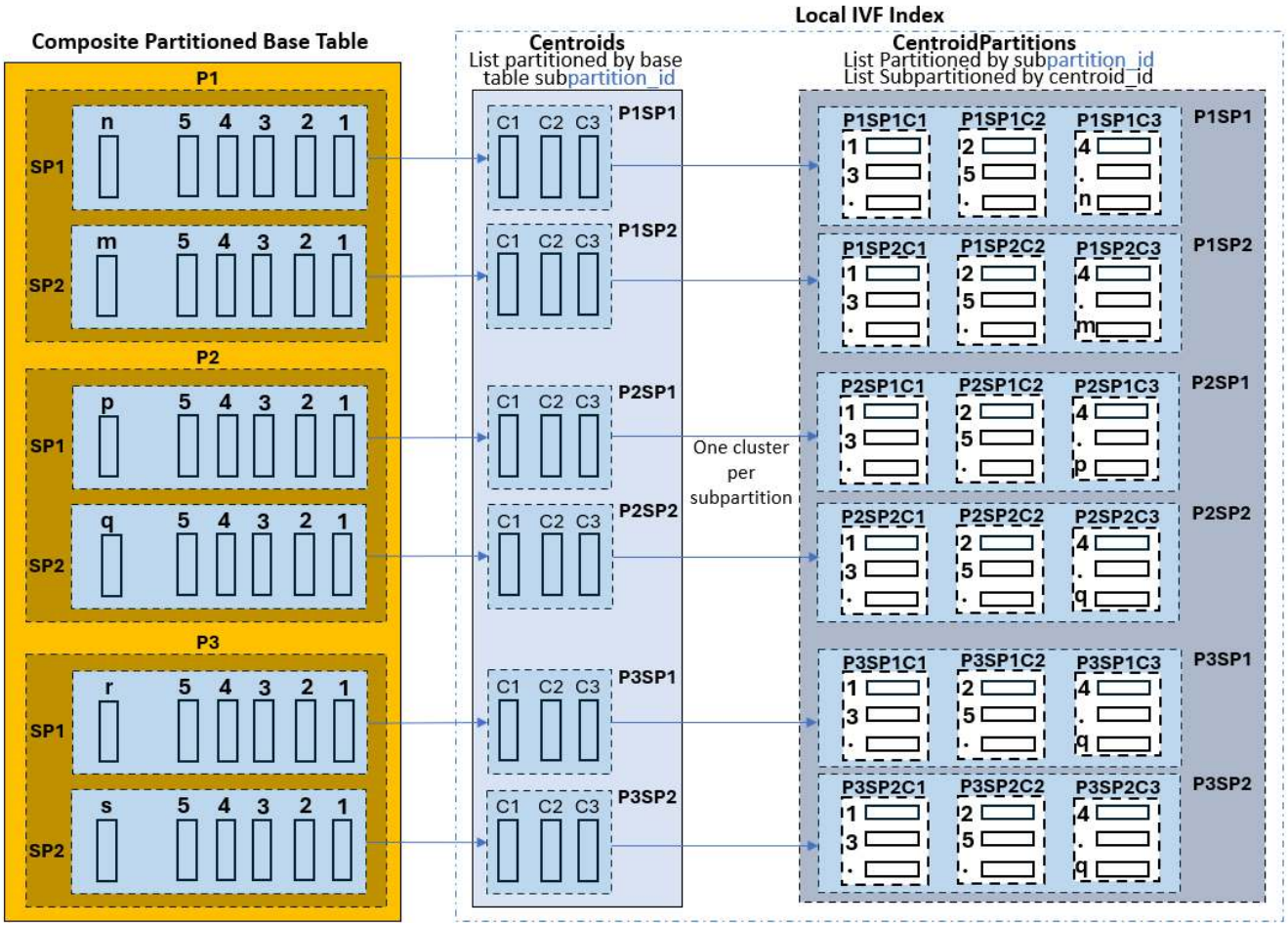
- One called `VECTOR$<base table name>_IVF_IDX$<object info>$IVF_FLAT_CENTROIDS`, which is list-partitioned by base table partition ids, and is thus equi-partioned with the base table. Each partition containing the list of corresponding identified centroid vectors and associated ids.

- The second called `VECTOR$<base table name>_IVF_IDX$<object info>$IVF_FLAT_CENTROID_PARTITIONS`, which is list-partitioned by base table partition id and list-subpartitioned by centroid id. This table is also equi-partitioned with the base table, and each subpartition contains the base table vectors closely related (cluster) to the corresponding centroid id for that subpartition.



Coming back to our initial example where you search the top-10 houses in California similar to a vectorized picture; your query benefits from partition pruning on the base table and Centroids table (California) as they are both partitioned by state. In addition, and once the closest centroid is identified in that partition, the query simply needs to scan the corresponding centroid cluster subpartition in the Centroid Partitions table without having to scan other centroid subpartitions.

Another possibility is for the base table to be composite partitioned. Here is a graphical representation corresponding to that case. The Centroids table is list-partitioned according to base table subpartitions. Each partition in the Centroids table containing all centroid vectors found in the corresponding base table subpartition. The Centroid Partitions table is list-partitioned by base table subpartition id, and is further subpartitioned by centroid id:



> **Note:**
>
> You can create local IVF indexes only on a partitioned base table.
>
> Local IVF indexes inherit all system catalog tables and views used by regular local indexes. A flag (`idx_spare2`) in the `vecsys.vector$index` table indicates if an index is a local or global vector index.

Using local IVF indexes brings additional advantages:

- Simplified partition management operations (PMOP):

  For example, dropping a table partition just involves dropping the corresponding index partition.

- Flexible Indexing schemes:

  For example, marking certain index partitions `UNUSABLE` to avoid indexing certain table partitions through partial indexing.

> **Note:**
>
> If you want your user queries to use full potential of local IVF indexes, by taking the benefit of partition pruning, then user queries must satisfy the following conditions:
>
> - The base table is [sub]partitioned by a single column.
> - Conditions are a form of `[sub]partitition_key CMP` constant, where `CMP` can be:
>
>   `=, >, >=, <, <=, IN`
>
> - The `partition_key` condition is `AND`ed with other non-partition conditions.

Partition Management Operations (PMOP) and IVF Indexes

These are the PMOP possibilities and restrictions for local IVF indexes:

- `ALTER TABLE TRUNCATE [sub]partition <partition_name>`

  `ALTER TABLE DROP [sub]partition <partition_name>`

  These operations are supported with all partition schemes: `RANGE`, `HASH`, `LIST`. However, all corresponding IVF index partitions are marked as `UNUSABLE` after the operation.

- `ALTER TABLE ADD [sub]partition <partition_name>`

  If the base table is partitioned by `RANGE` or `LIST`, then the operation is supported. However, all corresponding IVF indexes are marked as `UNUSABLE` after the operation. If the table is partitioned by `HASH`, then the operation fails if there are any local IVF indexes on the base table.

- All other PMOP operations on the base table are not supported, such as `ALTER TABLE SPLIT/MERGE/MOVE/EXCHANGE/COALESCE`.

  These `ALTER TABLE` statements will fail if there are any local IVF indexes on the table being altered.

- `ALTER INDEX` on local IVF indexes is not supported.

Experiment with LOCAL IVF Vector Indexes

You can start experimenting with `LOCAL` IVF indexes using the following code. This is not a scenario but merely a series of SQL commands to help you get started on your own testing scenarios.

1. **Create the base table using the partition scheme of your choice:**
   `RANGE` partitioning:

```
DROP TABLE sales_data PURGE;


CREATE TABLE sales_data
(
    product_id NUMBER,
    customer_id NUMBER,
    sale_date DATE,
    amount_sold NUMBER,
    vec vector(8),
```

```
    region VARCHAR2(20)
)
PARTITION BY RANGE (product_id)
(
    PARTITION sales_1 VALUES LESS THAN (100),
    PARTITION sales_2 VALUES LESS THAN (200),
    PARTITION sales_3 VALUES LESS THAN (300),
    PARTITION sales_4 VALUES LESS THAN (400),
    PARTITION sales_5 VALUES LESS THAN (500),
    PARTITION sales_6 VALUES LESS THAN (600),
    PARTITION sales_7 VALUES LESS THAN (700),
    PARTITION sales_8 VALUES LESS THAN (800),
    PARTITION sales_9 VALUES LESS THAN (900),
    PARTITION sales_10 VALUES LESS THAN (1000),
    PARTITION sales_default VALUES LESS THAN (1000000)
);
```

LIST partitioning:

```
DROP TABLE sales_data PURGE;

CREATE TABLE sales_data
(
    product_id   NUMBER,
    customer_id  NUMBER,
    sale_date    DATE,
    amount_sold  NUMBER,
    vec          VECTOR(8),
    region       VARCHAR2(20)
)
PARTITION BY LIST (region)
(
    PARTITION RegionA_Partition VALUES
('RegionA1','RegionA2','RegionA3','RegionA4','RegionA5'),
    PARTITION RegionB_Partition VALUES
('RegionB1','RegionB2','RegionB3','RegionB4','RegionB5'),
    PARTITION RegionC_Partition VALUES
('RegionC1','RegionC2','RegionC3','RegionC4','RegionC5'),
    PARTITION RegionD_Partition VALUES
('RegionD1','RegionD2','RegionD3','RegionD4','RegionD5'),
    PARTITION RegionE_Partition VALUES
('RegionE1','RegionE2','RegionE3','RegionE4','RegionE5'),
    PARTITION RegionF_Partition VALUES
('RegionF1','RegionF2','RegionF3','RegionF4','RegionF5'),
    PARTITION RegionG_Partition VALUES
('RegionG1','RegionG2','RegionG3','RegionG4','RegionG5'),
    PARTITION RegionH_Partition VALUES
('RegionH1','RegionH2','RegionH3','RegionH4','RegionH5'),
    PARTITION RegionI_Partition VALUES
('RegionI1','RegionI2','RegionI3','RegionI4','RegionI5'),
    PARTITION RegionJ_Partition VALUES
('RegionJ1','RegionJ2','RegionJ3','RegionJ4','RegionJ5'),
    PARTITION Other_Region_Partition VALUES (DEFAULT)
);
```

HASH partitioning:

```
DROP TABLE sales_data PURGE;

CREATE TABLE sales_data
(
    product_id   NUMBER,
    customer_id  NUMBER,
    sale_date    DATE,
    amount_sold  NUMBER,
    vec          VECTOR(8),
    region       VARCHAR2(20)
)
PARTITION BY HASH (product_id)
PARTITIONS 10;
```

2. **Use this procedure to randomly insert data into the SALES_DATA table:**

```
CREATE OR REPLACE PROCEDURE insert_sales_data(numRows IN INTEGER,
maxProductId IN INTEGER) AS
  TYPE vec_array IS VARRAY(8) OF NUMBER;
BEGIN
  DBMS_RANDOM.INITIALIZE(100);
    FOR i IN 1..numRows LOOP
      INSERT INTO sales_data (product_id, customer_id, sale_date,
amount_sold, vec, region)
        VALUES (round(DBMS_RANDOM.VALUE(1, MaxProductId)),    -- Random
product_id between 1 and 1000
                round(DBMS_RANDOM.VALUE(1, 10000)),   -- Random
customer_id between 1 and 10000
                (DATE '2024-05-10' - DBMS_RANDOM.VALUE(1, 1460)), --
Random sale_date within the last 4 years
                DBMS_RANDOM.VALUE(10, 10000),  -- Random amount_sold
between 10 and 1000
                '[' ||
                    to_char(DBMS_RANDOM.VALUE(0, 1000)) || ',' ||
                    to_char(DBMS_RANDOM.VALUE(0, 1000)) || ',' ||
                    to_char(DBMS_RANDOM.VALUE(0, 1000)) || ',' ||
                    to_char(DBMS_RANDOM.VALUE(0, 1000)) || ',' ||
                    to_char(DBMS_RANDOM.VALUE(0, 1000)) || ',' ||
                    to_char(DBMS_RANDOM.VALUE(0, 1000)) || ',' ||
                    to_char(DBMS_RANDOM.VALUE(0, 1000)) || ',' ||
                    to_char(DBMS_RANDOM.VALUE(0, 1000)) ||
                ']',
                CASE MOD(i, 10)
                    WHEN 0 THEN 'RegionA' || MOD(i,5)
                    WHEN 1 THEN 'RegionB' || MOD(i,5)
                    WHEN 2 THEN 'RegionC' || MOD(i,5)
                    WHEN 3 THEN 'RegionD' || MOD(i,5)
                    WHEN 4 THEN 'RegionE' || MOD(i,5)
                    WHEN 5 THEN 'RegionF' || MOD(i,5)
                    WHEN 6 THEN 'RegionG' || MOD(i,5)
                    WHEN 7 THEN 'RegionH' || MOD(i,5)
                    WHEN 8 THEN 'RegionI' || MOD(i,5)
                    ELSE        'RegionJ' || MOD(i,5)
                END);
```

```
        IF MOD(i, 10000) = 0 THEN
            COMMIT;
        END IF;
    END LOOP;
    COMMIT;
END insert_sales_data;
/

EXEC insert_sales_data(20000, 501);
```

3. **Create a `LOCAL` IVF index on the VEC column of the `SALES_DATA` table:**

```
CREATE VECTOR INDEX vidxivf ON sales_data(vec)
ORGANIZATION NEIGHBOR PARTITIONS
WITH TARGET ACCURACY 95
DISTANCE EUCLIDEAN PARAMETERS(TYPE IVF, NEIGHBOR PARTITION 20) LOCAL;
```

4. **Check all index partitions state before and after running PMOP commands:**

```
SELECT INDEX_NAME, PARTITION_NAME, STATUS
FROM USER_IND_PARTITIONS
WHERE index_name LIKE upper('vidxivf')
ORDER BY 1, 2;
```

5. **Test the following `ALTER TABLE` commands to see what is supported and what is not:**
   If the table is `RANGE` partitioned:

```
ALTER TABLE sales_data ADD PARTITION sales_1000000 VALUES LESS THAN
(2000000);

ALTER TABLE sales_data
SPLIT PARTITION sales_2
AT (150)
INTO (
      PARTITION sales_21,
      PARTITION sales_22
);

ALTER TABLE sales_data DROP PARTITION sales_3;
```

   If the table is `LIST` partitioned:

```
ALTER TABLE sales_data
SPLIT PARTITION RegionE_Partition VALUES ('RegionE1', 'RegionE2',
'RegionE3')
INTO
  ( PARTITION RegionE1_Partition,
    PARTITION RegionE2_Partition
  );

ALTER TABLE sales_data DROP PARTITION RegionB_Partition;
```

If the table is `HASH` partitioned:

```
ALTER TABLE sales_data ADD PARTITION;

SELECT partition_name FROM user_tab_partitions WHERE
table_name='SALES_DATA';

ALTER TABLE sales_data COALESCE PARTITION;
```

Included Columns

Included columns in vector indexes facilitate faster searches with attribute filters by incorporating non-vector columns within a Neighbor Partition Vector Index.

Consider a classic example where you are analyzing the home price data. You want only the top 10 homes from your description where the price is less than 2 million dollars and/or from the specific zip code. Or to a finer granularity, you just need to retrieve only the top home that matches the description and your filter attribute. Included columns make this possible by incorporating filterable attributes, such as price and zip code, directly into the vector index. This integration allows the index to evaluate and return results based on both the vector similarity and the attribute filters, without requiring additional steps to cross-reference the base table.

By bridging the gap between traditional database filters and AI-powered vector searches, included columns deliver faster, more efficient results. This synergy enhances search performance, especially in use cases requiring fine-grained filtering alongside advanced similarity computations.

Syntax and Specifications

Create a base table, which contains the sales data for all products sold in the current year:

```
create table houses(id number, zip_code number, price number,
description clob, data_vector vector);
```

Then, you can create an IVF index **with included column on price** as follows:

```
create vector index vidx_ivf on houses(data_vector) include (price)
organization neighbor partitions with target accuracy 95
distance EUCLIDEAN parameters(type IVF, neighbor partitions 2);
```

The above syntax creates a vector index `VIDX_IVF` with `organization` as `NEIGHBOR PARTITIONS` on the `DATA_VECTOR` column where the `PRICE` column is an included column. The `INCLUDE` keyword allows a user to specify the attributes they wish to include in the IVF index.

> **Note:**
>
> Included columns does not work with partition local indexes.

Recall that the IVF index has two auxiliary tables : Centroids table - `IVF_FLAT_CENTROIDS` and the Centroid Partitions table - `IVF_FLAT_CENTROID_PARTITIONS`. The following code block describes the Centroid Partitions table without the included columns:

```
 Name                          Null?    Type
 ----------------------------------------- --------
----------------------------
 BASE_TABLE_ROWID               NOT NULL ROWID
 CENTROID_ID                    NOT NULL NUMBER
 DATA_VECTOR                             VECTOR(384, FLOAT32, DENSE)
--------------------------------------------------------------------------------
--
```

The list of included columns are augmented in the `IVF_FLAT_CENTROID_PARTITIONS` table of IVF index. The BASE_TABLE_ROWID column can be thought of as an implicit included column. The following table describes the `IVF_FLAT_CENTROID_PARTITIONS` table with the included column augmented as the last row. In this case, the `price`.

```
 Name                          Null?    Type
 ----------------------------------------- --------
----------------------------
 BASE_TABLE_ROWID               NOT NULL ROWID
 CENTROID_ID                    NOT NULL NUMBER
 DATA_VECTOR                             VECTOR(384, FLOAT32, DENSE)
 PRICE                                   NUMBER
--------------------------------------------------------------------------------
--
```

DML operations on the base table also maintain the included columns. Updating any row on the base table for the included columns will all also update the centroid partitions for the IVF index.

> **Note:**
>
> **Restrictions on `INCLUDE` :**
> The `INCLUDE` list can contain:
>
> - Limited to a maximum of 31 columns - Oracle indexes support 32 key columns.
> - Supports only the following data types : `NUMBER`, `CHAR`, `VARCHAR2`, `DATE`, `TIMESTAMP`, `RAW`, `JSON`, and reference-based LOB types such as `BLOB` and `CLOB`.
> - Does not support NLS types and `LONG` types.

Benefits of Using Included Columns

1. **IVF No-Filter with and without Included Columns**:
   If you consider a non-filter query that fetches the two homes nearest to a specified vector, the resulting query will result in an execution plan with multiple joins.

   ```
   select /*+ VECTOR_INDEX_TRANSFORM(t) */ price from houses t order by
   VECTOR_DISTANCE(data_vector, :query_vector) fetch first 2 rows only;
   ```

> **Note:**
>
> `query_vector` contains the actual input vector. You can use the instructions mentioned in SQL Quick Start Using a FLOAT32 Vector Generator to generate query_vector.

The multiple joins are displayed in the execution plan below:

```
Execution Plan
----------------------------------------------------------
Plan hash value: 466477707

-------------------------------------------------------------------------------
----------------------------
| Id  | Operation            | Name                                 |
-------------------------------------------------------------------------------
----------------------------
|   0 | SELECT STATEMENT        |
|
|   1 |  VIEW                |
|
|   2 |   NESTED LOOPS            |
|
|   3 |    VIEW             | VW_IVPSR_11E7D7DE
|
|*  4 |     COUNT STOPKEY        |
|
|   5 |      VIEW            | VW_IVPSJ_578B79F1
|
|*  6 |       SORT ORDER BY STOPKEY    |
|
|   7 |        NESTED LOOPS        |
|
|   8 |     VIEW            | VW_IVCR_B5B87E67
|
|*  9 |      COUNT STOPKEY        |
|
|  10 |       VIEW            | VW_IVCN_9A1D2119
|
|* 11 |        SORT ORDER BY STOPKEY|
|
|  12 |         TABLE ACCESS FULL    |
VECTOR$VIDX_IVF$74478_74488_0$IVF_FLAT_CENTROIDS         |
|  13 |     PARTITION LIST ITERATOR |
|
|* 14 |      TABLE ACCESS FULL    |
VECTOR$VIDX_IVF$74478_74488_0$IVF_FLAT_CENTROID_PARTITIONS |
|  15 |    TABLE ACCESS BY USER ROWID    |
HOUSES                              |
-------------------------------------------------------------------------------
----------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
```

```
   4 - filter(ROWNUM<=2)
   6 - filter(ROWNUM<=2)
   9 - filter(ROWNUM<=1)
  11 - filter(ROWNUM<=1)
  14 - filter("VW_IVCR_B5B87E67"."CENTROID_ID"="VTIX_CNPART"."CENTROID_ID")
```

You can see in the execution plan that the IVF centroids are joined with the centroid partition table to scan for the centroids in the nearest neighbor partitions fetched from the centroid table. This operation is expected. However, you are also performing a join with the base table: HOUSES to apply the attribute filters. This second join against the base table could be avoided if the columns were included in the centroid partitions table.

The following execution plan displays the same operation when using a neighbor partition vector index (IVF) created with included columns:

```
Execution Plan
----------------------------------------------------------
Plan hash value: 504484986

-------------------------------------------------------------------------------
-----------------------------
| Id  | Operation               | Name                                  |
-------------------------------------------------------------------------------
-----------------------------
|   0 | SELECT STATEMENT           |                                     |
|   1 |  VIEW                   |                                     |
|   2 |   VIEW                   |
VW_IVPSR_11E7D7DE                          |
|*  3 |    COUNT STOPKEY           |
|
|   4 |     VIEW                |
VW_IVPSJ_578B79F1                          |
|*  5 |      SORT ORDER BY STOPKEY     |
|
|   6 |       NESTED LOOPS           |
|
|   7 |        VIEW                |
VW_IVCR_B5B87E67                           |
|*  8 |     COUNT STOPKEY           |                                     |
|   9 |      VIEW                |
VW_IVCN_9A1D2119                           |
|* 10 |       SORT ORDER BY STOPKEY|                                     |
|  11 |        TABLE ACCESS FULL   |
VECTOR$VIDX_IVF_1$74483_74508_0$IVF_FLAT_CENTROIDS          |
|  12 |        PARTITION LIST ITERATOR |
|
|* 13 |     TABLE ACCESS FULL      |
VECTOR$VIDX_IVF_1$74483_74508_0$IVF_FLAT_CENTROID_PARTITIONS |
-------------------------------------------------------------------------------
-----------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter(ROWNUM<=2)
```

```
   5 - filter(ROWNUM<=2)
   8 - filter(ROWNUM<=1)
  10 - filter(ROWNUM<=1)
  13 - filter("VW_IVCR_B5B87E67"."CENTROID_ID"="VTIX_CNPART"."CENTROID_ID")
```

As you can see in the execution plan above, adding included columns to a vector index ensures there is no base table pre-filter evaluation required. The query can be resolved by scanning the centroids partitions table for the neighbor partitioned vector index.

2. **IVF Filtering with Included Columns**:
One of the more commonly used plans for neighbor graph vector indexes is the pre-filter plan. In this case you are filtering the contents of the base table to eliminate non-relevant rows. But evaluating a query without using a filter attribute can be very expensive as the operation involves several joins. The following query uses a neighbor partition vector index without included columns:

```
SELECT /*+VECTOR_INDEX_TRANSFORM(houses, vidx_ivf) */ price FROM houses
WHERE price = 1400000 ORDER BY vector_distance(data_vector, :query_vector)
FETCH APPROX FIRST 4 ROWS ONLY;
```

> **Note:**
>
> `query_vector` contains the actual input vector. You can use the instructions mentioned in SQL Quick Start Using a FLOAT32 Vector Generator to generate query_vector.

The execution plan shows multiple joins and the mandatory join with the base table `HOUSES`.

```
Execution Plan
----------------------------------------------------------
Plan hash value: 3359903466

-------------------------------------------------------------------------------
---------------------------------
| Id  | Operation                   | Name
|
-------------------------------------------------------------------------------
---------------------------------
|   0 | SELECT STATEMENT               |
|
|*  1 |  COUNT STOPKEY                   |
|
|   2 |   VIEW                        |                                |
|*  3 |    SORT ORDER BY STOPKEY
|                                 |
|   4 |     NESTED LOOPS               |
|
|   5 |      MERGE JOIN CARTESIAN
|                                 |
|*  6 |       TABLE ACCESS FULL            |
HOUSES                              |
|   7 |       BUFFER SORT                |
```

```
|
|   8 |        VIEW                    |
VW_IVCR_2D77159E                        |
|*  9 |     COUNT STOPKEY               |
|
|  10 |      VIEW                     |
VW_IVCN_9A1D2119                          |
|* 11 |       SORT ORDER BY STOPKEY
|                                 |
|  12 |        TABLE ACCESS FULL           |
VECTOR$VIDX_IVF$74478_74488_0$IVF_FLAT_CENTROIDS        |
|* 13 |      TABLE ACCESS BY GLOBAL INDEX ROWID|
VECTOR$VIDX_IVF$74478_74488_0$IVF_FLAT_CENTROID_PARTITIONS |
|* 14 |       INDEX UNIQUE SCAN            |
SYS_C008791                              |
---------------------------------------------------------------------------
-----------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter(ROWNUM<=4)
   3 - filter(ROWNUM<=4)
   6 - filter("HOUSES"."PRICE"=1400000)
   9 - filter(ROWNUM<=3)
  11 - filter(ROWNUM<=3)
  13 - filter("VW_IVCR_2D77159E"."CENTROID_ID"="VTIX_CNPART"."CENTROID_ID")
  14 - access("HOUSES".ROWID="VTIX_CNPART"."BASE_TABLE_ROWID")
```

On the other hand, if you use the same in-filter query on the IVF index created with `price` as the `included` column, the execution plan would show less joins, the join with the base table is avoided. This is shown in the plan below:

```
Execution Plan
----------------------------------------------------------
Plan hash value: 4183240211

---------------------------------------------------------------------------
----------------------------
| Id  | Operation               | Name                                |
---------------------------------------------------------------------------
----------------------------
|   0 | SELECT STATEMENT           |                                     |
|*  1 |  COUNT STOPKEY               |                                       |
|   2 |   VIEW                    |                                    |
|*  3 |    SORT ORDER BY STOPKEY       |
|
|*  4 |     HASH JOIN                |                                      |
|   5 |      PART JOIN FILTER CREATE
| :BF0000                              |
|   6 |       VIEW                |
VW_IVCR_2D77159E                           |
|*  7 |        COUNT STOPKEY           |
|
|   8 |     VIEW                | VW_IVCN_9A1D2119
```

```
|
|*  9 |      SORT ORDER BY STOPKEY |                                    |
|  10 |       TABLE ACCESS FULL    |
VECTOR$VIDX_IVF_1$74483_74508_0$IVF_FLAT_CENTROIDS          |
|  11 |      PARTITION LIST JOIN-FILTER|
|
|* 12 |       TABLE ACCESS FULL        |
VECTOR$VIDX_IVF_1$74483_74508_0$IVF_FLAT_CENTROID_PARTITIONS |
-------------------------------------------------------------------------------
----------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter(ROWNUM<=4)
   3 - filter(ROWNUM<=4)
   4 - access("VW_IVCR_2D77159E"."CENTROID_ID"="VTIX_CNPART"."CENTROID_ID")
   7 - filter(ROWNUM<=3)
   9 - filter(ROWNUM<=3)
  12 - filter("VTIX_CNPART"."PRICE"=1400000)
```

Inverted File Flat Index Syntax and Parameters

Review the syntax and examples for Inverted File Flat vector indexes.

Syntax

```
CREATE VECTOR INDEX <vector index name>
ON <table name> ( <vector column> )
[GLOBAL] ORGANIZATION [NEIGHBOR] PARTITIONS
[WITH] [DISTANCE <metric name>]
[WITH TARGET ACCURACY <percentage value>
[PARAMETERS ( TYPE IVF, { NEIGHBOR PARTITIONS <number of partitions> |
                          SAMPLES_PER_PARTITION <number of samples> |
                          MIN_VECTORS_PER_PARTITION <minimum number of
vectors per partition> }
           )]]
[PARALLEL <degree of parallelism>]
[LOCAL];
```

The `GLOBAL` clause specifies a global IVF index. By default, IVF vector indexes are globally partitioned by centroid.

Specify the `LOCAL` clause to create a local IVF index.

IVF PARAMETERS

`NEIGHBOR PARTITIONS` determines the target number of centroid partitions that are created by the index.

`SAMPLES_PER_PARTITION` decides the total number of vectors that are passed to the clustering algorithm (`samples_per_partition * neighbor_partitions`). Passing all the vectors could significantly increase the total index creation time. The goal is to pass in a subset of vectors that can capture the data distribution.

MIN_VECTORS_PER_PARTITION represents the target minimum number of vectors per partition. The goal is to trim out any partition that can end up with few vectors (<= 100).

The valid range for IVF vector index parameters are:

- ACCURACY: > 0 and <= 100
- DISTANCE: EUCLIDEAN, L2_SQUARED (aka EUCLIDEAN_SQUARED), COSINE, DOT, MANHATTAN, HAMMING
- TYPE: IVF
- NEIGHBOR PARTITIONS: >0 and <= 10000000
- SAMPLES_PER_PARTITION: from 1 to (num_vectors/neighbor_partitions)
- MIN_VECTORS_PER_PARTITION: from 0 (no trimming of centroid partitions) to total number of vectors (would result in 1 centroid partition)

Examples

- To create a global IVF index:

```
CREATE VECTOR INDEX galaxies_ivf_idx ON galaxies (embedding)
ORGANIZATION NEIGHBOR PARTITIONS
DISTANCE COSINE
WITH TARGET ACCURACY 95;


CREATE VECTOR INDEX galaxies_ivf_idx ON galaxies (embedding)
ORGANIZATION NEIGHBOR PARTITIONS
DISTANCE COSINE
WITH TARGET ACCURACY 90 PARAMETERS (type IVF, neighbor partitions 10);
```

- To create a local IVF index:

```
CREATE VECTOR INDEX galaxies_ivf_idx ON galaxies (embedding)
ORGANIZATION NEIGHBOR PARTITIONS
DISTANCE COSINE
WITH TARGET ACCURACY 90 PARAMETERS (type IVF, neighbor partitions 10)
LOCAL;
```

For detailed information on the syntax, see CREATE VECTOR INDEX in *Oracle Database SQL Language Reference*.

Related Topics

- Inverted File Flat Vector Indexes Partitioning Schemes
  Inverted File Flat vector indexes support both global and local indexes on partitioned tables. By default, IVF indexes are globally partitioned by centroid.

Guidelines for Using Vector Indexes

Use these guidelines to create and use Hierarchical Navigable Small World (HNSW) or Inverted File Flat (IVF) vector indexes.

Create Index Guidelines

The minimum information required to create a vector index is to specify one `VECTOR` data type table column and a vector index type: `INMEMORY NEIGHBOR GRAPH` for HNSW and `NEIGHBOR PARTITIONS` for IVF. However, you also have the possibility to specify more information, such as the following:

- You can optionally provide more information including the distance metric to use. Supported metrics are `EUCLIDEAN SQUARED`, `EUCLIDEAN`, `COSINE`, `DOT`, `MANHATTAN`, and `HAMMING`. If not specified, `COSINE` is used by default.
- Specific parameters that impact the accuracy of index creation and approximate searches. A target accuracy percentage value and, `NEIGHBORS` (or M) and `EFCONSTRUCTION` for HNSW and `NEIGHBOR PARTITIONS` for IVF.
- You can create globally partitioned vector indexes.
- You can also specify the degree of parallelism to use for index creation.

You cannot currently define a vector index on:

- External tables
- IOTs
- Clusters/Cluster tables
- Global Temp tables
- Blockchain tables
- Immutable tables
- Materialized views
- Function-based vector index

You can find information about your vector indexes by looking at `ALL_INDEXES`, `DBA_INDEXES`, and `USER_INDEXES` family of views. The columns of interest are `INDEX_TYPE` (`VECTOR`) and `INDEX_SUBTYPE` (`INMEMORY_NEIGHBOR_GRAPH_HNSW` or `NEIGHBOR_PARTITIONS_IVF`). In the case the index is not a vector index, `INDEX_SUBTYPE` is `NULL`.

See VECSYS.VECTOR$INDEX for detailed information about vector indexes.

Note:

- The `VECTOR` column is designed to be extremely flexible to support vectors of any number of dimensions and any format for the vector dimensions. However, you can create a vector index only on a `VECTOR` column containing vectors that all have the same number of dimensions. This is required as you can't compute distances over vectors with different dimensions. For example, if a `VECTOR` column is defined as `VECTOR(*, FLOAT32)`, and two vectors with different dimensions (128 and 256 respectively) are inserted in that column. When you try to create the vector index on that column, you will get an error.
- Once an IVF index is created on a vector column of an empty table and non-null vectors with a particular dimension have been inserted into it, new vectors with a different dimension cannot be inserted into the same vector column due to the existence of the IVF index.
- If a table is truncated, any IVF index created on the table is marked as `UNUSABLE`. Thus, the IVF index will not be maintained on any subsequent `INSERT` statements with vectors of the same or different dimension count. You must rebuild or recreate the IVF index before using the index in a query again.
- You can only create one type of vector index per vector column.
- Oracle recommends that you allocate larger, temporary tablespaces for proper IVF vector index creation with big vector spaces and vector sizes. In such cases, the system internally makes extensive use of temporary space.
- On a RAC environment you can set up a vector pool on each instance for the best performance of vector indexes.

Use Index Guidelines

For the Oracle Database Optimizer to consider a vector index, you must ensure these conditions in your SQL statements:

- The similarity search SQL query must include the `APPROX` or `APPROXIMATE` keyword.
- The vector index must exist.
- The distance function for the index must be the same as the distance function used in the `vector_distance()` function.
- If the vector index DDL does not specify the distance function and the `vector_distance()` function uses `EUCLIDEAN`, `DOT`, `MANHATTAN` or `HAMMING`, then the vector index is not used.
- If the vector index DDL uses the `DOT` distance function and the `vector_distance()` function uses the default distance function `COSINE`, then the vector index is not used.
- The `vector_distance()` must not be encased in another SQL function.
- If using the partition row-limiting clause, then the vector index is not used.
- Index accuracy with an IVF index may diminish over time due to DML operations being performed on the underlying table. You can check for this by using the `INDEX_ACCURACY_QUERY` function provided by the `DBMS_VECTOR` package. In such a case, the index can be rebuilt using the `REBUILD_INDEX` function also provided by the `DBMS_VECTOR` package. See DBMS_VECTOR for more information about `DBMS_VECTOR` and its subprograms.

> **Note:**
>
> Except for `ALTER TABLE tab SUBPARTITION/PARTITION (RANGE/LIST)` with or without Update Global Indexes, global vector indexes are marked as unusable as part of other Partition Management Operations on a partitioned table. In those cases you must manually rebuild the global vector indexes.

Index Accuracy Report

The index accuracy reporting feature lets you determine the accuracy of your vector indexes.

Accuracy of One Specific Query Vector

After a vector index is created, you may be interested to know how accurate your vector searches are. One possibility might be to run two queries using the same query vector, that is, one performing an approximate search using a vector index and the other performing an exact search without an index. Then, you need to manually compare the results to determine the real accuracy of your index.

Instead, you can use an index accuracy report provided by the `DBMS_VECTOR.INDEX_ACCURACY_QUERY` procedure. This procedure provides an accuracy report for a top-K index search for a specific query vector and a specific target accuracy. For syntax and parameter details, see INDEX_ACCURACY_QUERY.

Here is a usage example of this procedure using our galaxies scenario:

```
declare
    q_v VECTOR;
    report varchar2(128);
begin
    q_v := to_vector('[0,1,1,0,0]');
    report := dbms_vector.index_accuracy_query(
        OWNER_NAME => 'COSMOS',
        INDEX_NAME => 'GALAXIES_HNSW_IDX',
        qv => q_v, top_K =>10,
        target_accuracy =>90 );
    dbms_output.put_line(report);
end;
/
```

The preceding example computes the top-10 accuracy of the `GALAXIES_HNSW_IDX` vector index using the embedding corresponding to the NGC 1073 galaxy and a 90% accuracy requested.

The index accuracy report for this may look like:

```
Accuracy achieved (100%) is 10% higher than the Target Accuracy requested
(90%)
```

The possible parameters are:

- `owner_name`: Index owner name
- `index_name`: Index name
- `qv`: Query vector

- `top_K`: Top K value for accuracy computation
- `target_accuracy`: Target accuracy for the index

Accuracy of Automatically Captured Query Vectors

An overloaded version of the `DBMS_VECTOR.INDEX_ACCURACY_REPORT` function allows you to capture from your past workloads, accuracy values achieved by your approximate searches using a particular vector index for a certain period of time. Query vectors used for approximate searches are captured automatically in memory and persisted to a catalog table every hour.

The `INDEX_ACCURACY_REPORT` function computes the achieved accuracy using the captured query vectors for a given index. To compute the achieved accuracy for each query vector, the function compares the result set of approximate similarity searches with exact similarity searches for the same query vectors.

The accuracy findings are stored in dictionary and exposed using the `DBA_VECTOR_INDEX_ACCURACY_REPORT` dictionary view.

Here is a usage example of this function using the galaxies scenario:

```
VARIABLE t_id NUMBER;
BEGIN
  :t_id := DBMS_VECTOR.INDEX_ACCURACY_REPORT('VECTOR', 'GALAXIES_HNSW_IDX');
END;
/
```

You can also run the following statement to get the corresponding task identifier:

```
SELECT DBMS_VECTOR.INDEX_ACCURACY_REPORT('VECTOR', 'GALAXIES_HNSW_IDX');
```

The following are possible parameters for the `INDEX_ACCURACY_REPORT` function:

- `owner_name` (IN): Index owner name
- `ind_name` (IN): Index name
- `start_time` (IN): Query vectors captured from this time are considered for the accuracy computation. A `NULL` `start_time` uses query vectors captured in the last 24 hours.
- `end_time` (IN): Query vectors captured until this time are considered for accuracy computation. A `NULL` `end_time` uses query vectors captured from `start_time` until the current time.
- Return Values: A numeric task ID if the accuracy for the given index was successfully computed. Otherwise, a `NULL` task ID is returned.

Note:

- If both `start_time` and `end_time` are `NULL`, accuracy is computed using query vectors captured in the last 24 hours.
- If `start_time` is `NULL` and `end_time` is not `NULL`, accuracy is computed using query vectors captured between 24 hours before `end_time` until `end_time`.
- If `start_time` is not `NULL` and `end_time` is `NULL`, accuracy is computed using query vectors captured between `start_time` and the current time.

You can see the analysis results using the `DBA_VECTOR_INDEX_ACCURACY_REPORT` view:

```
desc DBA_VECTOR_INDEX_ACCURACY_REPORT


 Name                                      Null?    Type
 ----------------------------------------- --------
----------------------------
 TASK_ID                                            NUMBER
 TASK_TIME                                          TIMESTAMP(6)
 OWNER_NAME                                         VARCHAR2(128)
 INDEX_NAME                                         VARCHAR2(128)
 INDEX_TYPE                                         VARCHAR2(16)
 MIN_TARGET_ACCURACY                                NUMBER
 MAX_TARGET_ACCURACY                                NUMBER
 NUM_VECTORS                                        NUMBER
 MEDIAN_ACHIEVED_ACCURACY                           NUMBER
 MIN_ACHIEVED_ACCURACY                              NUMBER
 MAX_ACHIEVED_ACCURACY                              NUMBER
```

Select target accuracy values in the following statement:

```
SELECT MIN_TARGET_ACCURACY, MAX_TARGET_ACCURACY, num_vectors,
MIN_ACHIEVED_ACCURACY, MEDIAN_ACHIEVED_ACCURACY, MAX_ACHIEVED_ACCURACY
FROM DBA_VECTOR_INDEX_ACCURACY_REPORT WHERE task_id = 1;



MIN_TARGET_ACCURACY MAX_TARGET_ACCURACY NUM_VECTORS MIN_ACHIEVED_ACCURACY
MEDIAN_ACHIEVED_ACCURACY MAX_ACHIEVED_ACCURACY
------------------- ------------------- ----------- ---------------------
----------------------- ---------------------
                  1                  10           2
49                        57                    65
                 11                  20           3
60                        73                    83
                 21                  30           3
44                        64                    84
                 31                  40           2
63                      76.5                    90
                 41                  50           3
63                        81                    90
```

```
                61                  70           2
57                        68                  79
                71                  80           3
79                        87                  89
                81                  90           3
70                        71                  78
                91                 100           4
67                      79.5                  88
```

Each row in the output represents a bucket of 10 target accuracy values: 1-10, 11-20, 21-30, … , 91-100.

Consider the following partial statement that runs an approximate similarity search for a particular query vector and a particular target accuracy:

```
SELECT ...
FROM ...
WHERE ...
ORDER BY VECTOR_DISTANCE( embedding, :my_query_vector, COSINE )
FETCH APPROXIMATE FIRST 3 ROWS ONLY WITH TARGET ACCURACY 65;
```

Once the preceding approximate similarity search is run and captured by your accuracy report task, the value of `NUM_VECTORS` would be increased by one in row 6 (bucket values between 61 and 70) of the results of the select statement on the `DBA_VECTOR_INDEX_ACCURACY_REPORT` view for your task. `NUM_VECTORS` represents the number of query vectors that fall in a particular target accuracy bucket.

`MIN_ACHIEVED_ACCURACY`, `MEDIAN_ACHIEVED_ACCURACY`, and `MAX_ACHIEVED_ACCURACY` are the actual achieved accuracy values for the given target accuracy bucket.

> **Note:**
>
> The initialization parameter `VECTOR_QUERY_CAPTURE` is used to enable and disable capture of query vectors. The parameter value is set to `ON` by default. You can turn off this background functionality by setting `VECTOR_QUERY_CAPTURE` to `OFF`. When `VECTOR_QUERY_CAPTURE` is `ON`, the database captures some of the query vectors through sampling. The captured query vectors are retrained for a week and then purged automatically.

Vector Index Status, Checkpoint, and Advisor Procedures

Review these high-level details on the `GET_INDEX_STATUS`, `ENABLE_CHECKPOINT`, `DISABLE_CHECKPOINT`, and `INDEX_VECTOR_MEMORY_ADVISOR` procedures that are available with the `DBMS_VECTOR` PL/SQL package.

GET_INDEX_STATUS

Purpose: To query the status of a vector index creation.

Syntax:

```
DBMS_VECTOR.GET_INDEX_STATUS ('USER_NAME','INDEX_NAME');
```

Usage Notes:

- You can use the `GET_INDEX_STATUS` procedure only during a vector index creation.
- The `Percentage` value is shown in the output only for Hierarchical Navigable Small World (HNSW) indexes (and not for Inverted File Flat (IVF) indexes).
- Along with the `DB_DEVELOPER_ROLE` privilege, you must have read access to the `VECSYS.VECTOR$INDEX$BUILD$` table.
- You can use the following query to view all auxiliary tables:

```
select IDX_AUXILIARY_TABLES from vecsys.vector$index;
```

  - For HNSW indexes:

    `rowid_vid_map` stores the mapping between a row ID and vector ID. `shared_journal_change_log` stores the DML changes that are yet to be incorporated into an HNSW graph.

  - For IVF indexes:

    `centroids` stores the location for each centroid. `centroid_partitions` stores the closest centroid for each vector.

- The possible values of `Stage` for HNSW vector indexes are:

Value	Description
`HNSW Index Initialization`	Initialization phase for the HNSW vector index creation
`HNSW Index Auxiliary Tables Creation`	Creation of the internal auxiliary tables for the HNSW Neighbor Graph vector index
`HNSW Index Graph Allocation`	Allocation of memory from the vector memory pool for the HNSW graph
`HNSW Index Loading Vectors`	Loading of the base table vectors into the vector pool memory
`HNSW Index Graph Construction`	Creation of the multi-layered HNSW graph with the previously loaded vectors
`HNSW Index Creation Completed`	HNSW vector index creation finished

- The possible values of `Stage` for IVF vector indexes are:

Value	Description
`IVF Index Initialization`	Initialization phase for the IVF vector index creation
`IVF Index Centroids Creation`	The K-means clustering phase that computes the cluster centroids on a sample of base table vectors
`IVF Index Centroid Partitions Creation`	Centroids assignment phase for the base table vectors
`IVF Index Creation Completed`	IVF vector index creation completed

Example:

```
exec DBMS_VECTOR.GET_INDEX_STATUS('VECTOR_USER','VIDX_HNSW');

Index objn: 74745
```

```
Stage: HNSW Index Loading Vectors
Percentage: 80%
```

ENABLE_CHECKPOINT

Purpose: To enable the Checkpoint feature for a given HNSW index user and HNSW index name.

> **Note:**
>
> This procedure only allows the index to create checkpoints. The checkpoint is created as part of the next HNSW graph refresh.

The `INDEX_NAME` clause is optional. If you do not specify the index name, then this procedure enables the Checkpoint feature for all HNSW indexes under the given user.

Syntax:

```
DBMS_VECTOR.ENABLE_CHECKPOINT('INDEX_USER',['INDEX_NAME']);
```

Example 1: Using index name and index user:

```
DBMS_VECTOR.ENABLE_CHECKPOINT('VECTOR_USER','VIDX1');
```

Example 2: Using index user:

```
DBMS_VECTOR.ENABLE_CHECKPOINT('VECTOR_USER');
```

> **Note:**
>
> By default, HNSW checkpointing is enabled. You can disable it using the `DBMS_VECTOR.DISABLE_CHECKPOINT` procedure.

DISABLE_CHECKPOINT

Purpose: To purge all older checkpoint data. This procedure disables the Checkpoint feature for a given HNSW index user and HNSW index name. It also disables the creation of future checkpoints as part of the HNSW graph refresh.

The *INDEX_NAME* clause is optional. If you do not specify the index name, then this procedure disables the Checkpoint feature for all HNSW indexes under the given user.

Syntax:

```
DBMS_VECTOR.DISABLE_CHECKPOINT('INDEX_USER',['INDEX_NAME']);
```

Example 1: Using index name and index user:

```
DBMS_VECTOR.DISABLE_CHECKPOINT('VECTOR_USER','VIDX1');
```

Example 2: Using index user:

```
DBMS_VECTOR.DISABLE_CHECKPOINT('VECTOR_USER');
```

INDEX_VECTOR_MEMORY_ADVISOR

Purpose: To determine the vector memory size needed for a particular vector index. This helps you evaluate the number of indexes (HNSW or IVF) that can fit for each simulated vector memory size.

Syntax:

- Using the number and type of vector dimensions that you want to store in your vector index.

```
DBMS_VECTOR.INDEX_VECTOR_MEMORY_ADVISOR(
    INDEX_TYPE,
    NUM_VECTORS,
    DIM_COUNT,
    DIM_TYPE,
    PARAMETER_JSON,
    RESPONSE_JSON);
```

- Using the table and vector column on which you want to create your vector index:

```
DBMS_VECTOR.INDEX_VECTOR_MEMORY_ADVISOR(
    TABLE_OWNER,
    TABLE_NAME,
    COLUMN_NAME,
    INDEX_TYPE,
    PARAMETER_JSON,
    RESPONSE_JSON);
```

`INDEX_TYPE` can be one of the following:

- `IVF` for IVF vector indexes
- `HNSW` for HNSW vector indexes

`PARAMETER_JSON` can have only one of the following form:

- `PARAMETER_JSON=>{"accuracy":value}`
- `INDEX_TYPE=>IVF, parameter_json=>{"neighbor_partitions":value}`
- `INDEX_TYPE=>HNSW, parameter_json=>{"neighbors":value}`

> **Note:**
>
> You cannot specify values for `accuracy` along with `neighbor_partitions` or `neighbors`.

Example 1: Using neighbors in the parameters list:

```
exec DBMS_VECTOR.INDEX_VECTOR_MEMORY_ADVISOR(
    INDEX_TYPE=>'HNSW',
```

```
    NUM_VECTORS=>10000,
    DIM_COUNT=>768,
    DIM_TYPE=>'FLOAT32',
    PARAMETER_JSON=>'{"neighbors":128}',
    RESPONSE_JSON=>:response_json);

Suggested vector memory pool size: 59918628 Bytes
```

Example 2: Using accuracy in the parameters list:

```
exec DBMS_VECTOR.INDEX_VECTOR_MEMORY_ADVISOR(
    INDEX_TYPE=>'HNSW',
    NUM_VECTORS=>10000,
    DIM_COUNT=>768,
    DIM_TYPE=>'FLOAT32',
    PARAMETER_JSON=>'{"accuracy":90}',
    RESPONSE_JSON=>:response_json);

Suggested vector memory pool size: 53926765 Bytes
```

Example 3: Using the table and vector column on which you want to create the vector index:

```
exec DBMS_VECTOR.INDEX_VECTOR_MEMORY_ADVISOR(
    'VECTOR_USER',
    'VECTAB',
    'DATA_VECTOR',
    'HNSW',
    RESPONSE_JSON=>:response_json);

Using default accuracy: 90%
Suggested vector memory pool size: 76396251 Bytes
```

Related Topics

- DBMS_VECTOR
  The `DBMS_VECTOR` package simplifies common operations with Oracle AI Vector Search, such as extracting chunks or embeddings from user data, generating text for a given prompt or an image, creating a vector index, or reporting on index accuracy.

Manage Hybrid Vector Indexes

Learn how to manage a hybrid vector index, which is a single index for searching by *similarity* and *keywords*, to enhance the accuracy of your search results.

- Understand Hybrid Vector Indexes
  A hybrid vector index inherits all the information retrieval capabilities of Oracle Text search indexes and leverages the semantic search capabilities of Oracle AI Vector Search vector indexes.
- Guidelines and Restrictions for Hybrid Vector Indexes
  Review these guidelines and the current set of restrictions when working with hybrid vector indexes.

- CREATE HYBRID VECTOR INDEX
  Use the `CREATE HYBRID VECTOR INDEX` SQL statement to create a hybrid vector index, which allows you to index and query documents using a combination of full-text search and vector similarity search.
- ALTER INDEX
  Use the `ALTER INDEX` SQL statement to modify an existing hybrid vector index.

Understand Hybrid Vector Indexes

A hybrid vector index inherits all the information retrieval capabilities of Oracle Text search indexes and leverages the semantic search capabilities of Oracle AI Vector Search vector indexes.

Hybrid vector indexes allow you to index and query documents using a combination of full-text search and semantic vector search. A hybrid vector index is a class of specialized Domain Index that combines the existing Oracle Text indexing data structures and vector indexing data structures into *one unified structure*. A single index contains both textual and vector fields for a document, enabling you to perform a combination of keyword search and vector search simultaneously.

The purpose of a hybrid vector index is to enhance search relevance of an Oracle Text index by allowing users to search by both *vectors* and *keywords* in various combinations, using out-of-the-box and custom scoring techniques. By integrating traditional keyword-based text search with vector-based similarity search, you can improve the overall search experience and provide users with more accurate information.

When to Use a Hybrid Vector Index

Consider using a hybrid vector index for hybrid search scenarios where your query requires information that is semantically similar but pertains to a specific focus area, that is, involves a particular organization, user name, product code, technical term, date, or time. For example, a typical hybrid search query can be to find "`top 10 instances of stock fraud for ABC Corporation`".

Such a query involves two separate components:

- One where you want to identify the notion of "`stock fraud`"
- The second where you want to narrow the results to focus only on "`ABC Corporation`"

Pure keyword search may return results that specifically contain the query words like "stock", "fraud", "ABC", or "Corporation" because it focuses on matching the exact keywords or surface-level representation of words or phrases with tokenized terms in a text index. Therefore, keyword search alone may not be suitable here because it can overlook the semantic meaning behind the words in our query, especially if the exact terms are not present in the content.

Pure vector search focuses on understanding the meaning and context of words or phrases rather than just matching keywords. Vector search considers semantic relationship between the query words, so it may include more contextually-relevant results like "corporate fraud", "stock market manipulation", "stock misconduct", "financial irregularities", or "lawsuits in the financial sector". Vector search also may not be suitable here because it can include results about the broader topic of stock fraud involving ABC or similar organizations, especially if the exact phrase "stock fraud for ABC Corporation" is not present in the content.

Hybrid search can address both components of such a query by running keyword search and vector search on the same data and then combining the two search results into a single result

set. In this way, you can utilize the strengths of both text indexes and vector indexes to retrieve the most relevant results.

Why Choose a Hybrid Vector Index?

Let us summarize the advantages of a hybrid vector index.

- Higher recall compared to pure vector search or keyword search:

  As discussed earlier, hybrid search lets you combine the power of Oracle AI Vector Search and Oracle Text Search to provide more accurate and personalized information. Keyword search or vector search alone may not be relevant in complex search scenarios and may lead to a lot of spurious results.

- Mitigates the downsides of chunking:

  Vector embedding models usually impose limits on the size of input text, which forces large documents to be split into smaller chunks of data for semantic search (as explained in Understand the Stages of Data Transformations). A chunk can lose broader context of the original document due to truncation, which may lead to missed results. A hybrid vector index helps to restore the entire context of each document by performing textual search at the document level.

- Simple to deploy and manage compared to maintaining independent indexes:

  A hybrid vector index is a single domain index that can maintain text and vectors with DML. Both keyword search and vector search are performed on all documents, and then the two search results are combined and scored to return a unified result set. It provides an end-to-end indexing pipeline that automatically transforms your input data for vector search alongside keyword search, thereby enhancing the indexing performance. This index exposes optional preferences to configure indexing parameters, but it is not required.

  Here is an example of the hybrid vector index DDL:

  ```
  CREATE HYBRID VECTOR INDEX my_hybrid_idx on
    DOCS(file_name)
    PARAMETERS ('MODEL MY_INDB_MODEL');
  ```

  Here, a hybrid vector index named `my_hybrid_idx` is created on the `file_name` column of the `DOCS` table. The embedding model used for vector generation is an in-database ONNX format model named `MY_INDB_MODEL`.

  The preceding example shows the minimum input requirements for index creation. For complete syntax, see CREATE HYBRID VECTOR INDEX.

- Unified query API to search by vectors and keywords:

  A single `SEARCH` API (available with the `DBMS_HYBRID_VECTOR` PL/SQL package) lets you specify both a traditional `CONTAINS` query on document text indexes and a `VECTOR_DISTANCE` query on vectorized chunk indexes. You can switch between keyword-only, vector-only, and hybrid search modes to retrieve the best documents or chunks.

  Here is an example of hybrid search using the `SEARCH` API for our earlier "`top 10 instances of stock fraud for ABC Corporation`" scenario:

  ```
  select DBMS_HYBRID_VECTOR.SEARCH(
      json('{ "hybrid_index_name"     : "my_hybrid_idx",
              "vector":
                      { "search_text" : "stock fraud" },
              "text"  :
  ```

```
                    { "contains"    : "$ABC AND $Corporation" },
              "return":
                    { "topN"        : 10 }
          }'))
from dual;
```

This query specifies the search text for vector search (using the vector distance function) as `stock fraud`, the search text for keyword search (using the `CONTAINS` Oracle Text index operator) as `$ABC AND $Corporation`, and the maximum number of rows to return as the top 10. To understand how to use stem (`$`) and other `CONTAINS` operators, see *Oracle Text Reference*.

For complete syntax, see Understand Hybrid Search.

Use Case Examples of a Hybrid Vector Index

Here are some use case scenarios to understand how you can implement a hybrid vector index.

- Fraud detection:

  You can identify fraudulent transactions that do not follow the exact patterns of known fraud cases but are similar in behavior or context. In this case, a hybrid vector index can combine traditional rule-based filters (such as transactions above a certain threshold or in specific locations) with a vector index that compares transaction embeddings (which capture features like timing, merchant type, and transaction history) to identify semantically similar but not identical fraudulent patterns.

- Legal document analysis:

  You need to search large volumes of legal documents for relevant cases and precedents. You might first filter documents using traditional legal jargon and keyword searches. Then, a vector index built from document embeddings can further filter previously identified documents that are semantically similar, even if they use different terminology or legal reasoning, ensuring comprehensive research.

  You can perform the search in a different order if, for example, you are looking for cases that are contextually similar to a landmark case but need to ensure that the cases also include specific legal terminologies or citations. Here, the search can start with vector-based search to retrieve cases that are semantically similar to the landmark case using embeddings. Then, you can apply a traditional keyword-based filter to ensure that the results include specific legal terms, statutes, or citations.

- Medical document analysis:

  Medical research papers or articles either directly reference the terms of interest or do not mention those keywords at all because these documents are written by researchers or scientists in a distinct field of specialization. Here, you can switch between keyword-only and vector-only searches to quickly locate documents that mention specific medical terms of interest and specialization areas, respectively. You can also combine semantic search with keyword search in a different order, starting with a vector-based semantic search using embeddings to first identify papers that are contextually similar to your research area. Then, apply a keyword-based filter to further refine the results to include only specific references or medical terms that are critical to your research.

- HR recruitment:

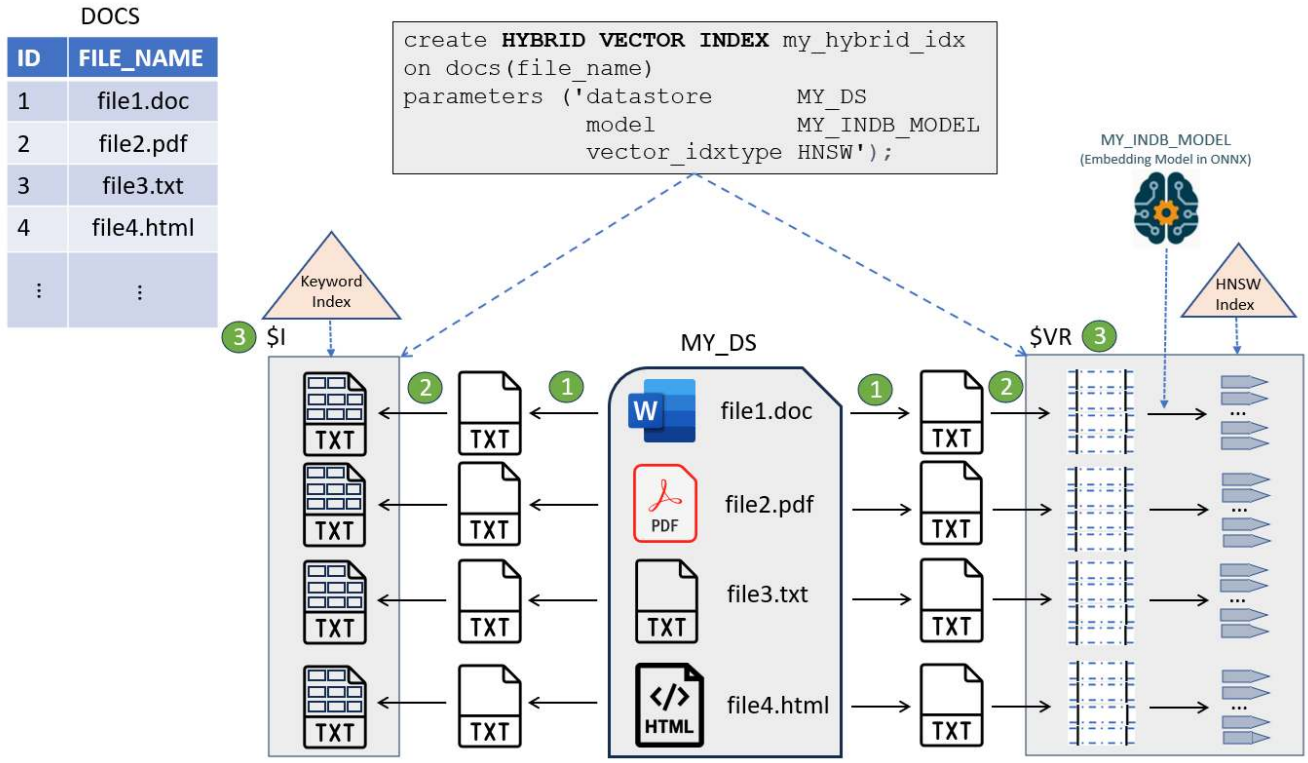
  You want to recruit new employees with strong technical skills in programming languages but who also display certain personality and interpersonal traits. Here, you can first apply a keyword-based filter to best match the technical skills (such as "Java" and "Database"). Then, you can perform a semantic search to identify the personality and interpersonal traits

(such as "team work" and "leadership experience"). A hybrid search approach can help recruiters to effectively shortlist candidates by scanning and comparing resumes or portfolios with a dual focus.

Hybrid Vector Index Creation Overview

You create a hybrid vector index by simply specifying on which table and column to create it along with some details, such as the local or remote location where all source documents are stored (datastore), the ONNX in-database embedding model to use for generating embeddings, and the type of vector index to create. You can specify additional parameters that are discussed later in this chapter.

As illustrated in the following diagram, the hybrid vector index DDL creates a single index that contains both textual fields (with derived text tokens) and vector fields (with extracted chunks and corresponding embeddings) for each indexed document.



As you can see, the implementation of a hybrid vector index leverages the existing capabilities of the Oracle Text search index and the Oracle AI Vector Search vector index. You can define PL/SQL preferences to customize all these indexing pipeline stages for both the index types.

A document table `DOCS` contains IDs and corresponding document names or file names stored in a location called `MY_DS` datastore.

The indexing pipeline starts with reading the documents from `MY_DS` (datastore), and then passes the documents through a series of processing stages:

1. Filter (conversion of binary documents such as PDF, Word, or Excel to plain text)
2. Tokenizer (tokenization of data for keyword search) and Vectorizer (chunking and embedding generation for vector search)

3. Indexing Engine (creation of secondary tables)

   As the system passes documents through this indexing engine, it populates and indexes a set of secondary tables that are collectively part of a hybrid vector index.

   The two main secondary tables created are:

   - `$I` is the same structure as the existing Oracle Text index, which contains the inverted indexed data with tokenized terms.

     An Oracle Text index is created over a textual column based on your specifications. For a deeper understanding of the Oracle Text indexing process, see *Oracle Text Application Developer's Guide*.

   - `$VR` contains the indexed data with generated chunks and corresponding embeddings.

     A vector index is also created over the `VECTOR` column based on your specifications. `$VR` also contains a `ROWID` column that maps back to the document table's row IDs, and a `DOCID` column that links back to Oracle Text document-level IDs. This creates a link between chunks, tokens, and documents.

     You can directly examine the `$VR` table using the dictionary view `<index name>$VECTORS`, which lets you query all row ids, chunks, and embeddings. See <index name>$VECTORS.

   Temporarily, the `$D` secondary table (not shown in the diagram) is also created to save a copy of the document if the original document is not already in the database or needs filtering. This table gets truncated after chunks are obtained.

Hybrid Vector Index Maintenance Operations

A hybrid vector index supports mostly all the traditional operations of an Oracle Text index, such as `MAINTENANCE AUTO` , `SYNC`, or `OPTIMIZE`. For details, see Guidelines and Restrictions for Hybrid Vector Indexes.

Related Topics

- Creating and Querying Oracle Text Indexes
- Perform Hybrid Search
  Hybrid search is an advanced information retrieval technique that lets you search documents by *keywords* and *vectors*, to achieve more relevant search results.

Guidelines and Restrictions for Hybrid Vector Indexes

Review these guidelines and the current set of restrictions when working with hybrid vector indexes.

Index Creation Guidelines

A hybrid vector index is a single domain index that includes both Oracle Text search index and Oracle AI Vector Search vector index. Any failure during the construction of one index type can cause the entire index creation to fail as a whole. Therefore, you must adhere to all the indexing guidelines that are individually applicable to a vector index and an Oracle Text search index.

For example, if there is insufficient vector memory pool for the HNSW vector index or if there is inadequate tablespace, then the vector index-related error can lead to a failure in creating a hybrid vector index.

- If a destructive operation occurs, such as running `Ctrl+C` or shutting down the database, then you must check the status of the index. If the index status does not display as `valid`,

it indicates that the index may be corrupted or inconsistent, making it unsafe for use. In such cases, you must recreate the index.

- If there are any errors in the hybrid vector index DDL or if there is any DDL failure, then you must manually drop the index and create it again. Before using the index, ensure that it is safe to use by checking the index status.

To check the status of your index, you can query the regular Oracle Text data dictionary views, such as `CTX_INDEXES` and `CTX_USER_INDEXES`. See *Oracle Text Reference*.

Hybrid Vector Index Maintenance Operations

You can maintain your hybrid vector indexes just like an Oracle Text index. A hybrid vector index supports mostly all the traditional operations of Oracle Text indexes, such as Synchronization, Optimization, and Automatic Maintenance.

Here are some guidelines on maintaining indexes after DML operations on base tables:

- **Default index maintenance**:

  A hybrid vector index runs in an automatic maintenance mode (`MAINTENANCE AUTO`), which means that your DMLs are automatically synchronized into the index in the background at optimal intervals.

  You do not need to manually configure a maintenance type or synchronization type for maintaining a hybrid vector index. However, if required, you can modify the default settings to set any synchronization interval, specify a `SYNC` type (such as `MANUAL`, `EVERY`, or `ON COMMIT`), or schedule any background synchronization job using the `DBMS_SCHEDULER`.

  In an automatic maintenance mode, indexes are asynchronously maintained without any user intervention. Oracle recommends that you periodically examine regular Oracle Text views to know the status of all background maintenance events.

  For information on these views, see *Oracle Text Application Developer's Guide*.

- **Default index optimization**:

  A hybrid vector index runs with an automatic background Optimize Full job every midnight "local" time. This job optimizes your index to defragment it and clean up any lazy deletes from secondary tables such as the `$I`, `$D`, and `$VR`.

- **Optimizing the $VR Table**

  For a hybrid vector index, you can also optimize only the `$VR` table. This optimization is run in “section mode” for the new “section” (semantic index). This will compact the `$VR` table only by removing any deleted ROWIDs.

  ```
  CTX_DDL.OPTIMIZE_INDEX([schema.]index_name, 'token_type',
                         'ctx_ddl.section_semantic_index');
  ```

  Oracle recommends you to explicitly run index optimization more frequently, especially if your hybrid vector index involves a large number of inserts, updates, or deletes to base tables. This is because frequent index synchronization can cause fragmentation of your index, which can adversely affect query response time. You can reduce fragmentation and index size by optimizing the index with `CTX_DDL.OPTIMIZE_INDEX`.

  For information on how to configure optimization methods, see the index maintenance preferences in CREATE HYBRID VECTOR INDEX.

Guidelines to Run Direct SQL Queries on $VR

If you query secondary tables such as `$VR` directly before an index optimization job is run, you may still see old data or deleted rows because these are ignored at query time by Oracle provided search API (`DBMS_HYBRID_VECTOR.SEARCH`). For example, you may see lazy deletes that mark the row IDs as deleted but are only removed from secondary tables after the index is optimized.

Instead of examining the `$VR` table directly, you can use the data dictionary view `<index name>$VECTORS` to work with the contents of the `$VR` table, and accordingly run direct SQL queries on the view. This view lets you query the document table's row IDs by excluding all lazy deletes (along with corresponding chunks and embeddings that are generated).

Note that the `<index name>$VECTORS` view resides in a user's schema. Therefore, if a hybrid vector index is named `IDX`, then the view name is `IDX$VECTORS`.

For information on this view, see <index name>$VECTORS.

Guidelines for VPD Policy Protection

If you have applied an Oracle Virtual Private Database (VPD) policy to the base table, then you must:

- Protect the entire hybrid vector index with the same VPD policy, including the `$I`, `$VR`, and `$D` secondary tables.
- Protect the `<index name>$VECTORS` view (which is created on the `$VR` table) with the same VPD policy.
- Ensure that any SQL queries run on the hybrid vector index and any direct SQL queries run on secondary tables that are created within the schema also respect that VPD policy.

For information on how to use VPD to control data access in general, see *Oracle Database Security Guide*.

Hybrid Vector Index DDL Restrictions

- The supported data types are `CLOB`, `VARCHAR2`, or `BLOB`. Textual columns with the `IS JSON` check constraint are not supported.

  Other data types such as JSON are currently not supported for a hybrid vector index.
- Creating a local hybrid vector index is not supported.
- You currently cannot import externally created chunks or vectors directly into the `$VR` table of a hybrid vector index.
- Currently, only in-database ONNX embedding models are supported (not third-party embedding models) to generate vector embeddings for a hybrid vector index.
- The `HAMMING` and `JACCARD` distance metrics are currently not supported with hybrid vector indexes.
- The DML tracking support aligns with the behavior of the IVF and HNSW vector indexes for the vector index part of a hybrid vector index.
- Creating materialized views based on the outputs produced by the `VECTOR_CHUNKS`, `UTL_TO_CHUNKS`, `UTL_TO_TEXT`, and `UTL_TO_SUMMARY` functions is currently not supported.

> **Note:**
>
> With the current set of restrictions, Oracle recommends that you create a hybrid vector index in the following scenarios:
>
> - If you want to search through textual documents (with `CLOB`, `VARCHAR2`, or `BLOB` data types) in a hybrid search mode.
> - If you want to alternatively switch between keyword search and semantic search when querying textual data.
>
> If you have the data that will only be searched semantically or if you want to manually create your own third-party vector embeddings (using the `DBMS_VECTOR` and `DBMS_VECTOR_CHAIN` third-party REST APIs), then use a vector index instead of creating a hybrid vector index.

Related Topics

- Understand Hybrid Vector Indexes
  A hybrid vector index inherits all the information retrieval capabilities of Oracle Text search indexes and leverages the semantic search capabilities of Oracle AI Vector Search vector indexes.

CREATE HYBRID VECTOR INDEX

Use the `CREATE HYBRID VECTOR INDEX` SQL statement to create a hybrid vector index, which allows you to index and query documents using a combination of full-text search and vector similarity search.

Purpose

To create a class of specialized Domain Index called a hybrid vector index.

A **hybrid vector index** is an Oracle Text `SEARCH INDEX` type that combines the existing Oracle Text indexing data structures and vector indexing data structures into one unified structure. It is a single domain index that stores both text fields and vector fields for a document. Both text search and similarity search are performed on tokenized terms and vectors respectively. The two search results are combined and scored to return a unified result set.

The purpose of a hybrid vector index is to enhance search relevance of an Oracle Text index by allowing users to search by both *vectors* and *keywords*. By integrating traditional keyword-based text search with vector-based similarity search, you can improve the overall search experience and provide users with more accurate information.

Usage Notes

To create a hybrid vector index, you can provide minimal information such as:

- table or column on which you want to create the index
- in-database ONNX embedding model for generating embeddings

For cases where multiple columns or tables need to be indexed together, you can specify the `MULTI_COLUMN_DATASTORE` or `USER_DATASTORE` preference.

All other indexing parameters are predefined to facilitate the indexing of documents without requiring you to be an expert in any text processing, chunking, or embedding strategies. If required, you can modify the predefined parameters using:

- Vector search preferences for the *vector index* part of the index
- Text search preferences for the *text index* part of the index
- Index maintenance preferences for DML operations on the combined index

For detailed information on the creation process of a hybrid vector index or in general about what hybrid vector indexes are, see Understand Hybrid Vector Indexes.

> **Note:**
>
> There are some key points to note when creating and using hybrid vector indexes. See Guidelines and Restrictions for Hybrid Vector Indexes.

Syntax

```
CREATE HYBRID VECTOR INDEX [schema.]index_name ON
  [schema.]table_name(column_name)
   PARAMETERS ('paramstring')
  [FILTER BY filter_column[, filter_column]...]
  [ORDER BY oby_column[desc|asc][, oby_column[desc|asc]]...]
  [PARALLEL n];
```

Here is an example DDL specified with only the minimum required parameters.

```
CREATE HYBRID VECTOR INDEX my_hybrid_idx on
  doc_table(text_column)
  PARAMETERS('MODEL my_embed_model');
```

More comprehensive examples are given at the end of this section.

Let us explore all the required and optional indexing parameters:

[*schema*.]*index_name*
Specify the name of the hybrid vector index to create.

[*schema*.]*table_name*(*column_name*)
Specify the name of the table and column on which you want to create the hybrid vector index. You can create a hybrid vector index on one or more text columns with `VARCHAR2`, `CLOB`, and `BLOB` data types.

> **Note:**
>
> You cannot create hybrid vector indexes on a text column that uses the `IS JSON` check constraint.

Because the system can index most document formats, including HTML, PDF, Microsoft Word, and plain text, you can load a supported type into the text column. For a complete list, see Supported Document Formats.
For cases where multiple columns or tables need to be indexed together, specify a datastore preference (described later in Text search preferences).

PARAMETERS (paramstring)
Specify preferences in `paramstring`:

- **Vector Search Preferences**:

  Configures the "vector index" part of a hybrid vector index, pertaining to processing input for vector search.

  > **Note:**
  >
  > You can either pass a minimal set of parameters (the required `MODEL` and the optional `VECTOR_IDXTYPE` parameters) directly in the `PARAMETERS` clause or use a vectorizer preference to specify a complete set of vector search parameters. You cannot use both (directly set parameters along with vectorizer) in the `PARAMETERS` clause.

  – With `MODEL` and `VECTOR_IDXTYPE` directly specified:

```
CREATE HYBRID VECTOR INDEX [schema.]index_name ON
  [schema.]table_name(column_name)
   PARAMETERS ('MODEL <model_name>
                [VECTOR_IDXTYPE <vector_index_type>]')
  [FILTER BY filter_column[, filter_column]...]
  [ORDER BY oby_column[desc|asc][, oby_column[desc|asc]]...]
  [PARALLEL n];
```

  Here, `MODEL` specifies the vector embedding model that you import into the database for generating vector embeddings on your input data.

  > **Note:**
  >
  > Currently, only ONNX in-database embedding models are supported.

  `VECTOR_IDXTYPE` specifies the type of vector index to create, such as `IVF` (default) for the Inverted File Flat (IVF) vector index and `HNSW` for the Hierarchical Navigable Small World (HNSW) vector index. If you omit this parameter, then the IVF vector index is created by default.

  Creating a `LOCAL` index on an Hybrid Vector Index is supported when the underlying `index_type` is `IVF`. An example is shown below:

```
CREATE HYBRID VECTOR INDEX my_hybrid_idx on
doc_table(text_column)
parameters('MODEL my_doc_model
            VECTOR_IDXTYPE IVF')
LOCAL PARALLEL;
```

> **Caution:**
>
> Creating a `LOCAL` index on Hybrid Vector Index when the underlying `index_type` is `HNSW`, would throw an error before starting any document processing (early failure).

- With the vectorizer preference:

  A vectorizer preference is a JSON object that collectively holds all indexing parameters related to chunking (`UTL_TO_CHUNKS` or `VECTOR_CHUNKS`), embedding (`UTL_TO_EMBEDDING`, `UTL_TO_EMBEDDINGS`, or `VECTOR_EMBEDDING`), and vector index (`distance`, `accuracy`, or `vector_idxtype`).

  You use the `DBMS_VECTOR_CHAIN.CREATE_PREFERENCE` PL/SQL function to create a vectorizer preference. To create a vectorizer preference, see DBMS_VECTOR_CHAIN.CREATE_PREFERENCE.

  After creating a vectorizer preference, you can use the `VECTORIZER` parameter to pass the preference name here. For example:

```
begin
  DBMS_VECTOR_CHAIN.CREATE_PREFERENCE(
    'my_vectorizer_pref',
     dbms_vector_chain.vectorizer,
    json('{
            "vector_idxtype":  "hnsw",
            "model"         :  "my_doc_model",
            "by"            :  "words",
            "max"           :  100,
            "overlap"       :  10,
            "split"         :  "recursively"
          }'
        ));
end;
/

CREATE HYBRID VECTOR INDEX my_hybrid_idx on
  doc_table(text_column)
    parameters('VECTORIZER my_vectorizer_pref');
```

- **Text Search Preferences**:

  Configures the "Oracle Text index" part of a hybrid vector index, pertaining to processing input for keyword search.

  These parameters define the text processing and tokenization stages of a hybrid vector indexing pipeline. All these are the same set of parameters that you provide when working with Oracle Text indexes alone.

```
[DATASTORE datastore_pref]
[STORAGE storage_pref]
[MEMORY memsize]
[STOPLIST stoplist]
[LEXER lexer_pref]
```

```
[FILTER filter_pref]
[WORDLIST wordlist_pref]
[SECTION GROUP section_group]
```

DATASTORE datastore_pref

Specify the name of your datastore preference. Use the datastore preference to specify the local or remote location where your source files are stored.

If you want to index multiple columns or tables together, see MULTI_COLUMN_DATASTORE and USER_DATASTORE.

For a complete list of all datastore preferences, see Datastore Types.

Default: `DIRECT_DATASTORE`

STORAGE storage_pref

Specify the name of your storage preference for an Oracle Text search index. Use the storage preference to specify how the index tables are stored. See Storage Types.

MEMORY memsize

Specify the amount of run-time memory to use for indexing.

```
memsize = number[K|M|G]
```

K is for kilobytes, M is for megabytes, and G is for gigabytes.

The value you specify for `memsize` must be between 1M and the value of `MAX_INDEX_MEMORY` in the `CTX_PARAMETERS` view. To specify a memory size larger than the `MAX_INDEX_MEMORY`, you must reset this parameter with `CTX_ADM.SET_PARAMETER` to be larger than or equal to `memsize`. See CTX_ADM.SET_PARAMETER.

The default for Oracle Text search index is 500 MB.

The `memsize` parameter specifies the amount of memory Oracle Text uses for indexing before flushing the index to disk. Specifying a large amount memory improves indexing performance because there are fewer I/O operations and improves query performance and maintenance, because there is less fragmentation.

Specifying smaller amounts of memory increases disk I/O and index fragmentation, but might be useful when run-time memory is scarce.

STOPLIST stoplist

Specify the name of your stoplist. Use stoplist to identify words that are not to be indexed. See CTX_DDL.CREATE_STOPLIST.

Default: `CTXSYS.DEFAULT_STOPLIST`

LEXER lexer_pref

Specify the name of your lexer or multilexer preference. Use the lexer preference to identify the language of your text and how text is tokenized for indexing. See Lexer Types.

Default: `CTXSYS.DEFAULT_LEXER`

FILTER filter_pref

Specify the name of your filter preference. Use the filter preference to specify how to filter formatted documents to plain text. See Filter Types.

The default for binary text columns is `NULL_FILTER`. The default for other text columns is `AUTO_FILTER`.

WORDLIST wordlist_pref

Specify the name of your wordlist preference. Use the wordlist preference to enable features such as fuzzy, stemming, and prefix indexing for better wildcard searching. See Wordlist Type.

SECTION GROUP section_group

Specify the name of your section group. Use section groups to create sections in structured documents. See CTX_DDL.CREATE_SECTION_GROUP.

Default: `NULL_SECTION_GROUP`

- **Index Maintenance Preferences**:

  Configures the DML operations on the entire hybrid vector index, that is, how to synchronize and optimize the index.

  Because a hybrid vector index is basically an Oracle Text search index type, so all maintenance-specific capabilities of an Oracle Text index are applicable.

```
[MAINTENANCE AUTO | MAINTENANCE MANUAL]
[SYNC (MANUAL | EVERY "interval-string" | ON COMMIT)]
[OPTIMIZE (MANUAL | AUTO_DAILY | EVERY "interval-string")]
```

MAINTENANCE AUTO | MAINTENANCE MANUAL

Specify the maintenance type for synchronization of a hybrid vector index when there are inserts, updates, or deletes to the base table. The maintenance type specified for an index applies to all index partitions.

You can specify one of the following maintenance types:

Maintenance Type	Description
`MAINTENANCE AUTO`	This method sets your index to automatic maintenance, that is, the index is automatically synchronized in the background at optimal intervals. You do not need to manually configure a `SYNC` type or set any synchronization interval. The background mechanism automatically determines the synchronization interval and schedules background `SYNC.INDEX` operations by tracking the DML queue.
`MAINTENANCE MANUAL`	This method sets your index to manual maintenance. This is a non-automatic maintenance (synchronization) mode in which you can specify `SYNC` types, such as `MANUAL`, `EVERY`, or `ON COMMIT`.

SYNC (MANUAL | EVERY "interval-string" | ON COMMIT)

Specify the `SYNC` type for synchronization of a hybrid vector index when there are inserts, updates, or deletes to the base table. These `SYNC` settings are applicable only to the indexes that are set to manual maintenance.

> **Note:**
>
> By default, a hybrid vector index runs in an automatic maintenance mode (`MAINTENANCE AUTO`), which means that your DMLs are automatically synchronized into the index in the background at optimal intervals. Therefore, you do not need to manually configure a `SYNC` type for maintaining a hybrid vector index. However, if required, you can do so if you want to modify the default settings for an index.

You can specify one of the `SYNC` methods:

SYNC Type	Description
`MANUAL`	With this method, automatic synchronization is not provided. You must manually synchronize the index using `CTX_DDL.SYNC_INDEX`.
`EVERY` *interval-string*	Automatically synchronize the index at a regular interval specified by the value of *interval-string*, which takes the same syntax as that for scheduler jobs. Automatic synchronization using `EVERY` requires that the index creator have `CREATE JOB` privileges. Ensure that *interval-string* is set to a considerable time period so that any previous synchronization jobs will have completed. Otherwise, the synchronization job may stop responding. The *interval-string* argument must be enclosed in double quotation marks (" ").
`ON COMMIT`	Synchronize the index immediately after a commit. The commit does not return until the sync is complete. The operation uses the memory specified with the *memory* parameter. This sync type works best when the `STAGE_ITAB` index option is enabled, otherwise it causes significant fragmentation of the main index, requiring frequent `OPTIMIZE` calls.

With automatic (`EVERY`) synchronization, you can specify memory size and parallel synchronization. You can define repeating schedules in the *interval-string* argument using calendaring syntax values. These values are described in *Oracle Database PL/SQL Packages and Types Reference*.

Syntax:

```
SYNC [EVERY "interval-string"] MEMORY mem_size PARALLEL paradegree
```

For example, to sync the index at an interval of 20 seconds:

```
SYNC [EVERY "freq=secondly;interval=20"] MEMORY 500M PARALLEL 2
```

OPTIMIZE (MANUAL | AUTO_DAILY | EVERY "interval-string)

Specify `OPTIMIZE` to enable automatic background index optimization of a hybrid vector index. You can specify any one of the following `OPTIMIZE` methods:

OPTIMIZE Type	Description
`MANUAL`	Provides no automatic optimization. You must manually optimize the index with `CTX_DDL.OPTIMIZE_INDEX`.
`AUTO_DAILY`	This is the default setting. With `OPTIMIZE (AUTO_DAILY)`, the optimize `FULL` job is scheduled to run midnight from 12 A.M. local time everyday.
`EVERY "interval-string"`	Automatically runs optimize `token` at a regular interval specified by the value *interval-string*, which takes the same syntax as the scheduler jobs. Ensure that *interval-string* is set to a considerable time period so that the previous optimize jobs are complete; otherwise, the optimize job might stop responding. *interval-string* must be enclosed in double quotes, and any single quote within *interval-string* must be preceded by the escape character with another single quote.

With `AUTO_DAILY | EVERY "interval-string"` setting, you can specify parallel optimization.

Syntax:

```
OPTIMIZE [AUTO_DAILY | EVERY "interval-string"] PARALLEL paradegree ...
```

For example, to optimize the index at an interval of 20 minutes:

```
OPTIMIZE [EVERY "freq=minutely;interval=20"] PARALLEL 2
```

FILTER BY filter_column

Specify the structured indexed column on which a range or equality predicate in the `WHERE` clause of a mixed query will operate. You can specify one or more structured columns for `filter_column`, on which the relational predicates are expected to be specified along with the `CONTAINS()` predicate in a query.

- You can use these relational operators:

  <, <=, =, >=, >, `between`, and `LIKE` (for `VARCHAR2`)

- These columns can only be of CHAR, NUMBER, DATE, VARCHAR2, or RAW type. Additionally, CHAR, VARCHAR2 and VARCHAR2 types are supported only if the maximum length is specified and does not exceed 249 bytes.

  If the maximum length of a CHAR or VARCHAR2 column is specified in characters, for example, VARCHAR2 (50 CHAR), then it cannot exceed FLOOR (249/max_char_width), where max_char_width is the maximum width of any character in the database character set.

  For example, the maximum specified column length cannot exceed 62 characters, if the database character set is AL32UTF8. The ADT attributes of supported types (CHAR, NUMBER, DATE, VARCHAR2, or RAW) are also allowed.

  An error is raised for all other data types. Expressions, for example, func(cola), and virtual columns are not allowed.

- txt_column is allowed in the FILTER BY column list.
- DML operations on FILTER BY columns are always transactional.

ORDER BY oby_column[desc|asc]
Specify one or more structured indexed columns by which you want to sort query results. You can specify a list of structured *oby_columns*. These columns can only be of CHAR, NUMBER, DATE, VARCHAR2, or RAW type. VARCHAR2 and RAW columns longer than 249 bytes are truncated to the first 249 bytes. Expressions, for example func(cola), and virtual columns are not allowed. The order of the specified columns matters. The ORDER BY clause in a query can contain:

- The entire ordered ORDER BY columns
- Only the prefix of the ordered ORDER BY columns
- The score followed by the prefix of the ordered ORDER BY columns

DESC sorts the results in a descending order (from highest to lowest), while ASC (default) sorts the results in an ascending order (from lowest to highest).

[PARALLEL n]
Parallel indexing can improve index performance when you have multiple CPUs. To create an index in parallel, use the PARALLEL clause with a parallel degree.
Optionally specifies the parallel degree for parallel indexing. The actual degree of parallelism might be smaller depending on your resources. You can use this parameter on nonpartitioned tables. However, creating a nonpartitioned index in parallel does not turn on parallel query processing. Parallel indexing is supported for creating a local partitioned index.
The indexing memory size specified in the parameter clause applies to each parallel worker. For example, if indexing memory size is specified in the parameter clause as 500M and parallel degree is specified as 2, then you must ensure that there is at least 1GB of memory available for indexing.

Examples

- **With vector search preferences directly specified**:

  In this example, only the required parameter model is specified in the PARAMETERS clause:

```
CREATE HYBRID VECTOR INDEX my_hybrid_idx on
  doc_table(text_column)
  parameters('MODEL my_doc_model');
```

In this example, both the parameters `model` and `vector_idxtype` are specified:

```
CREATE HYBRID VECTOR INDEX my_hybrid_idx on
  doc_table(text_column)
  parameters('MODEL my_doc_model
              VECTOR_IDXTYPE HNSW');
```

- **With vector search preferences specified using VECTORIZER**:

  In this example, the `vectorizer` parameter is used in the `PARAMETERS` clause to specify the `my_vectorizer_spec` preference:

```
begin
  DBMS_VECTOR_CHAIN.CREATE_PREFERENCE(
    'my_vectorizer_spec',
     dbms_vector_chain.vectorizer,
    json('{"vector_idxtype" :  "hnsw",
           "model"          :  "my_doc_model",
           "by"             :  "words",
           "max"            :  100,
           "overlap"        :  10,
           "split"          :  "recursively"}'));
end;
/

CREATE HYBRID VECTOR INDEX my_hybrid_idx on
  doc_table(text_column)
  parameters('VECTORIZER my_vectorizer_spec');
```

- **With text search and vector search preferences directly specified**:

  In this example, only the required Vector Search parameter `MODEL` is specified in the `PARAMETERS` clause. Text Search parameters are also specified:

```
CREATE HYBRID VECTOR INDEX my_hybrid_idx on
  doc_table(text_column)
  parameters('MODEL my_doc_model
              DATASTORE my_datastore
              STORAGE my_storage
              STOPLIST my_stoplist
              LEXER my_lexer')
  ORDER BY docid asc;
```

- **With text search and index maintenance preferences directly specified and vector search preferences specified using VECTORIZER**:

  In this example, the `VECTORIZER` parameter is used to specify the `my_vectorizer_spec` preference that holds vector search parameters. All the Text Search and Index Maintenance preferences are directly specified.

```
begin
  DBMS_VECTOR_CHAIN.CREATE_PREFERENCE(
    'my_vectorizer_spec',
     dbms_vector_chain.vectorizer,
    json('{
           "vector_idxtype" :  "hnsw",
           "model"          :  "my_doc_model",
```

```
            "by"             :  "words",
            "max"            :  100,
            "overlap"        :  10,
            "split"          :  "recursively"
         }'
        ));
end;
/

CREATE HYBRID VECTOR INDEX my_hybrid_idx on
  doc_table(text_column)
  parameters('VECTORIZER my_vectorizer_spec
              DATASTORE my_datastore
              STORAGE my_storage
              MEMORY 128M
              MAINTENANCE AUTO
              OPTIMIZE AUTO_DAILY
              STOPLIST my_stoplist
              LEXER my_lexer
              FILTER my_filter
              WORDLIST my_wordlist
              SECTION GROUP my_section_group')
  FILTER BY category, author
  ORDER BY score(10), score(20) desc
  PARALLEL 3;
```

Related Topics

- Perform Hybrid Search
- Query Hybrid Vector Indexes End-to-End Example

ALTER INDEX

Use the `ALTER INDEX` SQL statement to modify an existing hybrid vector index.

Purpose

To make changes to hybrid vector indexes.

Syntax

```
ALTER INDEX [schema.]index_name REBUILD
  [PARAMETERS('UPDATE VECTOR INDEX')]
  [PARALLEL n];
```

> **Note:**
>
> - If you do not specify the `PARAMETERS` clause, then all parts of the hybrid vector index (both Oracle Text index and vector index) are recreated with existing preference settings.
> - Renaming hybrid vector indexes using the `ALTER INDEX RENAME` syntax is not supported.

[*schema*.]*index_name*
Specifies name of the hybrid vector index that you want to modify.

PARAMETERS(UPDATE VECTOR INDEX)
Recreates only the vector index part of a hybrid vector index with the original preference settings.

PARALLEL
Specifies parallel indexing, as described for the `CREATE HYBRID VECTOR INDEX` statement. For detailed information on the `PARALLEL` clause, see CREATE HYBRID VECTOR INDEX.

Examples

Here are some examples on how you can modify existing hybrid vector indexes:

- **To rebuild all parts of a hybrid vector index**:

  Use the following syntax to rebuild all parts of a hybrid vector index (both Oracle Text index and vector index) with the original preference settings:

  Syntax:

  ```
  ALTER INDEX index_name REBUILD [PARALLEL n];
  ```

  Note that you do not need to specify any `PARAMETERS` clause when rebuilding both parts of a hybrid vector index.

  Example:

  ```
  ALTER INDEX my_hybrid_idx REBUILD;

  SELECT (select id from doc_table where rowid = jt.doc_rowid) as doc,
         jt.chunk
  FROM JSON_TABLE(
           DBMS_HYBRID_VECTOR.SEARCH(
             json(
               '{ "hybrid_index_name" : "my_hybrid_idx",
                  "vector" :
                   { "search_text"    : "vector based search capabilities",
                     "search_mode"    : "CHUNK"
                   },
                  "return" :
                   { "topN"           : 10 }
               }')
             ),
             '$[*]' COLUMNS doc_rowid  PATH '$.rowid',
                            chunk      PATH '$.chunk_text') jt;
  ```

- **To rebuild only the vector index part**:

  Use the following syntax to rebuild only the vector index part of a hybrid vector index with the original preference settings:

  Syntax:

  ```
  ALTER INDEX index_name REBUILD
    PARAMETERS('UPDATE VECTOR INDEX') [PARALLEL n];
  ```

Example:

```
ALTER INDEX my_hybrid_idx REBUILD
  PARAMETERS('UPDATE VECTOR INDEX') PARALLEL 3;

SELECT (select id from doc_table where rowid = jt.doc_rowid) as doc,
       jt.chunk
FROM JSON_TABLE(
         DBMS_HYBRID_VECTOR.SEARCH(
           json(
             '{ "hybrid_index_name" : "my_hybrid_idx",
                "vector" :
                 { "search_text"    : "vector based search capabilities",
                   "search_mode"    : "CHUNK"
                 },
                "return" :
                 { "topN"           : 10 }
             }')
           ),
           '$[*]' COLUMNS doc_rowid  PATH '$.rowid',
                          chunk      PATH '$.chunk_text') jt;
```

Related Topics

- *Oracle Text Reference*

Vector Indexes in a Globally Distributed Database

Inverted File Flat (IVF) index and Hierarchical Navigable Small World (HNSW) index are supported on sharded tables in a distributed database; however there are some considerations.

> **Note:**
>
> - Global indexes are not supported on sharded tables; however, this limitation does not exist for the global HNSW and IVF index.
> - Hybrid Vector Indexes (HVI) are not currently supported on sharded tables.
> - GDSCTL commands `MOVE CHUNK`, `ADD CDB`, and `ADD SHARD` will raise ORA-05118 if there are global vector indexes on sharded tables. Drop the global vector indexes before performing these operations.

Inverted File Flat Index

Inverted File Flat Index (IVF Flat or simply IVF) is a partitioned-based index that lets you balance high-search quality with reasonable speed.

You can create a local IVF index on vector columns in a sharded table. There is no syntax change required.

- IVF indexes and HNSW indexes on a sharded table must be created on the shard catalog database with `SHARD DDL` enabled.

- The `CREATE INDEX` command is propagated as is to all of the shards by the shard coordinator. The `CREATE INDEX` clause scope is the shard.

There is no syntax change to create an IVF index on a sharded table, when compared to the syntax to create an IVF index on a non-sharded table.

```
CREATE VECTOR INDEX ivf_image
     ON houses (image)
 ORGANIZATION NEIGHBOR PARTITIONS WITH TARGET ACCURACY 95
 DISTANCE EUCLIDEAN PARAMETERS
 (type IVF, NEIGHBOR PARTITIONS 1000) PARALLEL 16;
```

Hierarchical Navigable Small World Index

There is no syntax change to create a Hierarchical Navigable Small World (HNSW) index on a sharded table, when compared to the syntax to create an HNSW index on a non-sharded table.

```
CREATE VECTOR INDEX hnsw_image
     ON houses (image)
 ORGANIZATION INMEMORY NEIGHBOR GRAPH
 WITH TARGET ACCURACY 95;
```

7

Use SQL Functions for Vector Operations

There are a number of SQL functions and operators that you can use with vectors in Oracle AI Vector Search.

> **Note:**
>
> You can also use PL/SQL packages to perform similar operations and additional tasks. See Vector Search PL/SQL Packages.

- Vector Distance Functions and Operators
  A vector distance function takes in two vector operands and a distance metric to compute a mathematical distance between those two vectors, based on the distance metric provided. You can optionally use shorthand distance functions and operators instead of their corresponding distance functions.
- Chunking and Vector Generation Functions
  Oracle AI Vector Search offers Vector Utilities, which provide the `VECTOR_CHUNKS` and `VECTOR_EMBEDDING` SQL functions for chunking data and generating vector embedding, respectively.
- Constructors, Converters, Descriptors, and Arithmetic Operators
  Other basic vector operations for Oracle AI Vector Search involve creating, converting, and describing vectors.
- JSON Compatibility with the VECTOR Data Type
  The JSON data type supports the `VECTOR` type as a JSON scalar type. A `VECTOR` instance is convertible to a JSON type instance and vice versa using the JSON constructor and the `VECTOR` constructor, respectively.

Vector Distance Functions and Operators

A vector distance function takes in two vector operands and a distance metric to compute a mathematical distance between those two vectors, based on the distance metric provided. You can optionally use shorthand distance functions and operators instead of their corresponding distance functions.

Distances determine similarity or dissimilarity between vectors.

- Vector Distance Metrics
  Measuring distances in a vector space is at the heart of identifying the most relevant results for a given query vector. That process is very different from the well-known keyword filtering in the relational database world.
- VECTOR_DISTANCE
  `VECTOR_DISTANCE` is the main function that you can use to calculate the distance between two vectors.

- L1_DISTANCE
  `L1_DISTANCE` is a shorthand version of the `VECTOR_DISTANCE` function that calculates the Manhattan distance between two vectors. It takes two vectors as input and returns the distance between them as a `BINARY_DOUBLE`.
- L2_DISTANCE
  `L2_DISTANCE` is a shorthand version of the `VECTOR_DISTANCE` function that calculates the Euclidean distance between two vectors. It takes two vectors as input and returns the distance between them as a `BINARY_DOUBLE`.
- COSINE_DISTANCE
  `COSINE_DISTANCE` is a shorthand version of the `VECTOR_DISTANCE` function that calculates the cosine distance between two vectors. It takes two vectors as input and returns the distance between them as a `BINARY_DOUBLE`.
- INNER_PRODUCT
  `INNER_PRODUCT` calculates the inner product of two vectors. It takes two vectors as input and returns the inner product as a `BINARY_DOUBLE`. `INNER_PRODUCT(<expr1>, <expr2>)` is equivalent to `-1 * VECTOR_DISTANCE(<expr1>, <expr2>, DOT)`.
- HAMMING_DISTANCE
  `HAMMING_DISTANCE` is a shorthand version of the `VECTOR_DISTANCE` function that calculates the hamming distance between two vectors. It takes two vectors as input and returns the distance between them as a `BINARY_DOUBLE`.
- JACCARD_DISTANCE
  `JACCARD_DISTANCE` is a shorthand version of the `VECTOR_DISTANCE` function that calculates the jaccard distance between two vectors. It takes two `BINARY` vectors as input and returns the distance between them as a `BINARY_DOUBLE`.

Related Topics

- Perform Exact Similarity Search
  A similarity search looks for the relative order of vectors compared to a query vector. Naturally, the comparison is done using a particular distance metric but what is important is the result set of your top closest vectors, not the distance between them.
- Perform Approximate Similarity Search Using Vector Indexes
  For a vector search to be useful, it needs to be fast and accurate. Approximate similarity searches seek a balance between these goals.

Vector Distance Metrics

Measuring distances in a vector space is at the heart of identifying the most relevant results for a given query vector. That process is very different from the well-known keyword filtering in the relational database world.

When working with vectors, there are several ways you can calculate distances to determine how similar, or dissimilar, two vectors are. Each distance metric is computed using different mathematical formulas. The time it takes to calculate the distance between two vectors depends on many factors, including the distance metric used as well as the format of the vectors themselves, such as the number of vector dimensions and the vector dimension formats. Generally it's best to match the distance metric you use to the one that was used to train the vector embedding model that generated the vectors.

- Euclidean and Euclidean Squared Distances
  Euclidean distance reflects the distance between each of the vectors' coordinates being compared—basically the straight-line distance between two vectors. This is calculated

using the Pythagorean theorem applied to the vector's coordinates `(SQRT(SUM((xi-yi)^2)))`.

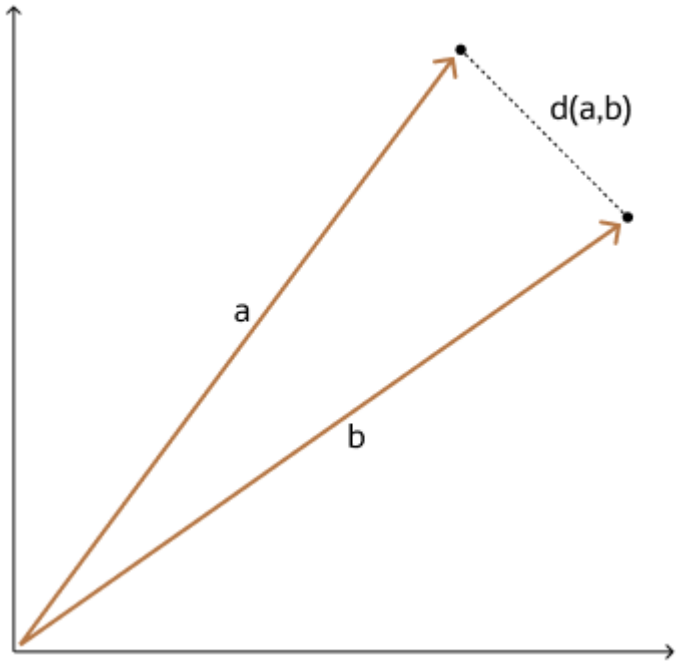
- Cosine Similarity
  One of the most widely used similarity metric, especially in natural language processing (NLP), is cosine similarity, which measures the cosine of the angle between two vectors.
- Dot Product Similarity
  The dot product similarity of two vectors can be viewed as multiplying the size of each vector by the cosine of their angle. The corresponding geometrical interpretation of this definition is equivalent to multiplying the size of one of the vectors by the size of the projection of the second vector onto the first one, or vice versa.
- Manhattan Distance
  This metric is calculated by summing the distance between the dimensions of the two vectors that you want to compare.
- Hamming Distance
  The Hamming distance between two vectors represents the number of dimensions where they differ.
- Jaccard Similarity
  The Jaccard similarity is used to determine the share of significant (non-zero) dimensions (bit's position) common between two `BINARY` vectors.
- Custom Distance Function
  JavaScript user-defined functions can be used to define a custom vector distance. This provides greater flexibility in the types of distance equations that can be employed, extending vector search functionality to a broader range of use cases.

Euclidean and Euclidean Squared Distances

Euclidean distance reflects the distance between each of the vectors' coordinates being compared—basically the straight-line distance between two vectors. This is calculated using the Pythagorean theorem applied to the vector's coordinates `(SQRT(SUM((xi-yi)^2)))`.

This metric is sensitive to both the vector's size and it's direction.

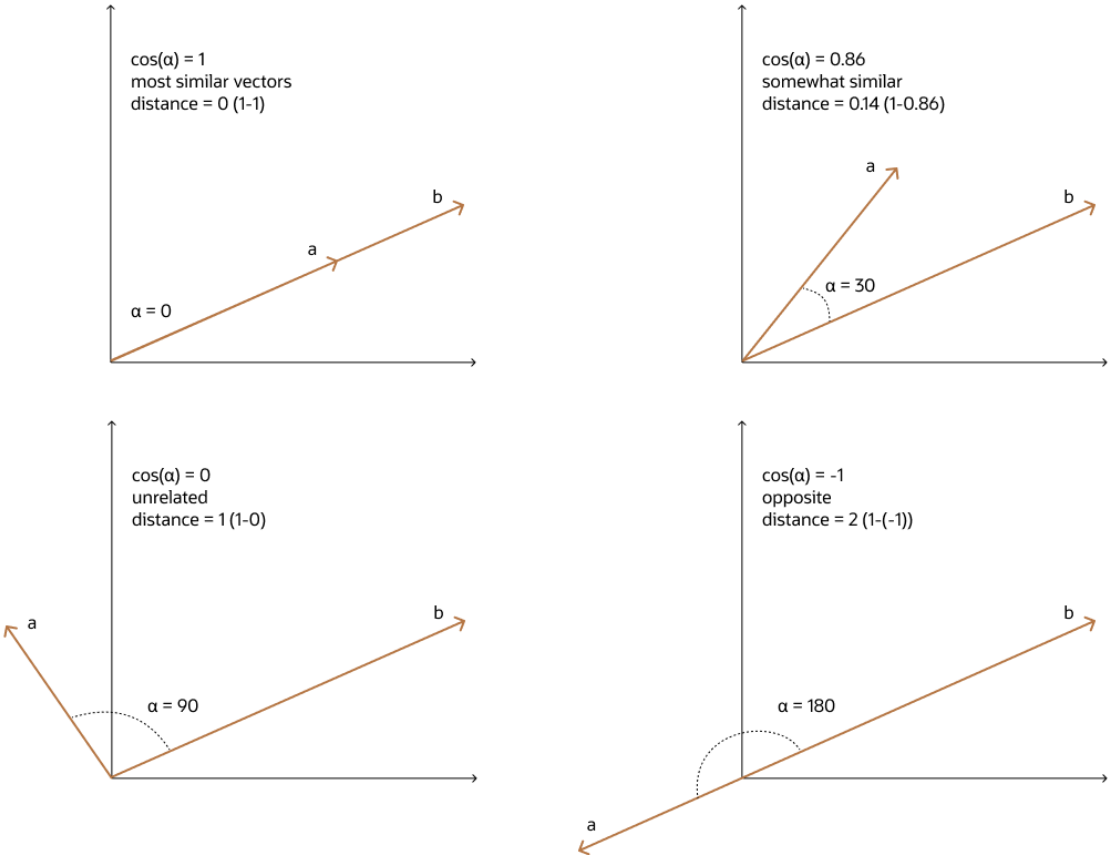
With Euclidean distances, comparing squared distances is equivalent to comparing distances. So, when ordering is more important than the distance values themselves, the Squared Euclidean distance is very useful as it is faster to calculate than the Euclidean distance (avoiding the square-root calculation).



Cosine Similarity

One of the most widely used similarity metric, especially in natural language processing (NLP), is cosine similarity, which measures the cosine of the angle between two vectors.

The smaller the angle, the more similar are the two vectors. Cosine similarity measures the similarity in the direction or angle of the vectors, ignoring differences in their size (also called *magnitude*). The smaller the angle, the bigger is its cosine. So the cosine distance and the cosine similarity have an inverse relationship. While cosine distance measures how different two vectors are, cosine similarity measures how similar two vectors are.



Dot Product Similarity

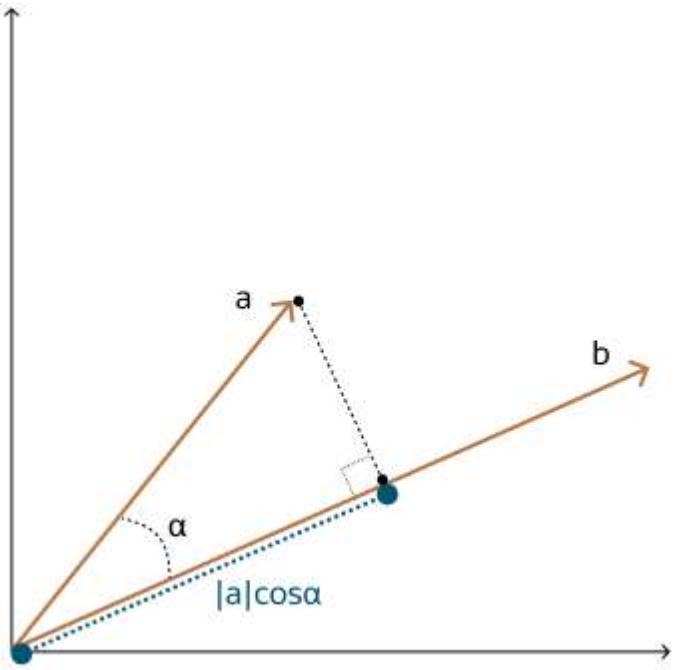
The dot product similarity of two vectors can be viewed as multiplying the size of each vector by the cosine of their angle. The corresponding geometrical interpretation of this definition is equivalent to multiplying the size of one of the vectors by the size of the projection of the second vector onto the first one, or vice versa.

As illustrated in the following diagram, you project one vector on the other and multiply the resulting vector sizes.

Incidentally, this is equivalent to the sum of the products of each vector's coordinate. Often, you do not have access to the cosine of the two vector's angle, hence this calculation is easier.

Larger dot product values imply that the vectors are more similar, while smaller values imply that they are less similar. Compared to using Euclidean distance, using the dot product similarity is especially useful for high-dimensional vectors.

Note that normalizing vectors and using the dot product similarity is equivalent to using cosine similarity. There are cases where dot product similarity is faster to evaluate than cosine similarity, and conversely where cosine similarity is faster than dot product similarity. A normalized vector is created by dividing each dimension by the norm (or length) of the vector, such that the norm of the normalized vector is equal to 1.

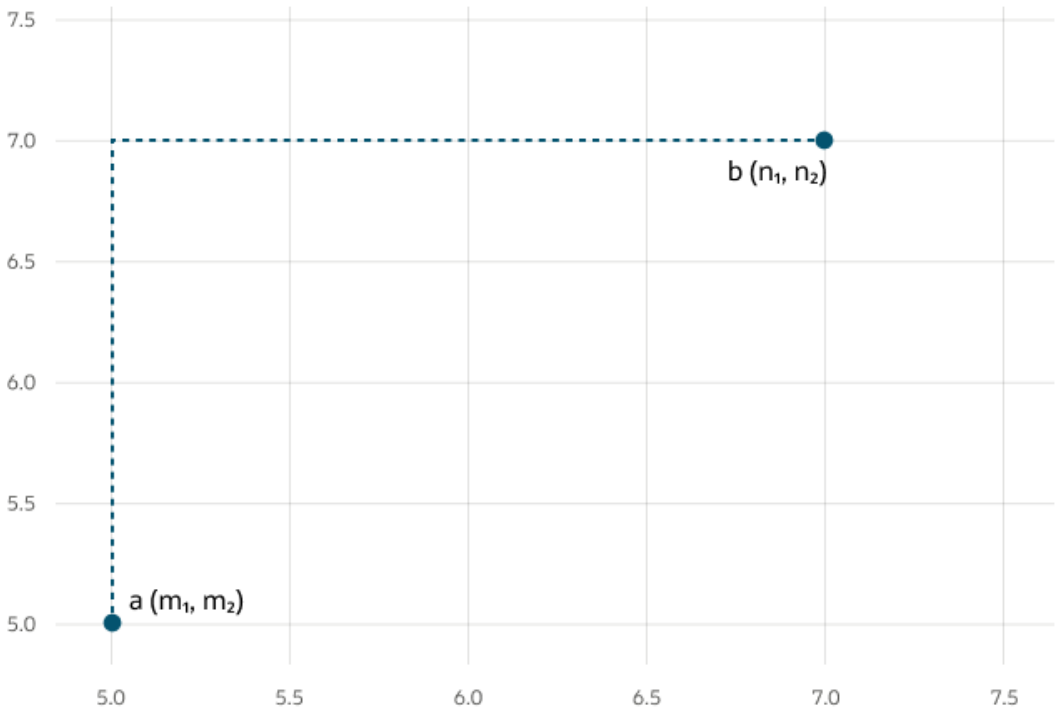


Manhattan Distance

This metric is calculated by summing the distance between the dimensions of the two vectors that you want to compare.

Imagine yourself in the streets of Manhattan trying to go from point A to point B. A straight line is not possible.

This metric is most useful for vectors describing objects on a uniform grid, such as city blocks, power grids, or a chessboard. It can be useful for higher dimensional vector spaces too. Compared to the Euclidean metric, the Manhattan metric is faster for calculations and you can use it advantageously for higher dimensional vector spaces.



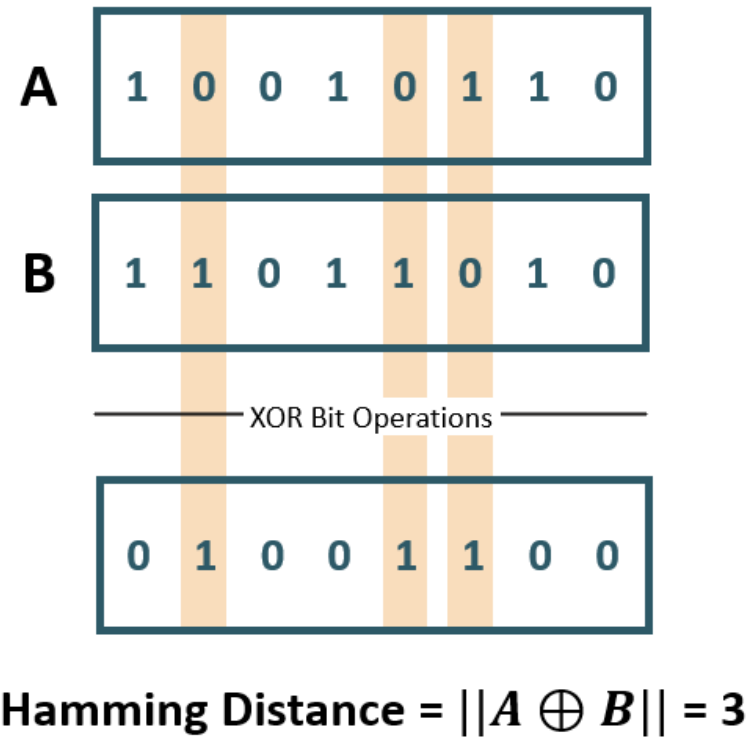
Hamming Distance

The Hamming distance between two vectors represents the number of dimensions where they differ.

For example, when using binary vectors, the Hamming distance between two vectors is the number of bits you must change to change one vector into the other. To compute the Hamming distance between two vectors *A* and *B*, you need to:

- Compare the position of each bit in the sequence. You do this by using an `exclusive or` (also called the XOR bit operation) between *A* and *B*. This operation outputs 1 if the bits in the sequence do not match, and 0 otherwise.
- Count the number of '1's in the resulting vector, the outcome of which is called the Hamming weight or norm of that vector.

It's important to note that the bit strings need to be of equal length for the comparison to make sense. The Hamming metric is mainly used with binary vectors for error detection over networks.



Hamming Distance = $||A \oplus B||$ = 3

Jaccard Similarity

The Jaccard similarity is used to determine the share of significant (non-zero) dimensions (bit's position) common between two `BINARY` vectors.

The Jaccard similarity is only applicable to `BINARY` vectors and only the non-zero bits of each vector are considered.

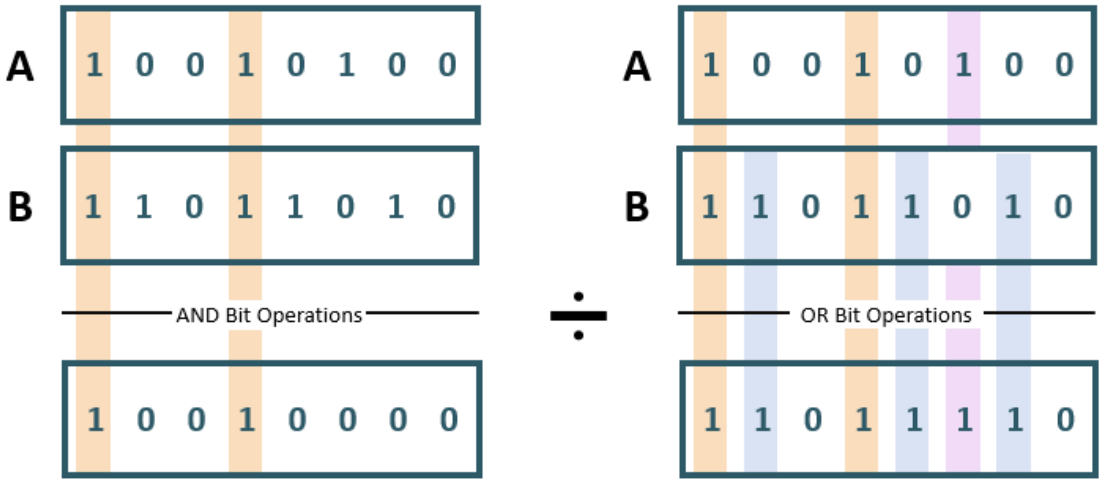
The Jaccard similarity between vectors *A* and *B* is the calculation of the Hamming weight (norm, or the number of '1's in the resulting vector) of the result of an `AND` bit operation between *A* and *B*, divided by the Hamming weight of the result of an `OR` bit operation between *A* and *B*.

As shown in the included diagram of vectors *A* and *B*:

- The `AND` bit operation outputs a 1 if the bits in the sequence match a 1, and 0 otherwise.
- The `OR` bit operation outputs a 1 if at least one of the bits in the sequence matches a 1, and 0 otherwise.

The result of the calculation is from 0 to 1, where results approaching 1 are more similar. A result of 0 means that the two vectors share no non-zero attributes while a result of 1 indicates the two vectors share identical sets of non-zero attributes. In the included diagram, the two vectors share 33% of the significant attributes.

While the Jaccard similarity indicates how similar two vectors are, the Jaccard distance indicates the *dissimilarity* between the vectors. The Jaccard distance can be found by subtracting the Jaccard similarity from 1. For example, two vectors with a Jaccard similarity of *0.25* have a Jaccard distance of *0.75*. When determining distance, the meaning of the result is opposite to that of a similarity calculation. A result of 0 indicates that the two vectors are identical while a result of 1 indicates the vectors are completely disjoint, sharing no common elements.



$$\text{Jaccard Similarity} = \frac{\|A \cap B\|}{\|A \cup B\|} = \frac{2}{6} = 0.33$$

$$\text{Jaccard Distance} = 1 - \text{Jaccard Similarity} = 1 - 0.33 = 0.67$$

Custom Distance Function

JavaScript user-defined functions can be used to define a custom vector distance. This provides greater flexibility in the types of distance equations that can be employed, extending vector search functionality to a broader range of use cases.

A custom distance function is created by a user-defined JavaScript function defined in a Multilingual Engine (MLE) inline call specification. The signature of the function must match the signature of existing built-in distance functions. As in, it must accept exactly two arguments of type `VECTOR` and return a `BINARY_DOUBLE`. The function signature must also include the `DETERMINISTIC` keyword. The following function definition provides an example of a custom distance function, in this case implementing the Euclidean Squared distance:

```
CREATE OR REPLACE FUNCTION euclidean_sq_vector_distance("a" VECTOR, "b"
VECTOR)
RETURN BINARY_DOUBLE
DETERMINISTIC PARALLEL_ENABLE
AS MLE LANGUAGE JAVASCRIPT PURE
{{
    let len = a.length;
    let sum = 0;
    for(let i = 0; i < len; i++) {
        const tmp = a[i] - b[i];
        sum += tmp * tmp;
    }
    return sum;
}};
/
```

Custom distance functions can be used with HNSW vector indexes. If the degree of parallelism in the vector index is greater than 1, the custom distance function must include the `PARALLEL_ENABLE` clause. Upon index creation, a custom distance can be specified by name in the `DISTANCE` clause. In queries, the custom distance can be used in the `ORDER BY` clause and in the `SELECT` list. The distance function tied to a vector index can be viewed by querying the `VECSYS.VECTOR$INDEX` view.

When used in the creation of an HNSW index, the `PURE` keyword must be specified in the MLE call specification. The `PURE` clause indicates that the JavaScript program should be run in a restricted execution context, which guarantees that the code will not modify stateful objects, such as database tables or PL/SQL packages, regardless of database privileges currently in effect. A user-defined function used to create a custom distance metric only handles computations on function inputs, which do not require access to database state. Restricted contexts provide an extra layer of security by prohibiting unwanted database modifications. For more information on restricted execution contexts and the `PURE` keyword, see *Oracle Database JavaScript Developer's Guide*.

In order to use a vector index dependent on a custom distance function, you must have `EXECUTE` privileges on the function specified during index creation. You must also have `EXECUTE` privileges on `JAVASCRIPT`. For vector indexes, only definer's rights are supported.

If the distance function is modified, the associated vector index will be in an `UNUSABLE` state.

> **Note:**
>
> The use of custom distance functions with IVF indexes is not currently supported.

Use the previously created custom distance function, `euclidean_sq_custom_distance`, to first create a vector index:

```
CREATE TABLE custom_dist_tab( id NUMBER, data_vector VECTOR(2, FLOAT32));

INSERT INTO custom_dist_tab VALUES (1, vector('[1.1,2.2]', 2, float32));
INSERT INTO custom_dist_tab VALUES (2, vector('[2.2,3.3]', 2, float32));
INSERT INTO custom_dist_tab VALUES (3, vector('[3.3,4.4]', 2, float32));
INSERT INTO custom_dist_tab VALUES (4, vector('[4.4,5.5]', 2, float32));
INSERT INTO custom_dist_tab VALUES (5, vector('[5.5,6.6]', 2, float32));

CREATE VECTOR INDEX cust_dist_idx_hnsw ON custom_dist_tab (data_vector)
ORGANIZATION INMEMORY
NEIGHBOR GRAPH WITH TARGET ACCURACY 95
DISTANCE CUSTOM EUCLIDEAN_SQ_VECTOR_DISTANCE
PARALLEL 3;
```

The custom distance function can be referenced in similarity search queries in the `ORDER BY` clause or in the `SELECT` list:

```
SELECT data_vector
FROM custom_dist_tab
ORDER BY euclidean_sq_vector_distance(data_vector, VECTOR('[1, 2]'))
FETCH FIRST 5 ROWS ONLY;

SELECT
```

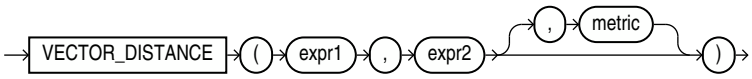
```
  data_vector,
  euclidean_sq_vector_distance(data_vector, VECTOR('[1, 2]')) edist
FROM custom_dist_tab
ORDER BY edist
FETCH FIRST 5 ROWS ONLY;
```

> **See Also:**
>
> - *Oracle Database JavaScript Developer's Guide* for information about using inline call specifications to publish JavaScript functions with the Multilingual Engine (MLE)

VECTOR_DISTANCE

`VECTOR_DISTANCE` is the main function that you can use to calculate the distance between two vectors.

Syntax



Purpose

`VECTOR_DISTANCE` takes two vectors as parameters. You can optionally specify a distance metric to calculate the distance. If you do not specify a distance metric, then the default distance metric is cosine. If the input vectors are `BINARY` vectors, the default metric is hamming.

You can optionally use the following shorthand vector distance functions:

- `L1_DISTANCE`
- `L2_DISTANCE`
- `COSINE_DISTANCE`
- `INNER_PRODUCT`
- `HAMMING_DISTANCE`
- `JACCARD_DISTANCE`

All the vector distance functions take two vectors as input and return the distance between them as a `BINARY_DOUBLE`.

Note the following caveats:

- If you specify a metric as the third argument, then that metric is used.
- If you do not specify a metric, then the following rules apply:
  - If there is a single column referenced in *`expr1`* and *`expr2`* as in: `VECTOR_DISTANCE(vec1, :bind)`, and if there is a vector index defined on *`vec1`*, then the metric used when defining the vector index is used.

If no vector index is defined on *vec1*, then the COSINE metric is used.

  - If there are multiple columns referenced in *expr1* and *expr2* as in: VECTOR_DISTANCE(vec1, vec2), or VECTOR_DISTANCE(vec1+vec2, :bind), then for all indexed columns, if their metrics used in the definitions of the indexes are the same, then that metric is used.

    On the other hand, if the indexed columns do not have a common metric, or none of the columns have an index defined, then the COSINE metric is used.

- In a similarity search query, if *expr1* or *expr2* reference an indexed column and you specify a distance metric that conflicts with the metric specified in the vector index, then the vector index is not used and the metric you specified is used to perform an exact search.
- Approximate (index-based) searches can be done if only one column is referenced by either *expr1* or *expr2*, and this column has a vector index defined, and the metric that is specified in the vector_distance matches the metric used in the definition of the vector index.

Parameters

- *expr1* and *expr2* must evaluate to vectors and have the same dimension format, storage format, and number of dimensions.

  If you use JACCARD_DISTANCE or the JACCARD metric, then *expr1* and *expr2* must evaluate to BINARY vectors.
- This function returns NULL if either *expr1* or *expr2* is NULL.
- *metric* must be one of the following tokens :
  - COSINE metric is the default metric. It calculates the cosine distance between two vectors.
  - DOT metric calculates the negated dot product of two vectors. The INNER_PRODUCT function calculates the dot product, as in the negation of this metric.
  - EUCLIDEAN metric, also known as L2 distance, calculates the Euclidean distance between two vectors.
  - EUCLIDEAN_SQUARED metric, also called L2_SQUARED, is the Euclidean distance without taking the square root.
  - HAMMING metric calculates the hamming distance between two vectors by counting the number dimensions that differ between the two vectors.
  - MANHATTAN metric, also known as L1 distance or taxicab distance, calculates the Manhattan distance between two vectors.
  - JACCARD metric calculates the Jaccard distance. The two vectors used in the query must be BINARY vectors.

Shorthand Operators for Distances

Syntax

- expr1 <-> expr2

  <-> is the **Euclidean distance operator**: expr1 <-> expr2 is equivalent to L2_DISTANCE(expr1, expr2) or VECTOR_DISTANCE(expr1, expr2, EUCLIDEAN)
- expr1 <=> expr2

`<=>` is the **cosine distance operator**: `expr1 <=> expr2` is equivalent to `COSINE_DISTANCE(expr1, expr2)` or `VECTOR_DISTANCE(expr1, expr2, COSINE)`

- `expr1 <#> expr2`

  `<#>` is the **negative dot product operator**: `expr1 <#> expr2` is equivalent to `-1*INNER_PRODUCT(expr1, expr2)` or `VECTOR_DISTANCE(expr1, expr2, DOT)`

Examples Using Shorthand Operators for Distances

```
 '[1, 2]' <-> '[0,1]'

v1 <-> '[' || '1,2,3' || ']' is equivalent to v1 <-> '[1, 2, 3]'

v1 <-> '[1,2]' is equivalent to L2_DISTANCE(v1, '[1,2]')

v1 <=> v2 is equivalent to COSINE_DISTANCE(v1, v2)

 v1 <#> v2 is equivalent to -1*INNER_PRODUCT(v1, v2)
```

Examples

`VECTOR_DISTANCE` with metric `EUCLIDEAN` is equivalent to `L2_DISTANCE`:

```
VECTOR_DISTANCE(expr1, expr2, EUCLIDEAN);


L2_DISTANCE(expr1, expr2);
```

`VECTOR_DISTANCE` with metric `COSINE` is equivalent to `COSINE_DISTANCE`:

```
VECTOR_DISTANCE(expr1, expr2, COSINE);


COSINE_DISTANCE(expr1, expr2);
```

`VECTOR_DISTANCE` with metric `DOT` is equivalent to -1 * `INNER_PRODUCT`:

```
VECTOR_DISTANCE(expr1, expr2, DOT);


-1*INNER_PRODUCT(expr1, expr2);
```

`VECTOR_DISTANCE` with metric `MANHATTAN` is equivalent to `L1_DISTANCE`:

```
VECTOR_DISTANCE(expr1, expr2, MANHATTAN);


L1_DISTANCE(expr1, expr2);
```

`VECTOR_DISTANCE` with metric `HAMMING` is equivalent to `HAMMING_DISTANCE`:

```
VECTOR_DISTANCE(expr1, expr2, HAMMING);

HAMMING_DISTANCE(expr1, expr2);
```

`VECTOR_DISTANCE` with metric `JACCARD` is equivalent to `JACCARD_DISTANCE`:

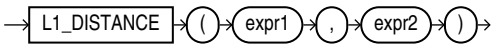
```
VECTOR_DISTANCE(expr1, expr2, JACCARD);
```

```
JACCARD_DISTANCE(expr1, expr2);
```

L1_DISTANCE

L1_DISTANCE is a shorthand version of the VECTOR_DISTANCE function that calculates the Manhattan distance between two vectors. It takes two vectors as input and returns the distance between them as a BINARY_DOUBLE.

Syntax



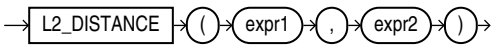
Parameters

- *expr1* and *expr2* must evaluate to vectors and have the same format and number of dimensions.
- L1_DISTANCE returns NULL, if either *expr1* or *expr2* is NULL.

L2_DISTANCE

L2_DISTANCE is a shorthand version of the VECTOR_DISTANCE function that calculates the Euclidean distance between two vectors. It takes two vectors as input and returns the distance between them as a BINARY_DOUBLE.

Syntax



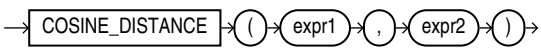
Parameters

- *expr1* and *expr2* must evaluate to vectors that have the same format and number of dimensions.
- L2_DISTANCE returns NULL, if either *expr1* or *expr2* is NULL.

COSINE_DISTANCE

COSINE_DISTANCE is a shorthand version of the VECTOR_DISTANCE function that calculates the cosine distance between two vectors. It takes two vectors as input and returns the distance between them as a BINARY_DOUBLE.

Syntax



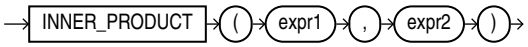
Parameters

- *expr1* and *expr2* must evaluate to vectors that have the same format and number of dimensions.
- COSINE_DISTANCE returns NULL, if either *expr1* or *expr2* is NULL.

INNER_PRODUCT

INNER_PRODUCT calculates the inner product of two vectors. It takes two vectors as input and returns the inner product as a BINARY_DOUBLE. INNER_PRODUCT(<*expr1*>, <*expr2*>) is equivalent to -1 * VECTOR_DISTANCE(<*expr1*>, <*expr2*>, DOT).

Syntax



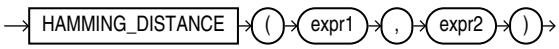
Parameters

- *expr1* and *expr2* must evaluate to vectors that have the same format and number of dimensions.
- INNER_PRODUCT returns NULL, if either *expr1* or *expr2* is NULL.

HAMMING_DISTANCE

HAMMING_DISTANCE is a shorthand version of the VECTOR_DISTANCE function that calculates the hamming distance between two vectors. It takes two vectors as input and returns the distance between them as a BINARY_DOUBLE.

Syntax



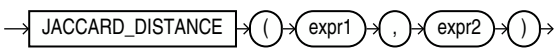
Parameters

- *expr1* and *expr2* must evaluate to vectors that have the same format and number of dimensions.
- HAMMING_DISTANCE returns NULL if either *expr1* or *expr2* is NULL.

JACCARD_DISTANCE

JACCARD_DISTANCE is a shorthand version of the VECTOR_DISTANCE function that calculates the jaccard distance between two vectors. It takes two BINARY vectors as input and returns the distance between them as a BINARY_DOUBLE.

Syntax



Parameters

- *expr1* and *expr2* must evaluate to BINARY vectors and have the same number of dimensions. If either expression is not a BINARY vector, an error is raised.
- JACCARD_DISTANCE returns NULL if either *expr1* or *expr2* is NULL.

Chunking and Vector Generation Functions

Oracle AI Vector Search offers Vector Utilities, which provide the VECTOR_CHUNKS and VECTOR_EMBEDDING SQL functions for chunking data and generating vector embedding, respectively.

- VECTOR_CHUNKS
  Use VECTOR_CHUNKS to split plain text into smaller chunks to generate vector embeddings that can be used with vector indexes or hybrid vector indexes.
- VECTOR_EMBEDDING
  Use VECTOR_EMBEDDING to generate a single vector embedding for different data types using embedding or feature extraction machine learning models.

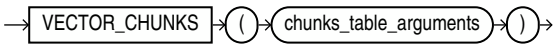
Related Topics

- Vector Generation Examples
  Run these end-to-end examples to see how you can generate vector embeddings, both within and outside the database.

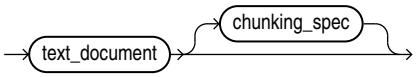
VECTOR_CHUNKS

Use VECTOR_CHUNKS to split plain text into smaller chunks to generate vector embeddings that can be used with vector indexes or hybrid vector indexes.

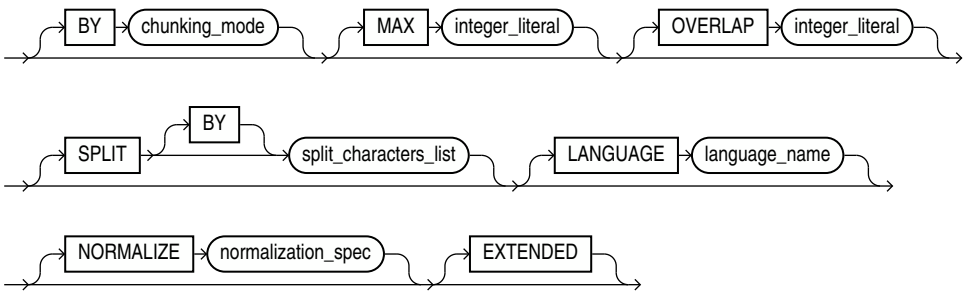
Syntax



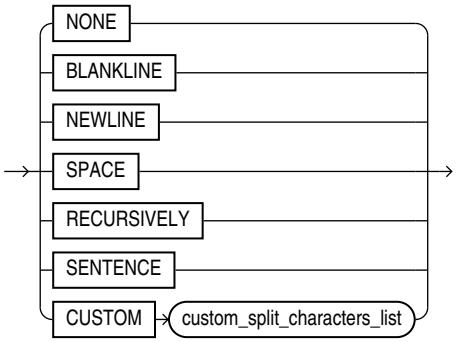
***chunks_table_arguments*::=**



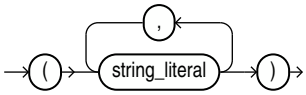
***chunking_spec*::=**



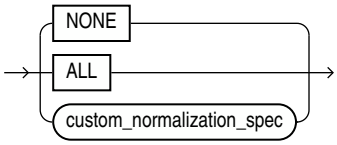
***split_characters_list*::=**



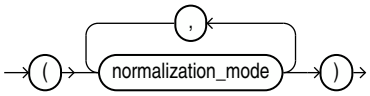
custom_split_characters_list



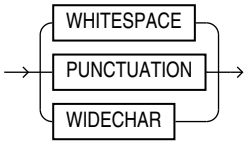
normalization_spec



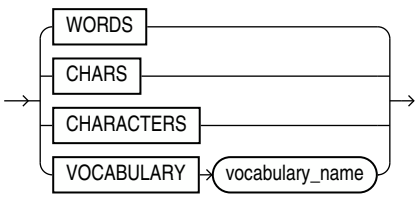
custom_normalization_spec



normalization_mode



***chunking_mode*::=**



Purpose

VECTOR_CHUNKS takes a character value as the *text_document* argument and splits it into chunks using a process controlled by the chunking parameters given in the optional *chunking_spec*. The chunks are returned as rows of a virtual relational table. Therefore, VECTOR_CHUNKS can only appear in the FROM clause of a subquery.

The returned virtual table has the following columns:

- CHUNK_OFFSET of data type NUMBER is the position of each chunk in the source document, relative to the start of the document, which has a position of 1.
- CHUNK_LENGTH of data type NUMBER is the length of each chunk.
- CHUNK_TEXT is a segment of text that has been split off from *text_document*.

The data type of the CHUNK_TEXT column and the length unit used by the values of CHUNK_OFFSET and CHUNK_LENGTH depend on the data type of *text_document* as listed in the following table:

Table 7-1 Input and Output Data Type Details

Input Data Type	Output Data Type	Offset and Length Unit
VARCHAR2	VARCHAR2	byte
CHAR	VARCHAR2	byte
CLOB	VARCHAR2	character
NVARCHAR2	NVARCHAR2	byte
NCHAR	NVARCHAR2	byte
NCLOB	NVARCHAR2	character

Note:

- For more information about data types, see *Data Types* in the SQL Reference Manual.
- The VARCHAR2 input data type is limited to 4000 bytes unless the MAX_STRING_SIZE parameter is set to EXTENDED, which increases the limit to 32767.

Parameters

All chunking parameters are optional, and the default chunking specifications are automatically applied to your chunk data.

When specifying chunking parameters for this API, ensure that you provide these parameters only in the listed order.

Table 7-2 Chunking Parameters Table

Parameter	Description and Acceptable Values
BY	Specifies the mode for splitting your data, that is, to split by counting the number of characters, words, or vocabulary tokens. **Valid values**: • CHARACTERS (or CHARS): Splits by counting the number of characters. • WORDS: Splits by counting the number of words. Words are defined as sequences of alphabetic characters, sequences of digits, individual punctuation marks, or symbols. For segmented languages without whitespace word boundaries (such as Chinese, Japanese, or Thai), each native character is considered a word (that is, unigram). • VOCABULARY: Splits by counting the number of vocabulary tokens. Vocabulary tokens are words or word pieces, recognized by the vocabulary of the tokenizer that your embedding model uses. You can load your vocabulary file using the VECTOR_CHUNKS helper API DBMS_VECTOR_CHAIN.CREATE_VOCABULARY. **Note**: For accurate results, ensure that the chosen model matches the vocabulary file used for chunking. If you are not using a vocabulary file, then ensure that the input length is defined within the token limits of your model. **Default value**: WORDS
MAX	Specifies a limit on the maximum size of each chunk. This setting splits the input text at a fixed point where the maximum limit occurs in the larger text. The units of MAX correspond to the BY mode, that is, to split data when it reaches the maximum size limit of a certain number of characters, words, numbers, punctuation marks, or vocabulary tokens. **Valid values**: • BY CHARACTERS: 50 to 4000 characters • BY WORDS: 10 to 1000 words • BY VOCABULARY: 10 to 1000 tokens **Default value**: 100

Table 7-2 (Cont.) Chunking Parameters Table

Parameter	Description and Acceptable Values
SPLIT [BY]	Specifies where to split the input text when it reaches the maximum size limit. This helps to keep related data together by defining appropriate boundaries for chunks. **Valid values**: • NONE: Splits at the MAX limit of characters, words, or vocabulary tokens. • NEWLINE, BLANKLINE, and SPACE: These are single-split character conditions that split at the last split character before the MAX value. Use NEWLINE to split at the end of a line of text. Use BLANKLINE to split at the end of a blank line (sequence of characters, such as two newlines). Use SPACE to split at the end of a blank space. • RECURSIVELY: This is a multiple-split character condition that breaks the input text using an ordered list of characters (or sequences). RECURSIVELY is predefined as BLANKLINE, NEWLINE, SPACE, NONE in this order: 1. If the input text is more than the MAX value, then split by the first split character. 2. If that fails, then split by the second split character. 3. And so on. 4. If no split characters exist, then split by MAX wherever it appears in the text. • SENTENCE: This is an end-of-sentence split condition that breaks the input text at a sentence boundary. This condition automatically determines sentence boundaries by using knowledge of the input language's sentence punctuation and contextual rules. This language-specific condition relies mostly on end-of-sentence (EOS) punctuations and common abbreviations. Contextual rules are based on word information, so this condition is only valid when splitting the text by words or vocabulary (not by characters). **Note**: This condition obeys the BY WORD and MAX settings, and thus may not determine accurate sentence boundaries in some cases. For example, when a sentence is larger than the MAX value, it splits the sentence at MAX. Similarly, it includes multiple sentences in the text only when they fit within the MAX limit. • CUSTOM: Splits based on a custom list of characters strings, for example, markup tags. You can provide custom sequences up to a limit of 16 split character strings, with a maximum length of 10 bytes each. Provide valid text literals as follows: `VECTOR_CHUNKS(c. doc, BY character SPLIT CUSTOM ('<html>' , '</html>')) vc` **Default value**: RECURSIVELY
OVERLAP	Specifies the amount (as a positive integer literal or zero) of the preceding text that the chunk should contain, if any. This helps in logically splitting up related text (such as a sentence) by including some amount of the preceding chunk text. The amount of overlap depends on how the maximum size of the chunk is measured (in characters, words, or vocabulary tokens). The overlap begins at the specified SPLIT condition (for example, at NEWLINE). **Valid value**: 5% to 20% of MAX **Default value**: 0

Table 7-2 (Cont.) Chunking Parameters Table

Parameter	Description and Acceptable Values
LANGUAGE	Specifies the language of your input data. This clause is important, especially when your text contains certain characters (for example, punctuations or abbreviations) that may be interpreted differently in another language. **Valid values**: • NLS-supported language name or its abbreviation, as listed in *Oracle Database Globalization Support Guide*. • Custom language name or its abbreviation, as listed in Supported Languages and Data File Locations. You use the DBMS_VECTOR_CHAIN.CREATE_LANG_DATA chunker helper API to load language-specific data (abbreviation tokens) into the database, for your specified language. You must use double quotation marks (") for any language name with spaces. For example: LANGUAGE "simplified chinese" For one-word language names, quotation marks are not needed. For example: LANGUAGE american **Default value**: NLS_LANGUAGE from session
NORMALIZE	Automatically pre-processes or post-processes issues (such as multiple consecutive spaces and smart quotes) that may arise when documents are converted into text. Oracle recommends you to use a normalization mode to extract high-quality chunks. **Valid values**: • NONE: Applies no normalization. • ALL: Normalizes multi-byte (Unicode) punctuation to standard single-byte. • Applies all supported normalization modes: PUNCTUATION, WHITESPACE, and WIDECHAR. – PUNCTUATION: Converts quotes, dashes, and other punctuation characters supported in the character set of the text to their common ASCII form. For example: * U+2013 (En Dash) maps to U+002D (Hyphen-Minus) * U+2018 (Left Single Quotation Mark) maps to U+0027 (Apostrophe) * U+2019 (Right Single Quotation Mark) maps to U+0027 (Apostrophe) * U+201B (Single High-Reversed-9 Quotation Mark) maps to U+0027 (Apostrophe) – WHITESPACE: Minimizes whitespace by eliminating unnecessary characters. For example, retain blanklines, but remove any extra newlines and interspersed spaces or tabs: " \n \n " => "\n\n" – WIDECHAR: Normalizes wide, multi-byte digits and (a-z) letters to single-byte. These are multi-byte equivalents for 0-9 and a-z A-Z, which can show up in Chinese, Japanese, or Korean text. **Default value**: NONE
EXTENDED	Increases the output limit of a VARCHAR2 string to 32767 bytes, without requiring you to set the MAX_STRING_SIZE parameter to EXTENDED. If EXTENDED is present in *chunking_spec*, the maximum length of a CHUNK_TEXT column value is 32767 bytes. If it is absent, the maximum length is 4000 bytes if MAX_STRING_SIZE is set to STANDARD and 32767 bytes if MAX_STRING_SIZE is set to EXTENDED.

Examples

VECTOR_CHUNKS can be called for a single character value provided in a character literal or a bind variable as shown in the following example:

```
COLUMN chunk_offset HEADING Offset FORMAT 999
COLUMN chunk_length HEADING Len    FORMAT 999
COLUMN chunk_text   HEADING Text   FORMAT a60

VARIABLE txt VARCHAR2(4000)
EXECUTE :txt := 'An example text value to split with VECTOR_CHUNKS, having over 10 words
because the minimum MAX value is 10';

SELECT * FROM VECTOR_CHUNKS(:txt BY WORDS MAX 10);

SELECT * FROM VECTOR_CHUNKS('Another example text value to split with VECTOR_CHUNKS,
having over 10 words because the minimum MAX value is 10' BY WORDS MAX 10);
```

To chunk values of a table column, the table needs to be joined with the VECTOR_CHUNKS call using left correlation as shown in the following example:

```
CREATE TABLE documentation_tab (
  id   NUMBER,
  text VARCHAR2(2000));

INSERT INTO documentation_tab
   VALUES(1, 'sample');

COMMIT;

SET LINESIZE 100;
SET PAGESIZE 20;
COLUMN pos FORMAT 999;
COLUMN siz FORMAT 999;
COLUMN txt FORMAT a60;

PROMPT SQL VECTOR_CHUNKS
SELECT D.id id, C.chunk_offset pos, C.chunk_length siz, C.chunk_text txt
FROM documentation_tab D, VECTOR_CHUNKS(D.text
                                BY words
                                MAX 200
                                OVERLAP 10
                                SPLIT BY recursively
                                LANGUAGE american
                                NORMALIZE all) C;
```

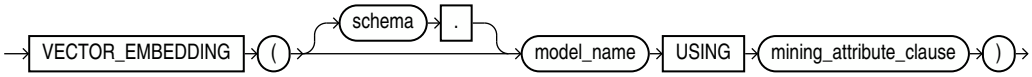
See Also:

- For a complete set of examples on each of the chunking parameters listed in the preceding table, see *Explore Chunking Techniques and Examples of the AI Vector Search User's Guide.*
- To run an end-to-end example scenario using this function, see *Convert Text to Chunks With Custom Chunking Specifications of the AI Vector Search User's Guide.*

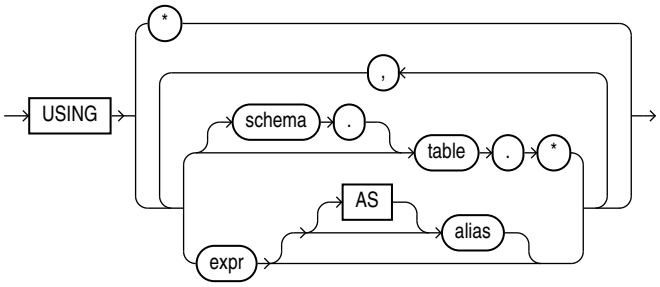
VECTOR_EMBEDDING

Use `VECTOR_EMBEDDING` to generate a single vector embedding for different data types using embedding or feature extraction machine learning models.

Syntax



***mining_attribute_clause*::=**



Purpose

The function accepts the following types as input:

`VARCHAR2` for text embedding models. Oracle automatically converts any other type to `VARCHAR2` except for `NCLOB`, which is automatically converted to `NVARCHAR2`. Oracle does not expect values whose textual representation exceeds the maximum size of a `VARCHAR2`, since embedding models support only text that translates to a couple of thousand tokens. An attribute with a type that has no conversion to `VARCHAR2` results in a SQL compilation error.

For feature extraction models Oracle Machine Learning for SQL supports standard Oracle data types except `DATE`, `TIMESTAMP`, `RAW`, and `LONG`. Oracle Machine Learning supports date type (datetime, date, timestamp) for case_id, `CLOB`/`BLOB`/`FILE` that are interpreted as text columns, and the following collection types as well:

- `DM_NESTED_CATEGORICALS`
- `DM_NESTED_NUMERICALS`
- `DM_NESTED_BINARY_DOUBLES`
- `DM_NESTED_BINARY_FLOATS`

The function always returns a `VECTOR` type, whose dimension is dictated by the model itself. The model stores the dimension information in metadata within the data dictionary.

You can use `VECTOR_EMBEDDING` in `SELECT` clauses, in predicates, and as an operand for SQL operations accepting a `VECTOR` type.

Parameters

model_name refers to the name of the imported embedding model that implements the embedding machine learning function.

mining_attribute_clause

- The *mining_attribute_clause* argument identifies the column attributes to use as predictors for scoring. This is used as a convenience, as the embedding operator only accepts single input value.
- USING * : all the relevant attributes present in the input (supplied in JSON metadata) are used. This is used as a convenience. For an embedding model, the operator only takes one input value as embedding models have only one column.
- USING *expr* [AS *alias*] [, *expr* [AS *alias*]] : all the relevant attributes present in the comma-separated list of column expressions are used. This syntax is consistent with the syntax of other machine learning operators. You may specify more than one attribute, however, the embedding model only takes one relevant input. Therefore, you must specify a single mining attribute.

Example

The following example generates vector embeddings with "hello" as the input, utilizing the pretrained ONNX format model my_embedding_model.onnx imported into the Database. For complete example, see Import ONNX Models and Generate Embeddings

```
SELECT TO_VECTOR(VECTOR_EMBEDDING(model USING 'hello' as data)) AS embedding;
--------------------------------------------------------------------------------
[-9.76553112E-002,-9.89954844E-002,7.69771636E-003,-4.16760892E-003,-9.69305634E-002,
-3.01141385E-002,-2.63396613E-002,-2.98553891E-002,5.96499592E-002,4.13885899E-002,
5.32859489E-002,6.57707453E-002,-1.47056757E-002,-4.18472625E-002,4.1588001E-002,
-2.86354572E-002,-7.56499246E-002,-4.16395674E-003,-1.52879998E-001,6.60010576E-002,
-3.9013084E-002,3.15719917E-002,1.2428958E-002,-2.47651711E-002,-1.16851285E-001,
-7.82847106E-002,3.34323719E-002,8.03267583E-002,1.70483496E-002,-5.42407483E-002,
6.54291287E-002,-4.81935125E-003,6.11041225E-002,6.64106477E-003,-5.47
```

> **See Also:**
>
> - *Data Requirements for Machine Learning*
> - *Vector Distance Metrics*

Constructors, Converters, Descriptors, and Arithmetic Operators

Other basic vector operations for Oracle AI Vector Search involve creating, converting, and describing vectors.

- Vector Constructors
  TO_VECTOR() and VECTOR() are synonymous constructors of vectors. The functions take a string of type VARCHAR2 or CLOB as input and return a vector as output.

- Vector Serializers
  `FROM_VECTOR()` and `VECTOR_SERIALIZE()` are synonymous serializers of vectors. The functions take a vector as input and return a string of type `VARCHAR2` or `CLOB` as output.
- VECTOR_NORM
  `VECTOR_NORM` returns the Euclidean norm of a vector as a `BINARY_DOUBLE`. This value is also called the magnitude or size and represents the Euclidean distance between the vector and the origin. For a vector `v = (v1, v2, ..., vn)`, the Euclidean norm is given by `||v|| = SQRT(v1^2 + v2^2 + ... + vn^2)`.
- VECTOR_DIMENSION_COUNT
  `VECTOR_DIMENSION_COUNT` returns the number of dimensions of a vector as a `NUMBER`.
- VECTOR_DIMS
  `VECTOR_DIMS` returns the number of dimensions of a vector as a `NUMBER`. `VECTOR_DIMS` is synonymous with `VECTOR_DIMENSION_COUNT`.
- VECTOR_DIMENSION_FORMAT
  `VECTOR_DIMENSION_FORMAT` returns the storage format of the vector. It returns a `VARCHAR2`, which can be one of the following values: `INT8`, `FLOAT32`, `FLOAT64`, or `BINARY`.
- Arithmetic Operators
  Addition, subtraction, and multiplication can be applied to vectors dimension-wise in SQL and PL/SQL.
- Aggregate Functions
  The aggregate functions `SUM` and `AVG` can be applied to a series of vectors dimension-wise.

Vector Constructors

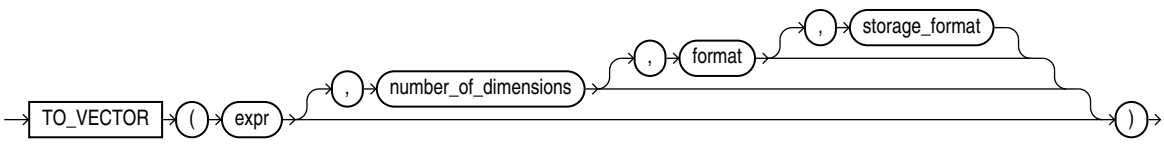
`TO_VECTOR()` and `VECTOR()` are synonymous constructors of vectors. The functions take a string of type `VARCHAR2` or `CLOB` as input and return a vector as output.

- TO_VECTOR
  `TO_VECTOR` is a constructor that takes a string of type `VARCHAR2`, `CLOB`, `BLOB`, or `JSON` as input, converts it to a vector, and returns a vector as output. `TO_VECTOR` also takes another vector as input, adjusts its format, and returns the adjusted vector as output. `TO_VECTOR` is synonymous with `VECTOR`.
- VECTOR
  `VECTOR` is synonymous with `TO_VECTOR`.

TO_VECTOR

`TO_VECTOR` is a constructor that takes a string of type `VARCHAR2`, `CLOB`, `BLOB`, or `JSON` as input, converts it to a vector, and returns a vector as output. `TO_VECTOR` also takes another vector as input, adjusts its format, and returns the adjusted vector as output. `TO_VECTOR` is synonymous with `VECTOR`.

Syntax



Parameters

- *expr* must evaluate to one of:
  - A string (of character types or CLOB) that represents a vector.
  - A VECTOR.
  - A BLOB. The BLOB must represent the vector's binary bytes.
  - A JSON array. All elements in the array must be numeric.

  If *expr* is NULL, the result is NULL.

  The string representation of the vector must be in the form of an array of non-null numbers enclosed with a bracket and separated by commas, such as [1, 3.4, -05.60, 3e+4]. TO_VECTOR converts a valid string representation of a vector to a vector in the format specified. If no format is specified the default format is used.
- *number_of_dimensions* must be a numeric value that describes the number of dimensions of the vector to construct. The number of dimensions may also be specified as an asterisk (*), in which case the dimension is determined by *expr*.
- *format* must be one of the following tokens: INT8, FLOAT32, FLOAT64, BINARY, or *. This is the target internal storage format of the vector. If * is used, the format will be FLOAT32.

  Note that this behavior is different from declaring a vector column. When you declare a column of type VECTOR(3, *), then all inserted vectors will be stored as is without a change in format.
- *storage_format* must be one of the following tokens: DENSE, SPARSE, or *. If no storage format is specified or if * is used, the following will be observed depending on the input type:
  - Textual input: the storage format will default to DENSE.
  - JSON input: the storage format will default to DENSE.
  - VECTOR input: there is no default and the storage format is not changed.
  - BLOB input: there is no default and the storage format is not changed.

Examples

```
SELECT TO_VECTOR('[34.6, 77.8]');

TO_VECTOR('[34.6,77.8]')
------------------------------------------------------------
[3.45999985E+001,7.78000031E+001]



SELECT TO_VECTOR('[34.6, 77.8]', 2, FLOAT32);

TO_VECTOR('[34.6,77.8]',2,FLOAT32)
------------------------------------------------------------
[3.45999985E+001,7.78000031E+001]



SELECT TO_VECTOR('[34.6, 77.8, -89.34]', 3, FLOAT32);

TO_VECTOR('[34.6,77.8,-89.34]',3,FLOAT32)
```

```
------------------------------------------------------------
[3.45999985E+001,7.78000031E+001,-8.93399963E+001]


SELECT TO_VECTOR('[34.6, 77.8, -89.34]', 3, FLOAT32, DENSE);

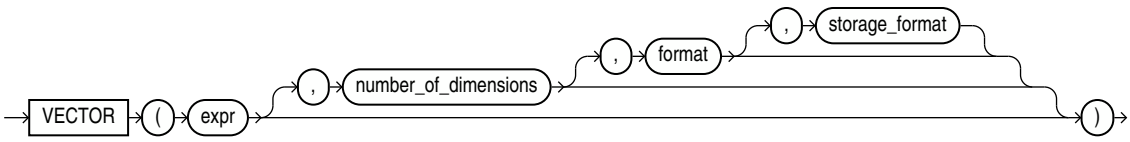
TO_VECTOR('[34.6,77.8,-89.34]',3,FLOAT32,DENSE)
--------------------------------------------------------------------------------
[3.45999985E+001,7.78000031E+001,-8.93399963E+001]
```

> **Note:**
>
> - For applications using Oracle Client libraries prior to 23ai connected to Oracle Database 23ai, use the `TO_VECTOR` function to insert vector data. For example:
>
>   ```
>   INSERT INTO vecTab VALUES(TO_VECTOR('[1.1, 2.9, 3.14]'));
>   ```
>
> - Applications using Oracle Client 23ai libraries or Thin mode drivers can insert vector data directly as a string or a `CLOB`. For example:
>
>   ```
>   INSERT INTO vecTab VALUES ('[1.1, 2.9, 3.14]');
>   ```

VECTOR

`VECTOR` is synonymous with `TO_VECTOR`.

Syntax



Purpose

See TO_VECTOR for semantics and examples.

> **Note:**
>
> Applications using Oracle Client 23ai libraries or Thin mode drivers can insert vector data directly as a string or a `CLOB`. For example:
>
> ```
> INSERT INTO vecTab VALUES ('[1.1, 2.9, 3.14]');
> ```

Examples

```
SELECT VECTOR('[34.6, 77.8]');
```

```
VECTOR('[34.6,77.8]')
-------------------------------------------------------
[3.45999985E+001,7.78000031E+001]


SELECT VECTOR('[34.6, 77.8]', 2, FLOAT32);

VECTOR('[34.6,77.8]',2,FLOAT32)
-------------------------------------------------------
[3.45999985E+001,7.78000031E+001]


SELECT VECTOR('[34.6, 77.8, -89.34]', 3, FLOAT32);

VECTOR('[34.6,77.8,-89.34]',3,FLOAT32)
-----------------------------------------------------------
[3.45999985E+001,7.78000031E+001,-8.93399963E+001]
```

Vector Serializers

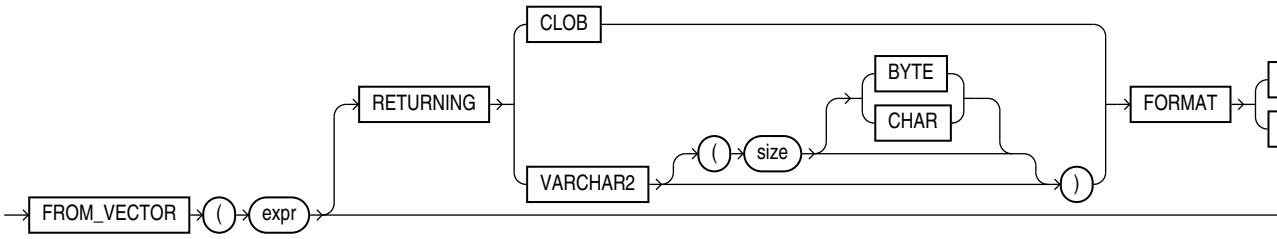
`FROM_VECTOR()` and `VECTOR_SERIALIZE()` are synonymous serializers of vectors. The functions take a vector as input and return a string of type `VARCHAR2` or `CLOB` as output.

- FROM_VECTOR
  `FROM_VECTOR` takes a vector as input and returns a string of type `VARCHAR2` or `CLOB` as output.
- VECTOR_SERIALIZE
  `VECTOR_SERIALIZE` is synonymous with `FROM_VECTOR`.

FROM_VECTOR

`FROM_VECTOR` takes a vector as input and returns a string of type `VARCHAR2` or `CLOB` as output.

Syntax



Purpose

`FROM_VECTOR` optionally takes a `RETURNING` clause to specify the data type of the returned value.

If `VARCHAR2` is specified without size, the size of the returned value size is 32767.

In SQL, you can optionally specify the storage format of the input vector in the `FORMAT` clause, using the tokens `SPARSE` or `DENSE`. The return storage format cannot be specified in PL/SQL; rather, the output format is determined by the storage format of the input vector.

There is no support to convert to `CHAR`, `NCHAR`, and `NVARCHAR2`.

`FROM_VECTOR` is synonymous with `VECTOR_SERIALIZE`.

Parameters

`expr` must evaluate to a vector. The function returns NULL if *`expr`* is NULL.

Examples

- ```
  SELECT FROM_VECTOR(TO_VECTOR('[1, 2, 3]') );
  ```

  Result:

  ```
  FROM_VECTOR(TO_VECTOR('[1,2,3]'))

  ------------------------------------------------------------
  [1.0E+000,2.0E+000,3.0E+000]

  1 row selected.
  ```

- ```
  SELECT FROM_VECTOR(TO_VECTOR('[1.1, 2.2, 3.3]', 3, FLOAT32) );
  ```

  Result:

  ```
  FROM_VECTOR(TO_VECTOR('[1.1,2.2,3.3]',3,FLOAT32))
  ------------------------------------------------------------------

  [1.10000002E+000,2.20000005E+000,3.29999995E+000]

  1 row selected.
  ```

- ```
  SELECT FROM_VECTOR( TO_VECTOR('[1.1, 2.2, 3.3]', 3, FLOAT32) RETURNING
  VARCHAR2(1000));
  ```

  Result:

  ```
  FROM_VECTOR(TO_VECTOR('[1.1,2.2,3.3]',3,FLOAT32)RETURNINGVARCHAR2(1000))
  ------------------------------------------------------------------------
  -----

  [1.10000002E+000,2.20000005E+000,3.29999995E+000]

  1 row selected.
  ```

- ```
  SELECT FROM_VECTOR(TO_VECTOR('[1.1, 2.2, 3.3]', 3, FLOAT32) RETURNING
  CLOB );
  ```

  Result:

  ```
  FROM_VECTOR(TO_VECTOR('[1.1,2.2,3.3]',3,FLOAT32)RETURNINGCLOB)
  ```

```
--------------------------------------------------------------------------------
-----

[1.10000002E+000,2.20000005E+000,3.29999995E+000]

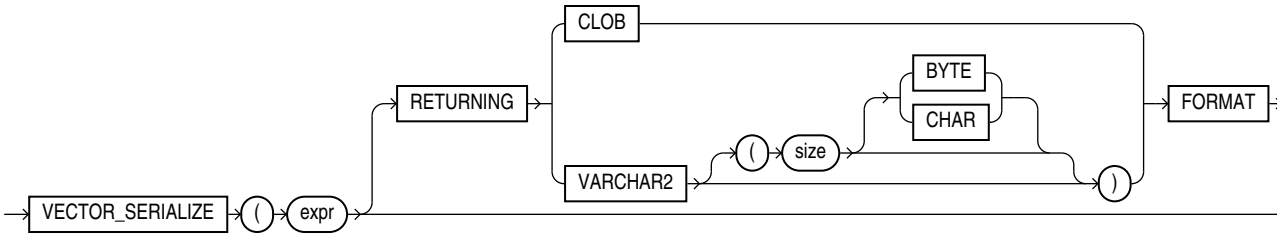
1 row selected.
```

> **Note:**
>
> - Applications using Oracle Client 23ai libraries or Thin mode drivers can fetch vector data directly, as shown in the following example:
>
>   ```
>   SELECT dataVec FROM vecTab;
>   ```
>
> - For applications using Oracle Client 23ai libraries prior to 23ai connected to Oracle Database 23ai, use the `FROM_VECTOR` to fetch vector data, as shown by the following example:
>
>   ```
>   SELECT FROM_VECTOR(dataVec) FROM vecTab;
>   ```

VECTOR_SERIALIZE

`VECTOR_SERIALIZE` is synonymous with `FROM_VECTOR`.

Syntax



Purpose

See FROM_VECTOR for semantics and examples.

Examples

- ```
  SELECT VECTOR_SERIALIZE(VECTOR('[1.1,2.2,3.3]',3,FLOAT32));
  ```

  Result:

  ```
  VECTOR_SERIALIZE(VECTOR('[1.1,2.2,3.3]',3,FLOAT32))
  ----------------------------------------------------------------
  [1.10000002E+000,2.20000005E+000,3.29999995E+000]

  1 row selected.
  ```
  ```

- ```
  SELECT VECTOR_SERIALIZE(VECTOR('[1.1, 2.2, 3.3]',3,FLOAT32) RETURNING
  VARCHAR2(1000));
  ```

  Result:

  ```
  VECTOR_SERIALIZE(VECTOR('[...]',3,FLOAT32)RETURNINGVARCHAR2(1000))
  ------------------------------------------------------------------
  [1.10000002E+000,2.20000005E+000,3.29999995E+000]

  1 row selected.
  ```

- ```
  SELECT VECTOR_SERIALIZE(VECTOR('[1.1, 2.2, 3.3]',3,FLOAT32) RETURNING
  CLOB);
  ```

  Result:

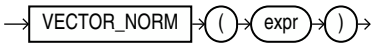
  ```
  VECTOR_SERIALIZE(VECTOR('[1.1, 2.2, 3.3]',3,FLOAT32)RETURNINGCLOB)
  ------------------------------------------------------------
  [1.10000002E+000,2.20000005E+000,3.29999995E+000]

  1 row selected.
  ```

VECTOR_NORM

`VECTOR_NORM` returns the Euclidean norm of a vector as a `BINARY_DOUBLE`. This value is also called the magnitude or size and represents the Euclidean distance between the vector and the origin. For a vector `v = (v1, v2, ..., vn)`, the Euclidean norm is given by `||v|| = SQRT(v1^2 + v2^2 + ... + vn^2)`.

Syntax



Parameters

expr must evaluate to a vector.

If *expr* is NULL, NULL is returned.

Example

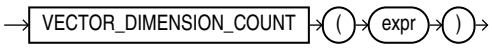
```
SELECT VECTOR_NORM( TO_VECTOR('[4, 3]', 2, FLOAT32) );

VECTOR_NORM(TO_VECTOR('[4,3]',2,FLOAT32))
_________________________________________
5.0
```

VECTOR_DIMENSION_COUNT

`VECTOR_DIMENSION_COUNT` returns the number of dimensions of a vector as a `NUMBER`.

Syntax



Purpose

`VECTOR_DIMENSION_COUNT` is synonymous with VECTOR_DIMS.

Parameters

`expr` must evaluate to a vector.

If *`expr`* is NULL, NULL is returned.

Example

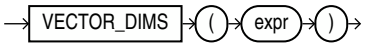
```
SELECT VECTOR_DIMENSION_COUNT( TO_VECTOR('[34.6, 77.8]', 2, FLOAT64) );

VECTOR_DIMENSION_COUNT(TO_VECTOR('[34.6,77.8]',2,FLOAT64))
----------------------------------------------------------
2
```

VECTOR_DIMS

`VECTOR_DIMS` returns the number of dimensions of a vector as a `NUMBER`. `VECTOR_DIMS` is synonymous with `VECTOR_DIMENSION_COUNT`.

Syntax



Purpose

Refer to VECTOR_DIMENSION_COUNT for full semantics.

VECTOR_DIMENSION_FORMAT

`VECTOR_DIMENSION_FORMAT` returns the storage format of the vector. It returns a `VARCHAR2`, which can be one of the following values: `INT8`, `FLOAT32`, `FLOAT64`, or `BINARY`.

Syntax

Parameters

expr must evaluate to a vector.

If *expr* is NULL, NULL is returned.

Examples

```
SELECT VECTOR_DIMENSION_FORMAT(TO_VECTOR('[34.6, 77.8]', 2, FLOAT64));
FLOAT64

SELECT VECTOR_DIMENSION_FORMAT(TO_VECTOR('[34.6, 77.8, 9]', 3, FLOAT32));
FLOAT32

SELECT VECTOR_DIMENSION_FORMAT(TO_VECTOR('[34.6, 77.8, 9, 10]', 3, INT8));
SELECT VECTOR_DIMENSION_FORMAT(TO_VECTOR('[34.6, 77.8, 9, 10]', 3,
INT8))
                                                                        *
ERROR at line 1:
ORA-51803: Vector dimension count must match the dimension count specified in the column
definition (actual: 4, required: 3).

SELECT VECTOR_DIMENSION_FORMAT(TO_VECTOR('[34.6, 77.8, 9.10]', 3, INT8));
VECTOR_DIMENSION_FORMAT(TO_VECTOR('[34.6,77.8,9.10
--------------------------------------------------
INT8

SELECT TO_VECTOR('[34.6, 77.8, 9.10]', 3, INT8);
TO_VECTOR('[34.6,77.8,9.10]',3,INT8)
--------------------------------------------------
[3.5E+001,7.8E+001,9.0E+000]
```

Arithmetic Operators

Addition, subtraction, and multiplication can be applied to vectors dimension-wise in SQL and PL/SQL.

Addition

Vector addition is often used in Natural Language Processing (NLP) where words are often represented as vectors that capture their meaning in a numerical format. Vector addition is used to combine these meanings and understand relationships between words, a task also referred to as word analogy.

Given two vectors `A=(a1, a2, a3)` and `B=(b1, b2, b3)`, `C = A + B` is computed as `C = (a1+b1, a2+b2, a3+b3)`.

Subtraction

Vector subtraction can be used in word analogy scenarios but is also useful in the context of facial recognition. Each face can be represented by a vector of facial features (distance between eyes, eye color, and so on). Subtracting one vector from another gives you the main differences between the two faces, giving you the information needed to recognize whether they are similar or not.

Given two vectors `A=(a1, a2, a3)` and `B=(b1, b2, b3)`, `C = A - B` is computed as `C = (a1-b1, a2-b2, a3-b3)`.

Multiplication

Vector multiplication of each corresponding coordinate of two vectors, or element-wise product, is called the Hadamard product. The Hadamard product is often used in neural networks and computer vision.

Given two vectors `A=(a1, a2, a3)` and `B=(b1, b2, b3)`, the Hadamard product `A*B` is computed as `A*B = (a1*b1, a2*b2, a3*b3)`.

Semantics

Both sides of the operation must evaluate to vectors with matching dimensions and must not be `BINARY` or `SPARSE` vectors. The resulting vector has the same number of dimensions as the operands and the format is determined based on the formats of the inputs. If one side of the operation is not a vector, an attempt is made automatically to convert the value to a vector. If the conversion fails, an error is raised.

The format used for the result is ranked in the following order: flexible, `FLOAT64`, `FLOAT32`, then `INT8`. As in, if either side of the operation has a flexible format, the result will be flexible, otherwise, if either side has the format `FLOAT64`, the result will be `FLOAT64`, and so on.

Consider two vectors with the following values:

```
v1 = [1, 2, 3]
v2 = [10, 20, 30]
```

Using arithmetic operators on `v1` and `v2` would, for example, result in the following:

- `v1 + v2` is `[11, 22, 33]`
- `v1 - v2` is `[-9, -18, -27]`
- `v1 * v2` is `[10, 40, 90]`
- `v1 + NULL` is `NULL`

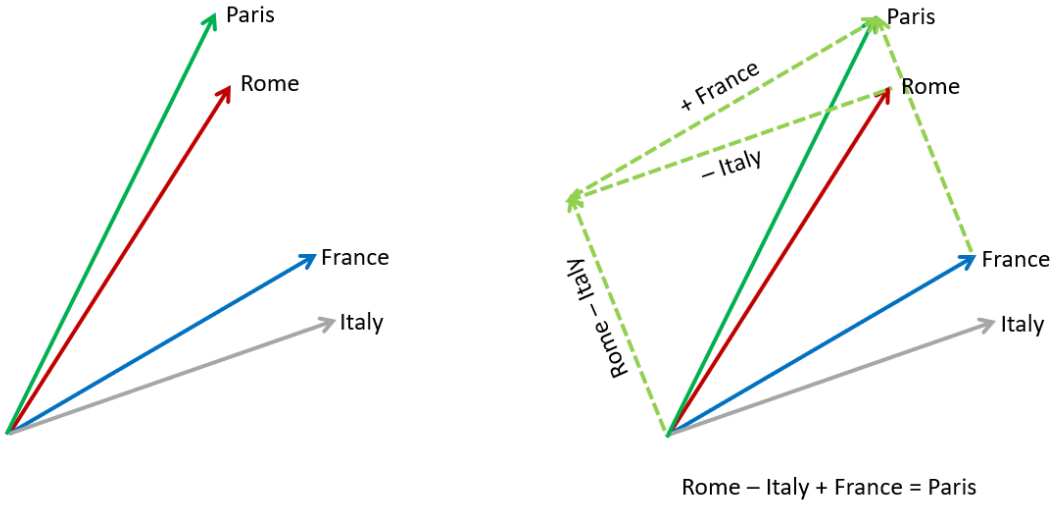
If either side of the arithmetic operation is `NULL`, the result is `NULL`. In the case of dimension overflow, an error is raised. For example, adding `VECTOR('[1, 127]', 2, INT8)` to `VECTOR('[1, 1]', 2, INT8)` results in an error because `127+1=128`, which overflows the `INT8` format.

The use of division operators on vectors is not supported.

Examples

Word Analogy

Using word embeddings, suppose you want to find the relationship between "Rome" and "Paris". You can take the vector for "Rome", subtract the vector for "Italy", and then add the vector for "France". This results in a new vector that approximates the meaning of the word "Paris". The calculation should be "Rome - Italy + France = Paris" using an ideal embedding model.



Basic Vector Arithmetic

```
SELECT VECTOR('[5, 10, 15]') - VECTOR('[2, 4, 6]');

VECTOR('[5,10,15]')-VECTOR('[2,4,6]')
--------------------------------------------------------------
[3.0E+000,6.0E+000,9.0E+000]


SELECT VECTOR('[1, 2, 3]', 3, FLOAT64) + VECTOR('[4, 5, 6]', 3, FLOAT32) *
'[2, 2, 2]';

VECTOR('[1,2,3]',3,FLOAT64)+VECTOR('[4,5,6]',3,FLOAT32)*'[2,2,2]'
--------------------------------------------------------------------------------
--
[9.0E+000,1.2E+001,1.5E+001]


DECLARE
  v1 VECTOR := VECTOR('[10, 20, 30]', 3, INT8);
  v2 VECTOR := VECTOR('[6, 4, 2]', 3, INT8);
BEGIN
  DBMS_OUTPUT.PUT_LINE(TO_CHAR(v1 + v2));
  DBMS_OUTPUT.PUT_LINE(TO_CHAR(v1 - v2));
  DBMS_OUTPUT.PUT_LINE(TO_CHAR(v1 * v2));
END;
/
```

Result:

```
[16,24,32]
[4,16,28]
[60,80,60]
```

Aggregate Functions

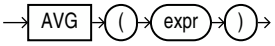
The aggregate functions `SUM` and `AVG` can be applied to a series of vectors dimension-wise.

- AVG
  The `AVG` function takes a vector expression as input and returns the average as a vector with format `FLOAT64`.
- SUM
  The `SUM` function takes a vector expression as input and returns the sum as a vector with format `FLOAT64`.

AVG

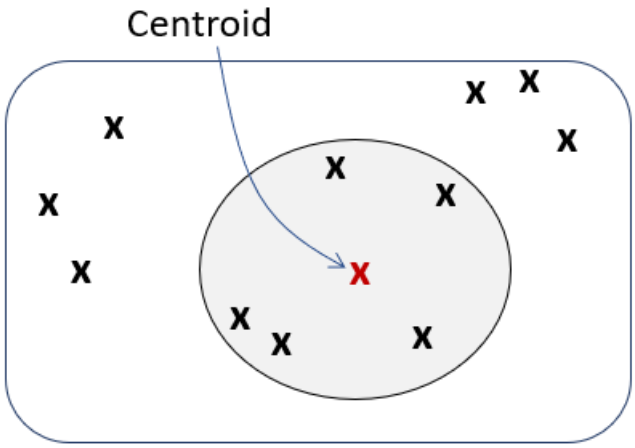
The `AVG` function takes a vector expression as input and returns the average as a vector with format `FLOAT64`.

Syntax



Purpose

`AVG` is mainly used to create an overall representation (as in a centroid) for a vector set. In applications like Natural Language Processing (NLP), you can compute the average of several vectors to create a single centroid or overall representation. For example, to represent a sentence, you might average the word embeddings of each word in the sentence. This can be used for tasks like text classification, document similarity, or clustering.



The result of `AVG` with a vector expression is the equivalent of consecutively performing vector addition operations on all non-`NULL` inputs and then dividing by the total number of non-`NULL` inputs. The returned vector has the same number of dimensions as the input and has the format `FLOAT64`. When the expression has a flexible number of dimensions, all inputs must have the same number of dimensions within each aggregate group.

The AVG function with vector expressions as input can be used as a single set aggregate or in the GROUP BY clause. Using ROLLUP is also supported. The AVG function accepts vector expressions as input for aggregate operations but cannot currently be applied to analytic operations.

NULL vectors are ignored and are not counted when calculating the average vector. If all inputs within an aggregate group are NULL, the result is NULL for that group. If the result overflows the FLOAT64 maximum value, an error is raised, regardless of the format of the input vector type.

With vector inputs, using DISTINCT, CUBE, and GROUPING SETS is not supported. Also, BINARY and SPARSE vectors cannot be supplied as input.

For the full definition and implementation of the AVG function, see *Oracle Database SQL Language Reference*.

```
CREATE TABLE avg_t (v VECTOR, k1 NUMBER, k2 VARCHAR2(100));
INSERT INTO avg_t VALUES ('[2, 4, 6]', 2, 'even');
INSERT INTO avg_t VALUES ('[8, 10, 12]', 2, 'even');
INSERT INTO avg_t VALUES ('[1, 3, 5]', 3, 'odd');
INSERT INTO avg_t VALUES ('[7, 9, 11]', 3, 'odd');


SELECT AVG(v) v_avg FROM avg_t;

V_AVG
--------------------------------------------------
[4.5E+000,6.5E+000,8.5E+000]


SELECT AVG(v) v_avg, k1 FROM avg_t GROUP BY k1;

V_AVG                                 K1
------------------------------------  ----------
[5.0E+000,7.0E+000,9.0E+000]          2
[4.0E+000,6.0E+000,8.0E+000]          3

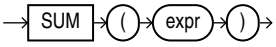

SELECT AVG(v) v_avg FROM avg_t GROUP BY ROLLUP(k1, k2);

V_AVG
--------------------------------------------------
[5.0E+000,7.0E+000,9.0E+000]
[5.0E+000,7.0E+000,9.0E+000]
[4.0E+000,6.0E+000,8.0E+000]
[4.0E+000,6.0E+000,8.0E+000]
[4.5E+000,6.5E+000,8.5E+000]
```

SUM

The SUM function takes a vector expression as input and returns the sum as a vector with format FLOAT64.

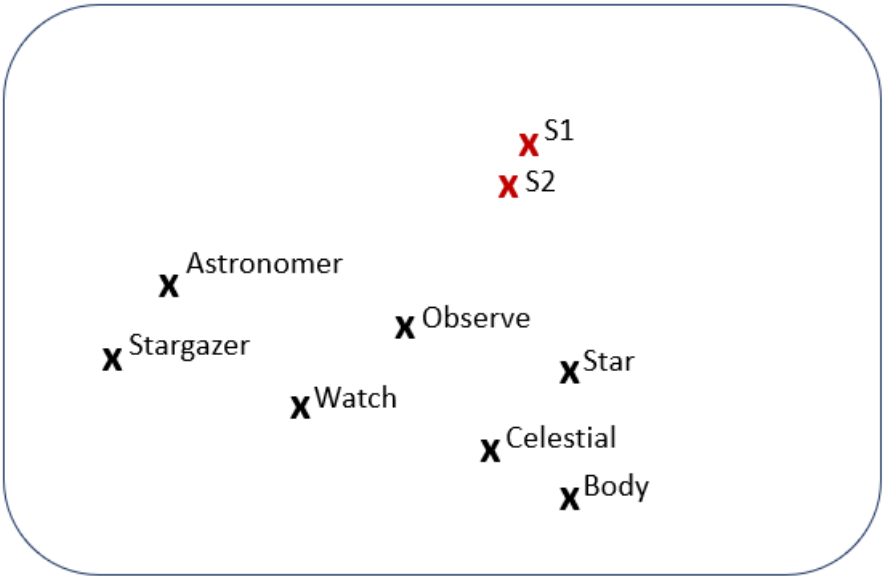
Syntax



Purpose

`SUM` is mainly used, in Natural Language Processing (NLP), to compute a representation of a sentence or a document. It is common to sum the word embeddings of all words. The resulting vector represents the entire text, allowing models to work with sentences instead of just words.

The following diagram illustrates the vector representation, in a 2D space, of word embeddings. If you take S1, "The astronomer observed the stars", and S2, "The stargazer watched the celestial bodies", you can observe that the corresponding vector sum of each constituting word of each sentence (S1 and S2 in the diagram) are very close, for example, in terms of cosine similarity. This means that the two sentences are very similar in meaning.



S1 = Astronomer + Observe + Star
S2 = Stargazer + Watch + Celestial + Body

The result of `SUM` with a vector expression is the equivalent of consecutively performing vector addition operations on all non-`NULL` inputs. The returned vector has the same number of dimensions as the input and has the format `FLOAT64`. When the expression has a flexible number of dimensions, all inputs must have the same number of dimensions within each aggregate group.

The `SUM` function with vector expressions as input can be used as a single set aggregate or in the `GROUP BY` clause. Using `ROLLUP` is also supported. The `SUM` function accepts vector expressions as input for aggregate operations but cannot currently be applied to analytic operations.

`NULL` vectors are ignored and are not counted when calculating the sum vector. If all inputs within an aggregate group are `NULL`, the result is `NULL` for that group. If the result overflows the `FLOAT64` maximum value, an error is raised, regardless of the format of the input vector type.

With vector inputs, using `DISTINCT`, `CUBE`, and `GROUPING SETS` is not supported. Also, `BINARY` and `SPARSE` vectors cannot be supplied as input.

For the full definition and implementation of the `SUM` function, see *Oracle Database SQL Language Reference*.

```
CREATE TABLE sum_t (v VECTOR, k1 NUMBER, k2 VARCHAR2(100));
INSERT INTO sum_t VALUES ('[2, 4, 6]', 2, 'even');
INSERT INTO sum_t VALUES ('[8, 10, 12]', 2, 'even');
INSERT INTO sum_t VALUES ('[1, 3, 5]', 3, 'odd');
INSERT INTO sum_t VALUES ('[7, 9, 11]', 3, 'odd');


SELECT SUM(v) v_sum FROM sum_t;

V_SUM
--------------------------------------------------
[1.8E+001,2.6E+001,3.4E+001]


SELECT SUM(v) v_sum, k1 FROM sum_t GROUP BY k1;

V_SUM                                  K1
-------------------------------------  ----------
[1.0E+001,1.4E+001,1.8E+001]           2
[8.0E+000,1.2E+001,1.6E+001]           3


SELECT SUM(v) v_sum FROM sum_t GROUP BY ROLLUP(k1, k2);

V_SUM
--------------------------------------------------
[1.0E+001,1.4E+001,1.8E+001]
[1.0E+001,1.4E+001,1.8E+001]
[8.0E+000,1.2E+001,1.6E+001]
[8.0E+000,1.2E+001,1.6E+001]
[1.8E+001,2.6E+001,3.4E+001]


CREATE TABLE sum_diff_dim_t (v VECTOR, k1 NUMBER, k2 VARCHAR2(100));
INSERT INTO sum_diff_dim_t VALUES ('[2, 4, 6]', 2, 'even');
INSERT INTO sum_diff_dim_t VALUES ('[8, 10, 12]', 2, 'even');
INSERT INTO sum_diff_dim_t VALUES ('[1, 3, 5, 7]', 3, 'odd');
INSERT INTO sum_diff_dim_t VALUES ('[9, 11, 13, 15]', 3, 'odd');


SELECT SUM(v) v_sum, k2 FROM sum_diff_dim_t GROUP BY k2;

V_SUM                                  K2
-------------------------------------  ----------
[1.0E+001,1.4E+001,1.8E+001]           even
[1.0E+001,1.4E+001,1.8E+001,2.2E+001]  odd


SELECT SUM(v) v_sum FROM sum_diff_dim_t;

ERROR:
```

```
ORA-51808: SUM(vector) requires all vectors to have the same dimension count.
Encountered (3, 4).
```

JSON Compatibility with the VECTOR Data Type

The JSON data type supports the `VECTOR` type as a JSON scalar type. A `VECTOR` instance is convertible to a JSON type instance and vice versa using the JSON constructor and the `VECTOR` constructor, respectively.

Note that when converting a JSON array to a `VECTOR` type instance, the numbers in the JSON array do not have to be of the same numeric type. The input data types do not change the default numeric type for the vector and the `VECTOR` constructor includes an argument to set the format. Using the `VECTOR` constructor (or `TO_VECTOR`) with a JSON data type instance will succeed as long as there is no conversion error.

The following SQL/JSON functions and item methods are compatible for use with the `VECTOR` data type:

- `JSON_VALUE`
- `JSON_OBJECT`
- `JSON_ARRAY`
- `JSON_TABLE`
- `JSON_SCALAR`
- `JSON_TRANSFORM`
- `JSON_SERIALIZE`
- `type()`
- `.vector()`
- `.string()`
- `.stringify()`

See Also:

- *Oracle Database JSON Developer's Guide* for more information about `VECTOR` as a data type for JSON data
- *Oracle Database JSON Developer's Guide* for information about SQL/JSON path expression item methods, including `vector()`

8

Query Data With Similarity and Hybrid Searches

Use Oracle AI Vector Search native SQL operations from your development environment to combine similarity with relational searches.

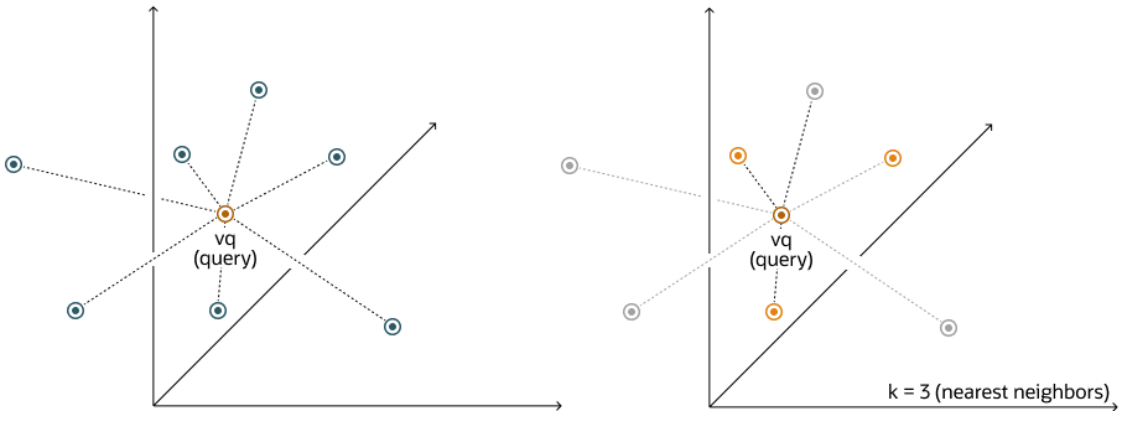
- Perform Exact Similarity Search
  A similarity search looks for the relative order of vectors compared to a query vector. Naturally, the comparison is done using a particular distance metric but what is important is the result set of your top closest vectors, not the distance between them.
- Perform Approximate Similarity Search Using Vector Indexes
  For a vector search to be useful, it needs to be fast and accurate. Approximate similarity searches seek a balance between these goals.
- Perform Multi-Vector Similarity Search
  Another major use-case of vector search is multi-vector search. Multi-vector search is typically associated with a multi-document search, where documents are split into chunks that are individually embedded into vectors.
- Perform Hybrid Search
  Hybrid search is an advanced information retrieval technique that lets you search documents by *keywords* and *vectors*, to achieve more relevant search results.

Perform Exact Similarity Search

A similarity search looks for the relative order of vectors compared to a query vector. Naturally, the comparison is done using a particular distance metric but what is important is the result set of your top closest vectors, not the distance between them.

As an example, and given a certain query vector, you can calculate its distance to all other vectors in your data set. This type of search, also called flat search, or exact search, produces the most accurate results with perfect search quality. However, this comes at the cost of significant search times. This is illustrated by the following diagrams:

Figure 8-1 Exact Search



With an exact search, you compare the query vector `vq` against every other vector in your space by calculating its distance to each vector. After calculating all of these distances, the search returns the nearest `k` of those as the nearest matches. This is called a k-nearest neighbors (kNN) search.

For example, the Euclidean similarity search involves retrieving the top-k nearest vectors in your space relative to the Euclidean distance metric and a query vector. Here's an example that retrieves the top 10 vectors from the `vector_tab` table that are the nearest to `query_vector` using the following exact similarity search query:

```
SELECT docID
FROM vector_tab
ORDER BY VECTOR_DISTANCE( embedding, :query_vector, EUCLIDEAN )
FETCH EXACT FIRST 10 ROWS ONLY;
```

In this example, `docID` and `embedding` are columns defined in the `vector_tab` table and `embedding` has the `VECTOR` data type.

In the case of Euclidean distances, comparing squared distances is equivalent to comparing distances. So, when ordering is more important than the distance values themselves, the Euclidean Squared distance is very useful as it is faster to calculate than the Euclidean distance (avoiding the square-root calculation). Consequently, it is simpler and faster to rewrite the query like this:

```
SELECT docID
FROM vector_tab
ORDER BY VECTOR_DISTANCE( embedding, :query_vector, EUCLIDEAN_SQUARED)
FETCH FIRST 10 ROWS ONLY;
```

> **Note:**
>
> The `EXACT` keyword is optional. If omitted while connected to an ADB-S instance, an approximate search using a vector index is attempted if one exists. For more information, see Perform Approximate Similarity Search Using Vector Indexes.

> **Note:**
>
> Ensure that you use the distance function that was used to train your embedding model.

> **See Also:**
>
> *Oracle Database SQL Language Reference* for the full syntax of the `ROW_LIMITING_CLAUSE`

Perform Approximate Similarity Search Using Vector Indexes

For a vector search to be useful, it needs to be fast and accurate. Approximate similarity searches seek a balance between these goals.

- Understand Approximate Similarity Search Using Vector Indexes
  For faster search speeds with large vector spaces, you can use approximate similarity search using vector indexes.
- Optimizer Plans for Vector Indexes
  Optimizer plans for HNSW and IVF indexes are described in the following sections.
- Approximate Similarity Search Examples
  Review these examples to see how you can perform an approximate similarity search using vector indexes.

Understand Approximate Similarity Search Using Vector Indexes

For faster search speeds with large vector spaces, you can use approximate similarity search using vector indexes.

Using vector indexes for a similarity search is called an **approximate search**. Approximate searches use **vector indexes**, which trade off accuracy for performance.

Approximate search for large vector spaces

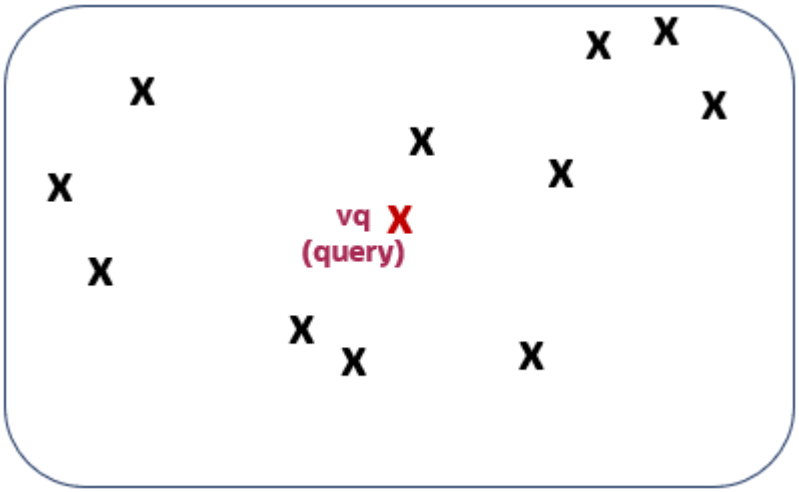
When search quality is your high priority and search speed is less important, Exact Similarity search is a good option. Search speed can be irrelevant for smaller vector spaces, or when you perform searches with high performance servers. However, ML algorithms often perform similarity searches on vector spaces with billions of embeddings. For example, the Deep1B data-set contains 1B images generated by a Convolutional Neural Network (CNN). Computing vector distances with every vector in the corpus to find Top-K matches at 100 percent accuracy is very slow.

Fortunately, there are many types of approximate searches that you can perform using vector indexes. Vector indexes can be less accurate, but they can consume less resources, and can be more efficient. Unlike traditional database indexes, vector indexes are constructed and perform searches using heuristic-based algorithms.

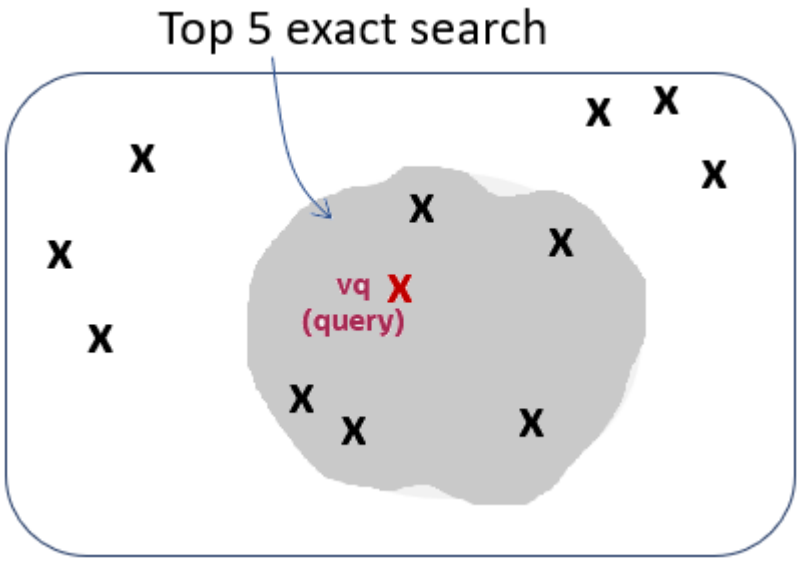
Because 100 percent accuracy cannot be guaranteed by heuristics, vector index searches use **target accuracy**. Internally, the algorithms used for both index creation and index search are doing their best to be as accurate as possible. However, you have the possibility to influence those algorithms by specifying a target accuracy. When creating the index or searching it, you can specify non-default target accuracy values either by specifying a percentage value, or by specifying internal parameters values, depending on the index type you are using.

Target accuracy example

To better understand what is meant by target accuracy look at the following diagrams. The first diagram illustrate a vector space where each vector is represented by a small cross. The one in red represents your query vector.

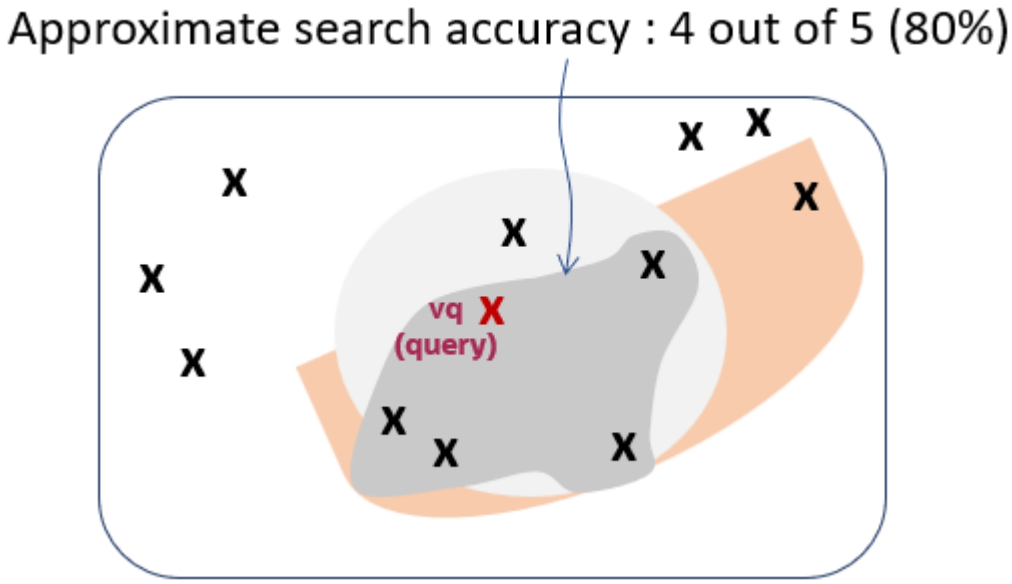


Running a top-5 exact similarity search in that context would return the five vectors shown on the second diagram:



Depending on how your vector index was constructed, running a top-5 approximate similarity search in that context could return the five vectors shown on the third diagram. This is because the index creation process is using heuristics. So searching through the vector index may lead to different results compared to an exact search:

As you can see, the retrieved vectors are different and, in this case, they differ by one vector. This means that, compared to the exact search, the similarity search retrieved 4 out of 5 vectors correctly. The similarity search has 80% accuracy compared to the exact similarity search. This is illustrated on the fourth diagram:



Due to the nature of vector indexes being approximate search structures, it's possible that fewer than K rows are returned in a top-K similarity search.

For information on how to set up your vector indexes, see Create Vector Indexes.

Optimizer Plans for Vector Indexes

Optimizer plans for HNSW and IVF indexes are described in the following sections.

- Optimizer Plans for HNSW Vector Indexes
  A Hierarchical Navigable Small World Graph (HNSW) is a form of In-Memory Neighbor Graph vector index. It is a very efficient index for vector approximate similarity search.

- Optimizer Plans for IVF Vector Indexes
  Inverted File Flat (IVF) is a form of Neighbor Partition Vector index. It is a partition-based index that achieves search efficiency by narrowing the search area through the use of neighbor partitions or clusters.
- Vector Index Hints
  If the optimizer does not choose your existing index while running your query and you still want that index to be used, you can rewrite your SQL statement to include vector index hints.

Optimizer Plans for HNSW Vector Indexes

A Hierarchical Navigable Small World Graph (HNSW) is a form of In-Memory Neighbor Graph vector index. It is a very efficient index for vector approximate similarity search.

In the simplest case, a query has a single table and it does not contain any relational filter predicates or subqueries. The following query example illustrates this situation:

```
SELECT chunk_id, chunk_data
FROM doc_chunks
ORDER BY VECTOR_DISTANCE( chunk_embedding, :query_vector, COSINE )
FETCH APPROX FIRST 4 ROWS ONLY WITH TARGET ACCURACY 80;
```

The corresponding execution plan should look like the following, if the optimizer decides to use the index (start from operation id 5):

Top-K similar vectors are first identified using the HNSW vector index. For each of them, corresponding rows are identified in the base table and required selected columns are extracted.

```
-----------------------------------------------------
| Id  | Operation                     | Name          |
-----------------------------------------------------
|   0 | SELECT STATEMENT              |               |
|*  1 |  COUNT STOPKEY                |               |
|   2 |   VIEW                        |               |
|*  3 |    SORT ORDER BY STOPKEY      |               |
|   4 |     TABLE ACCESS BY INDEX ROWID| DOC_CHUNKS    |
|   5 |      VECTOR INDEX HNSW SCAN   | DOCS_HNSW_IDX5 |
-----------------------------------------------------
```

> **Note:**
>
> The Hierarchical Navigable Small World (HNSW) vector index structure contains `rowids` of corresponding base table rows for each vector in the index.

However, your query may contain traditional relational data filters, as illustrated in the following example:

```
SELECT chunk_id, chunk_data
FROM doc_chunks
WHERE doc_id=1
```

```
ORDER BY VECTOR_DISTANCE( chunk_embedding, :query_vector, COSINE )
FETCH APPROX FIRST 4 ROWS ONLY WITH TARGET ACCURACY 80;
```

In that case, there are essentially two main alternatives for the optimizer to generate an execution plan using an HNSW vector index. These two alternatives are called **pre-filtering** and **in-filtering**.

The main difference between pre-filtering and in-filtering are:

- *Pre-filtering* runs the filtering evaluation on the base table first and only traverses the HNSW vector index for corresponding vectors. This can be very fast if the filter predicates are selective (that is, most rows filtered out).
- *In-filtering*, on the other hand, starts by traversing the HNSW vector index and invokes the filtering only for each vector matching candidate. This can be better than pre-filtering when more rows pass the filter predicates. In this case, the number of vector candidates to consider, while traversing the HNSW vector index, might be many fewer than the number of rows that pass the filters.

For both in-filtering and pre-filtering, the optimizer may choose to process projected columns from your select list before or after the similarity search operation. If it does so after, this is called a join-back operation. If it does so before, it is called a no-join-back operation. The tradeoff between the two depends on the number of rows returned by the similarity search.

To illustrate these four possibilities, here are some typical execution plans:

> **Note:**
>
> Depending on the version you are using, you can find variations of these plans. However, the idea stays the same, so the following plans are just for illustration purposes.

Pre-filter with Join-back (Start from Operation id 9)

Filtered `rowids` are first identified in the base table and joined to the auxiliary mapping table before the HNSW vector index identifies the relevant vectors. Relevant base table rows are then identified and required selected columns are extracted.

```
-------------------------------------------------------------------------------
--------------------------
| Id  | Operation                           |
Name                                                |
-------------------------------------------------------------------------------
--------------------------
|   0 | SELECT STATEMENT
|                                                       |
|*  1 |  COUNT STOPKEY
|                                                       |
|   2 |   VIEW
|                                                       |
|*  3 |    SORT ORDER BY STOPKEY
|                                                       |
|*  4 |     TABLE ACCESS BY INDEX ROWID       |
DOC_CHUNKS                                          |
|   5 |      VECTOR INDEX HNSW SCAN PRE-FILTER|
DOCS_HNSW_IDX3                                      |
```

```
|   6 |       VIEW                                |
VW_HPJ_56DFF779                                    |
|*  7 |        HASH JOIN RIGHT OUTER
|                                                      |
|   8 |         TABLE ACCESS FULL                 |
VECTOR$DOCS_HNSW_IDX3$81357_82726_0$HNSW_ROWID_VID_MAP |
|*  9 |         TABLE ACCESS FULL                 |
DOC_CHUNKS                                         |
-------------------------------------------------------------------------------
--------------------------
```

> **Note:**
>
> - HNSW indexes use an auxiliary table and associated index to store index information like mapping between vector `ids` and `rowids`. Mainly `VECTOR$<index name>$HNSW_ROWID_VID_MAP`.
> - For the join, you can also see plans like the following:
>
>   ```
>   NESTED LOOPS OUTER
>     TABLE ACCESS FULL  DOC_CHUNKS
>     TABLE ACCESS BY INDEX ROWID
>   VECTOR$DOCS_HNSW_IDX3$81357_82726_0$HNSW_ROWID_VID_MAP
>       INDEX UNIQUE SCAN SYS_C008586
>   ```

Pre-filter without Join-back (Start from Operation id 8)

Filtered `rowids` with relevant columns are first identified in the base table and buffered. The result is then joined to the auxiliary mapping table before the HNSW vector index identifies the top-K vectors. For those identified vectors, associated base table columns are shown.

```
-------------------------------------------------------------------------------
------------------------
| Id  | Operation                           |
Name                                               |
-------------------------------------------------------------------------------
------------------------
|   0 | SELECT STATEMENT
|                                                      |
|*  1 |  COUNT STOPKEY
|                                                      |
|   2 |   VIEW
|                                                      |
|*  3 |    SORT ORDER BY STOPKEY
|                                                      |
|   4 |     VECTOR INDEX HNSW SCAN PRE-FILTER|
DOCS_HNSW_IDX3                                     |
|   5 |      VIEW                           |
VW_HPF_B919B0A0                                    |
|*  6 |       HASH JOIN RIGHT OUTER
|                                                      |
|   7 |        TABLE ACCESS FULL            |
```

```
VECTOR$DOCS_HNSW_IDX3$81357_82726_0$HNSW_ROWID_VID_MAP |
|*  8 |        TABLE ACCESS FULL             |
DOC_CHUNKS                                          |
-------------------------------------------------------------------------------
------------------------
```

In-filter With Join-back (Start from Operation id 5)

HNSW vector index is traversed first, and for each identified vector, filters on the base table for the corresponding `rowid` are applied. Once the top-K `rowids` passing the filters are identified, base table columns are extracted.

```
-------------------------------------------------------------
| Id  | Operation                           | Name            |
-------------------------------------------------------------
|   0 | SELECT STATEMENT                    |                 |
|*  1 |  COUNT STOPKEY                      |                 |
|   2 |   VIEW                              |                 |
|*  3 |    SORT ORDER BY STOPKEY            |                 |
|*  4 |     TABLE ACCESS BY INDEX ROWID     | DOC_CHUNKS      |
|   5 |      VECTOR INDEX HNSW SCAN IN-FILTER| DOCS_HNSW_IDX3  |
|   6 |       VIEW                          | VW_HIJ_B919B0A0 |
|*  7 |        TABLE ACCESS BY USER ROWID   | DOC_CHUNKS      |
-------------------------------------------------------------
```

In-filter without Join-back (Start from Operation id 4)

HNSW vector index is traversed first, and for each identified vector, filters on the base table for the corresponding `rowid` are applied and relevant columns extracted. Once the top-K vectors are identified, corresponding base table columns are shown.

```
-------------------------------------------------------------
| Id  | Operation                           | Name            |
-------------------------------------------------------------
|   0 | SELECT STATEMENT                    |                 |
|*  1 |  COUNT STOPKEY                      |                 |
|   2 |   VIEW                              |                 |
|*  3 |    SORT ORDER BY STOPKEY            |                 |
|   4 |     VECTOR INDEX HNSW SCAN IN-FILTER| DOCS_HNSW_IDX3  |
|   5 |      VIEW                           | VW_HIF_B919B0A0 |
|*  6 |       TABLE ACCESS BY USER ROWID    | DOC_CHUNKS      |
-------------------------------------------------------------
```

HNSW Indexes in the Optimizer Plan

HNSW indexes plans may use an internal table and associated index to store index information like mapping between vector `ids` and `rowids`. Mainly `VECTOR$<`*`index name`*`>$HNSW_ROWID_VID_MAP`.

Table 8-1 HNSW Options

Operation	Options	Object_name
TABLE ACCESS	FULL	VECTOR$<*vector-index-name*>$<*id*>$HNSW_ROW_VID_MAP
TABLE ACCESS	STORAGE FULL	VECTOR$<*vector-index-name*>$<*id*>$HNSW_ROW_VID_MAP
TABLE ACCESS	INMEMORY FULL	VECTOR$<*vector-index-name*>$<*id*>$HNSW_ROW_VID_MAP

For the HNSW index itself, which is an In-Memory object, the plan uses a `VECTOR INDEX` operation. The object name `GALAXIES_HNSW_INX` is provided as an example of the user-specified HNSW index name:

Operation	Options	Object_name
VECTOR INDEX	HNSW SCAN	GALAXIES_HNSW_INX
VECTOR INDEX	HNSW SCAN PRE-FILTER	GALAXIES_HNSW_INX
VECTOR INDEX	HNSW SCAN IN-FILTER	GALAXIES_HNSW_INX

Optimizer Plans for IVF Vector Indexes

Inverted File Flat (IVF) is a form of Neighbor Partition Vector index. It is a partition-based index that achieves search efficiency by narrowing the search area through the use of neighbor partitions or clusters.

When using IVF vector indexes, you can see two possible plans for your similarity searches:

- *Pre-filtering* evaluates the filters first, before using the IVF vector index. Once the optimizer chooses which centroid partitions to scan, it evaluates the filters for all the rows in those partitions. Then it computes the distance to the query vector for the rows that passed the filters.
- *Post-filtering* evaluates the filters after using the IVF vector index. Once the optimizer chooses which centroid partitions to scan, it computes the distance to the query vector for all the rows in those partitions. Once the optimizer finds the closest rows, it then evaluates the filter for those rows and returns only the ones that pass the filters.

Consider the following query:

```
SELECT chunk_id, chunk_data,
FROM doc_chunks
WHERE doc_id=1
ORDER BY VECTOR_DISTANCE( chunk_embedding, :query_vector, COSINE )
FETCH APPROX FIRST 4 ROWS ONLY WITH TARGET ACCURACY 80;
```

The preceding query can lead to the following two different execution plans:

> **Note:**
>
> Depending on the version you are using, you can find variations of these plans. However, the idea stays the same, so the following plans are just for illustration purposes.

Pre-filter plan

- Plan line ids 5 to 9: This part happens first. The optimizer chooses the centroid `ids` that are closest to the query vector.
- Plan line ids 10 to 13: With these lines, the base table is joined to the identified centroid partitions using a `NESTED LOOPS` join. The base table is scanned to look for all the rows that pass the `WHERE` clause filter. For each of these rows, there is a look up in the index of the centroid partitions (plan line id 13). The `rowid` from the scan of the base table is used to find the row in the centroid partitions table that has a matching value for the base table `rowid` column.
- Plan line id 4: All the rows from 5-9 are joined to the rows from 10-13 with a `HASH JOIN`. That is, the rows that were filtered by 10-13 are reduced to only the rows from the partitions that were chosen as the closest ones.
- Plan line id 3: This step computes the vector distance and sorts on this value, only keeping the top-K rows.

```
-------------------------------------------------------------------------------
-----------------------------------
| Id  | Operation                                |
Name                                                                          |
-------------------------------------------------------------------------------
-----------------------------------
|   0 | SELECT STATEMENT
|                                                                               |
|*  1 |  COUNT STOPKEY
|                                                                               |
|   2 |   VIEW
|                                                                               |
|*  3 |    SORT ORDER BY STOPKEY
|                                                                               |
|*  4 |     HASH JOIN
|                                                                               |
|   5 |      VIEW                                |
VW_IVCR_2D77159E                                                             |
|*  6 |       COUNT STOPKEY
|                                                                               |
|   7 |        VIEW                              |
VW_IVCN_9A1D2119                                                             |
|*  8 |         SORT ORDER BY STOPKEY
|                                                                               |
|   9 |          TABLE ACCESS FULL               |
VECTOR$DOCS_IVF_IDX2$81357_82648_0$IVF_FLAT_CENTROIDS                        |
|  10 |      NESTED LOOPS
|                                                                               |
|* 11 |       TABLE ACCESS FULL                  |
DOC_CHUNKS                                                                   |
|  12 |       TABLE ACCESS BY GLOBAL INDEX ROWID|
```

```
VECTOR$DOCS_IVF_IDX2$81357_82648_0$IVF_FLAT_CENTROID_PARTITIONS |
|* 13 |        INDEX UNIQUE SCAN                |
SYS_C008661                                                    |
-------------------------------------------------------------------------------
-----------------------------------
```

> **Note:**
>
> IVF indexes use auxiliary partition tables and associated indexes to store index information. Mainly `VECTOR$<index name>$<ids>$IVF_FLAT_CENTROIDS` and `VECTOR$<index name>$<ids>$IVF_FLAT_CENTROIDS_PARTITIONS`.

Post-filter Plan

- Plan line ids 11 to 15: Here the optimizer chooses the closest centroids (similar to lines 5 through 9 of the pre-filter plan).
- Plan line id 10: This creates a special filter with information about which centroid `ids` were chosen.
- Plan line ids 16 and 17: These lines use the special filter from plan line 10 to scan only the corresponding centroid partitions.
- Plan line id 9: This `HASH JOIN` makes sure that only the rows with the closest centroid `ids` are returned.
- Plan line id 8: This computes the distance for those rows with the closest centroid `ids` and finds the top-K rows.
- Plan line id 4: This `NESTED LOOPS` applies the base table filter to the previously identified top-K rows.

```
-------------------------------------------------------------------------------
---------------------------------
| Id  | Operation                           |
Name                                                          |
-------------------------------------------------------------------------------
---------------------------------
|   0 | SELECT STATEMENT
|                                                                |
|*  1 |  COUNT STOPKEY
|                                                                |
|   2 |   VIEW
|                                                                |
|*  3 |    SORT ORDER BY STOPKEY
|                                                                |
|   4 |     NESTED LOOPS
|                                                                |
|   5 |      VIEW                           |
VW_IVPSR_11E7D7DE                                              |
|*  6 |       COUNT STOPKEY
|                                                                |
|   7 |        VIEW                         |
VW_IVPSJ_578B79F1                                              |
|*  8 |         SORT ORDER BY STOPKEY
|                                                                |
```

```
|*  9 |          HASH JOIN
|                                                                          |
|  10 |           PART JOIN FILTER CREATE
| :BF0000                                                                  |
|  11 |            VIEW                      |
VW_IVCR_B5B87E67                                                         |
|* 12 |             COUNT STOPKEY
|                                                                          |
|  13 |              VIEW                    |
VW_IVCN_9A1D2119                                                         |
|* 14 |               SORT ORDER BY STOPKEY
|                                                                          |
|  15 |                TABLE ACCESS FULL     |
VECTOR$DOCS_IVF_IDX4$81357_83292_0$IVF_FLAT_CENTROIDS                    |
|  16 |           PARTITION LIST JOIN-
FILTER|                                                                                |
|  17 |            TABLE ACCESS FULL         |
VECTOR$DOCS_IVF_IDX4$81357_83292_0$IVF_FLAT_CENTROID_PARTITIONS |
|* 18 |      TABLE ACCESS BY USER ROWID      |
DOC_CHUNKS                                                               |
--------------------------------------------------------------------------------
---------------------------------
```

IVF Indexes in the Optimizer Plan

If the IVF index is used by the optimizer, the plan contains the names of the centroids and centroid partitions tables accessed as well as corresponding indexes.

The value displayed in the `Options` column for tables accessed via IVF indexes depends upon whether the table scan is for a regular table or Exadata table.

Table 8-2 Centroids and Centroid Partition Table Options

Operation	Options	Object_name
TABLE ACCESS	FULL	VECTOR$<*vector-index-name*>$<*id*>$IVF_FLAT_CENTROIDS VECTOR$DOCS_IVF_IDX2$<*id*>$IVF_FLAT_CENTROID_PARTITIONS
TABLE ACCESS	STORAGE FULL	VECTOR$<*vector-index-name*>$<*id*>$IVF_FLAT_CENTROIDS VECTOR$DOCS_IVF_IDX2$<*id*>$IVF_FLAT_CENTROID_PARTITIONS
TABLE ACCESS	INMEMORY FULL	VECTOR$<*vector-index-name*>$<*id*>$IVF_FLAT_CENTROIDS VECTOR$DOCS_IVF_IDX2$<*id*>$IVF_FLAT_CENTROID_PARTITIONS

- Terminable Iteration for IVF Index
  Use this feature if you want to ensure that the desired number of rows are returned from a search.

Terminable Iteration for IVF Index

Use this feature if you want to ensure that the desired number of rows are returned from a search.

When using an Inverted File Flat (IVF) index, the optimizer estimates the number of probes (`nProbe`) or clusters to consider in the underlying centroid tables. For example, if the `nProbe = 5`, the five clusters nearest to the query vector are identified and the approximate search is exclusively made within these clusters and the results returned are based on vector proximity within these clusters.

Consider the following query where an IVF index is created on `chunk_embedding` (vector column) within the `doc_chunks` table. The complete example can be found here : SQL Quick Start Using a Vector Embedding Model Uploaded into the Database

```
SELECT chunk_id, chunk_data,
FROM doc_chunks
WHERE doc_id=1
ORDER BY VECTOR_DISTANCE( chunk_embedding, :query_vector, COSINE )
FETCH APPROX FIRST 10 ROWS ONLY WITH TARGET ACCURACY 80;
```

One of the existing problems is that the optimizer can under-estimate the number of probes. In this case, the query may return fewer rows than anticipated. In situations where the number of rows fetched is critical to an application, using terminable iteration with IVF indexes makes it possible to set a number (or range of rows) to be returned at a minimum. The underlying method would extend the search to more centroids ensuring that the needed K rows are returned. In order to use terminable iteration, you must specify the operation. If its optimizer cost is higher with a vector index, then the optimizer chooses not to use a vector index.

The terminable IVF index can be enabled by using one of the following methods:

Using Terminable IVF Index with Hint:

The terminable IVF index could be used by specifying a `IVF_ITERATION` as a hint in the query. For example, the above query could be rewritten using the hint as follows :

```
SELECT /*+ IVF_ITERATION*/ chunk_id, chunk_data,
FROM doc_chunks
WHERE doc_id=1
ORDER BY VECTOR_DISTANCE( chunk_embedding, :query_vector, COSINE )
FETCH APPROX FIRST 10 ROWS ONLY WITH TARGET ACCURACY 80;
```

Using Terminable IVF Index with Syntax:

The terminable iteration could also be specified by using the new syntax which explicitly sets the bounds for the number of rows that needs to be returned. The same example could be rewritten with the new syntax for terminable IVF index as follows:

```
SELECT chunk_id, chunk_data,
FROM doc_chunks
WHERE doc_id=1
```

```
ORDER BY VECTOR_DISTANCE( chunk_embedding, :query_vector, COSINE )
FETCH APPROX FIRST 10 to 20 ROWS ONLY WITH TARGET ACCURACY 80;
```

In the above example, - **10 to 20** - enables terminable iteration. The lower bound (in this example, 10) indicates that at least 10 rows are returned by performing terminable iteration. The upper bound (in this example, 20) indicates that the query will return at most 20 rows. Lower bound and upper bound can be identical, this would still enable the terminable iteration. No rows are returned if the lower bound is larger than the upper bound or if the upper bound is zero or negative. If there are not enough centroids available to extend the search, it is possible that the query would return less than 10 rows.

Vector Index Hints

If the optimizer does not choose your existing index while running your query and you still want that index to be used, you can rewrite your SQL statement to include vector index hints.

There are two types of vector index hints:

- *Access path hints* apply when running a simple query block with a single table with no predicates and a valid HNSW vector index. This only applies to HNSW vector indexes.
- *Query transformation hints* cover all other cases, where query transformations are used.

Syntax for Access Path Hints

The access path hints are modeled on the existing `INDEX` hints. For example:

```
VECTOR_INDEX_SCAN ( [ @ queryblock ] tablespec [ indexspec ])

NO_VECTOR_INDEX_SCAN ( [ @ queryblock ] tablespec [ indexspec ])
```

The `VECTOR_INDEX_SCAN` hint instructs the optimizer to use a vector index scan for the specified table. The index is optionally specified. The optimizer does not consider a full table scan.

`NO_VECTOR_INDEX_SCAN` disables the use of the optionally specified vector index scan.

> **See Also:**
>
> - *Oracle Database SQL Language Reference* for details about the `INDEX` hint

Access Path Hint Examples

```
SELECT /*+ VECTOR_INDEX_SCAN(galaxies) */ name
FROM galaxies
ORDER BY VECTOR_DISTANCE( embedding, to_vector('[0,1,1,0,0]'), COSINE )
FETCH FIRST 3 ROWS ONLY;


SELECT /*+ NO_VECTOR_INDEX_SCAN(galaxies) */ name
FROM galaxies
ORDER BY VECTOR_DISTANCE( embedding, to_vector('[0,1,1,0,0]'), COSINE )
FETCH FIRST 3 ROWS ONLY;
```

Note that even though the second example query is expected not to use a vector index, it is just a hint that the optimizer may or may not follow. To be certain that the index will not be used, it is better to use `EXACT` rather than the hint, as in the following:

```
SELECT name
FROM galaxies
ORDER BY VECTOR_DISTANCE( embedding, to_vector('[0,1,1,0,0]'), COSINE )
FETCH EXACT FIRST 3 ROWS ONLY;
```

Syntax for Query Transformation Hints

The syntax for query transformation hints is similar to the `INDEX` hints model. For example:

```
VECTOR_INDEX_TRANSFORM ( [ @ queryblock ] tablespec [ indexspec
[ filtertype ]] )

NO_VECTOR_INDEX_TRANSFORM ( [ @ queryblock ] tablespec [ indexspec ] )
```

`VECTOR_INDEX_TRANSFORM` forces the transformation and uses the optionally specified index, whereas `NO_VECTOR_INDEX_TRANSFORM` disables the transformation and the use of the optionally specified index. For more information about the potential query transformations, see Optimizer Plans for HNSW Vector Indexes and Optimizer Plans for IVF Vector Indexes.

The optional `filtertype` specification for the `VECTOR_INDEX_TRANSFORM` hint can have the following values:

- `PRE_FILTER_WITH_JOIN_BACK` (only applies to HNSW indexes)
- `PRE_FILTER_WITHOUT_JOIN_BACK` (applies to both HNSW and IVF indexes)
- `IN_FILTER_WITH_JOIN_BACK` (only applies to HNSW indexes)
- `IN_FILTER_WITHOUT_JOIN_BACK` (only applies to HNSW indexes)
- `POST_FILTER_WITHOUT_JOIN_BACK` (only applies to IVF indexes)

Each `filtertype` value corresponds to exactly one state in the query transformation. If no `filtertype` value is specified in the `VECTOR_INDEX_TRANSFORM` hint, then all of the valid filtering types are considered and the best one is chosen.

> **Note:**
>
> If you specify the `filtertype` value, you must also specify the `indexspec` value.

> **See Also:**
>
> - *Oracle Database SQL Language Reference* for details about the `INDEX` hint

Query Transformation Hint Examples

```
SELECT /*+ vector_index_transform(galaxies galaxies_ivf_idx
pre_filter_without_join_back) */ name
```

```
FROM galaxies
WHERE id<5
ORDER BY VECTOR_DISTANCE( embedding, to_vector('[0,1,1,0,0]'), COSINE )
FETCH FIRST 3 ROWS ONLY WITH TARGET ACCURACY 90;


SELECT /*+ vector_index_transform(galaxies) */ name
FROM galaxies
WHERE id<5
ORDER BY VECTOR_DISTANCE( embedding, to_vector('[0,1,1,0,0]'), COSINE )
FETCH FIRST 10 ROWS ONLY WITH TARGET ACCURACY 90;


SELECT /*+ no_vector_index_transform(galaxies) */ name
FROM galaxies
WHERE id<5
ORDER BY VECTOR_DISTANCE( embedding, to_vector('[0,1,1,0,0]'), COSINE )
FETCH FIRST 10 ROWS ONLY WITH TARGET ACCURACY 90;


SELECT /*+ vector_index_transform(galaxies galaxies_hnsw_idx
in_filter_with_join_back) */ name
FROM galaxies
WHERE id<5
ORDER BY VECTOR_DISTANCE( embedding, to_vector('[0,1,1,0,0]'), COSINE )
FETCH FIRST 10 ROWS ONLY WITH TARGET ACCURACY 90;
```

Note:

- The auxiliary tables for vector indexes are regular heap tables and require up-to-date optimizer statistics for accurate costing. The automatic optimizer statistics collection task gathers statistics for the auxiliary tables only when collecting statistics on their base tables.
- Using the `DBMS_STATS` package, you can perform manual statistics gathering operations on the auxiliary tables. However, `GATHER_INDEX_STATS` on vector indexes is not permitted. The in-memory HNSW index automatically maintains its statistics in-memory.

See Also:

- *Oracle Database SQL Tuning Guide* for information about configuring automatic optimizer statistics collection

The SQL examples included previously are based on the following:

```
DROP TABLE galaxies PURGE;
CREATE TABLE galaxies (id NUMBER, name VARCHAR2(50), doc VARCHAR2(500),
EMBEDDING VECTOR);
-- or CREATE TABLE galaxies (id NUMBER, name VARCHAR2(50), doc VARCHAR2(500),
```

```
EMBEDDING VECTOR(5,INT8));

INSERT INTO galaxies VALUES (1, 'M31', 'Messier 31 is a barred spiral galaxy
in the Andromeda constellation which has a lot of barred spiral galaxies.',
'[0,2,2,0,0]');
INSERT INTO galaxies VALUES (2, 'M33', 'Messier 33 is a spiral galaxy in the
Triangulum constellation.', '[0,0,1,0,0]');
INSERT INTO galaxies VALUES (3, 'M58', 'Messier 58 is an intermediate barred
spiral galaxy in the Virgo constellation.', '[1,1,1,0,0]');
INSERT INTO galaxies VALUES (4, 'M63', 'Messier 63 is a spiral galaxy in the
Canes Venatici constellation.', '[0,0,1,0,0]');
INSERT INTO galaxies VALUES (5, 'M77', 'Messier 77 is a barred spiral galaxy
in the Cetus constellation.', '[0,1,1,0,0]');
INSERT INTO galaxies VALUES (6, 'M91', 'Messier 91 is a barred spiral galaxy
in the Coma Berenices constellation.', '[0,1,1,0,0]');
INSERT INTO galaxies VALUES (7, 'M49', 'Messier 49 is a giant elliptical
galaxy in the Virgo constellation.', '[0,0,0,1,1]');
INSERT INTO galaxies VALUES (8, 'M60', 'Messier 60 is an elliptical galaxy in
the Virgo constellation.', '[0,0,0,0,1]');
INSERT INTO galaxies VALUES (9, 'NGC1073', 'NGC 1073 is a barred spiral
galaxy in Cetus constellation.', '[0,1,1,0,0]');
```

For HNSW indexes:

```
CREATE VECTOR INDEX galaxies_hnsw_idx ON galaxies (embedding)
ORGANIZATION INMEMORY NEIGHBOR GRAPH
DISTANCE COSINE
WITH TARGET ACCURACY 95;
```

For IVF indexes:

```
CREATE VECTOR INDEX galaxies_ivf_idx ON galaxies(embedding)
ORGANIZATION NEIGHBOR PARTITIONS
DISTANCE COSINE
WITH TARGET ACCURACY 95;
```

Approximate Similarity Search Examples

Review these examples to see how you can perform an approximate similarity search using vector indexes.

- Approximate Search Using HNSW
  This example shows how you can create the Hierarchical Navigable Small World (HNSW) index and run an approximate search using that index.
- Approximate Search Using IVF
  This example shows how you can create the Inverted File Flat (IVF) index and run an approximate search using that index.

Approximate Search Using HNSW

This example shows how you can create the Hierarchical Navigable Small World (HNSW) index and run an approximate search using that index.

```
create table galaxies (id number, name varchar2(50), doc varchar2(500),
embedding vector(5,INT8));

insert into galaxies values (1, 'M31', 'Messier 31 is a barred spiral galaxy
in the Andromeda constellation which has a lot of barred spiral galaxies.',
'[0,2,2,0,0]');
insert into galaxies values (2, 'M33', 'Messier 33 is a spiral galaxy in the
Triangulum constellation.', '[0,0,1,0,0]');
insert into galaxies values (3, 'M58', 'Messier 58 is an intermediate barred
spiral galaxy in the Virgo constellation.', '[1,1,1,0,0]');
insert into galaxies values (4, 'M63', 'Messier 63 is a spiral galaxy in the
Canes Venatici constellation.', '[0,0,1,0,0]');
insert into galaxies values (5, 'M77', 'Messier 77 is a barred spiral galaxy
in the Cetus constellation.', '[0,1,1,0,0]');
insert into galaxies values (6, 'M91', 'Messier 91 is a barred spiral galaxy
in the Coma Berenices constellation.', '[0,1,1,0,0]');
insert into galaxies values (7, 'M49', 'Messier 49 is a giant elliptical
galaxy in the Virgo constellation.', '[0,0,0,1,1]');
insert into galaxies values (8, 'M60', 'Messier 60 is an elliptical galaxy in
the Virgo constellation.', '[0,0,0,0,1]');
insert into galaxies values (9, 'NGC1073', 'NGC 1073 is a barred spiral
galaxy in Cetus constellation.', '[0,1,1,0,0]');

...

commit;
```

In the galaxies example, this is how you can create the HNSW index and how you would run an approximate search using that index:

```
CREATE VECTOR INDEX galaxies_hnsw_idx ON galaxies (embedding) ORGANIZATION
INMEMORY NEIGHBOR GRAPH
DISTANCE COSINE
WITH TARGET ACCURACY 95;

SELECT name
FROM galaxies
WHERE name <> 'NGC1073'
ORDER BY VECTOR_DISTANCE( embedding, to_vector('[0,1,1,0,0]'), COSINE )
FETCH APPROXIMATE FIRST 3 ROWS ONLY;
```

This approximate search example inherits a target accuracy of 95 as it is set when the index is defined. You could override the `TARGET ACCURACY` of your search by running the following query examples:

```
SELECT name
FROM galaxies
WHERE name <> 'NGC1073'
```

```
ORDER BY VECTOR_DISTANCE( embedding, to_vector('[0,1,1,0,0]'), COSINE  )
FETCH APPROXIMATE FIRST 3 ROWS ONLY WITH TARGET ACCURACY 90;


SELECT name
FROM galaxies
WHERE name <> 'NGC1073'
ORDER BY VECTOR_DISTANCE( embedding, to_vector('[0,1,1,0,0]'), COSINE  )
FETCH APPROXIMATE FIRST 3 ROWS ONLY WITH TARGET ACCURACY PARAMETERS (efsearch
500);
```

You can also create the index using the following syntax:

```
CREATE VECTOR INDEX galaxies_hnsw_idx ON galaxies (embedding) ORGANIZATION
INMEMORY NEIGHBOR GRAPH
DISTANCE COSINE
WITH TARGET ACCURACY 90 PARAMETERS (type HNSW, neighbors 40, efconstruction
500);
```

> **Note:**
>
> The `APPROX` and `APPROXIMATE` keywords are optional. If omitted while connected to an ADB-S instance, an approximate search using a vector index is attempted if one exists.

> **See Also:**
>
> *Oracle Database SQL Language Reference* for the full syntax of the `ROW_LIMITING_CLAUSE`

Approximate Search Using IVF

This example shows how you can create the Inverted File Flat (IVF) index and run an approximate search using that index.

```
create table galaxies (id number, name varchar2(50), doc varchar2(500),
embedding vector(5,INT8));

insert into galaxies values (1, 'M31', 'Messier 31 is a barred spiral galaxy
in the Andromeda constellation which has a lot of barred spiral galaxies.',
'[0,2,2,0,0]');
insert into galaxies values (2, 'M33', 'Messier 33 is a spiral galaxy in the
Triangulum constellation.', '[0,0,1,0,0]');
insert into galaxies values (3, 'M58', 'Messier 58 is an intermediate barred
spiral galaxy in the Virgo constellation.', '[1,1,1,0,0]');
insert into galaxies values (4, 'M63', 'Messier 63 is a spiral galaxy in the
Canes Venatici constellation.', '[0,0,1,0,0]');
insert into galaxies values (5, 'M77', 'Messier 77 is a barred spiral galaxy
in the Cetus constellation.', '[0,1,1,0,0]');
```

```
insert into galaxies values (6, 'M91', 'Messier 91 is a barred spiral galaxy
in the Coma Berenices constellation.', '[0,1,1,0,0]');
insert into galaxies values (7, 'M49', 'Messier 49 is a giant elliptical
galaxy in the Virgo constellation.', '[0,0,0,1,1]');
insert into galaxies values (8, 'M60', 'Messier 60 is an elliptical galaxy in
the Virgo constellation.', '[0,0,0,0,1]');
insert into galaxies values (9, 'NGC1073', 'NGC 1073 is a barred spiral
galaxy in Cetus constellation.', '[0,1,1,0,0]');

...

commit;
```

In the galaxies example, this is how you can create the IVF index and how you can run an approximate search using that index:

```
CREATE VECTOR INDEX galaxies_ivf_idx ON galaxies (embedding) ORGANIZATION
NEIGHBOR PARTITIONS
DISTANCE COSINE
WITH TARGET ACCURACY 95;

SELECT name
FROM galaxies
WHERE name <> 'NGC1073'
ORDER BY VECTOR_DISTANCE( embedding, to_vector('[0,1,1,0,0]'), COSINE )
FETCH APPROXIMATE FIRST 3 ROWS ONLY;
```

If the index is used by the optimizer, then this approximate search example inherits a target accuracy of `95` as it is set when the index is defined. You can override the `TARGET ACCURACY` of your search by running the following query examples:

```
SELECT name
FROM galaxies
WHERE name <> 'NGC1073'
ORDER BY VECTOR_DISTANCE( embedding, to_vector('[0,1,1,0,0]'), COSINE )
FETCH APPROXIMATE FIRST 3 ROWS ONLY WITH TARGET ACCURACY 90;


SELECT name
FROM galaxies
WHERE name <> 'NGC1073'
ORDER BY VECTOR_DISTANCE( embedding, to_vector('[0,1,1,0,0]') )
FETCH APPROXIMATE FIRST 3 ROWS ONLY WITH TARGET ACCURACY PARAMETERS
( NEIGHBOR PARTITION PROBES 10 );
```

You can also create the index using the following syntax:

```
CREATE VECTOR INDEX galaxies_ivf_idx ON galaxies (embedding) ORGANIZATION
NEIGHBOR PARTITIONS
DISTANCE COSINE
WITH TARGET ACCURACY 90 PARAMETERS (type IVF, neighbor partitions 100);
```

> **Note:**
>
> The `APPROX` and `APPROXIMATE` keywords are optional. If omitted while connected to an ADB-S instance, an approximate search using a vector index is attempted if one exists.

> **See Also:**
>
> *Oracle Database SQL Language Reference* for the full syntax of the `ROW_LIMITING_CLAUSE`

Perform Multi-Vector Similarity Search

Another major use-case of vector search is multi-vector search. Multi-vector search is typically associated with a multi-document search, where documents are split into chunks that are individually embedded into vectors.

A multi-vector search consists of retrieving top-K vector matches using grouping criteria known as partitions based on the documents' characteristics. This ability to score documents based on the similarity of their chunks to a query vector being searched is facilitated in SQL using the partitioned row limiting clause.

With multi-vector search, it is easier to write SQL statements to answer the following type of question:

- If they exist, what are the four best matching sentences found in the three best matching paragraphs of the two best matching books?

For example, imagine if each book in your database is organized into paragraphs containing sentences which have vector embedding representations, then you can answer the previous question using a single SQL statement such as:

```
SELECT bookId, paragraphId, sentence
FROM books
ORDER BY vector_distance(sentence_embedding, :sentence_query_vector)
FETCH FIRST 2 PARTITIONS BY bookId, 3 PARTITIONS BY paragraphId, 4 ROWS ONLY;
```

You can also use an approximate similarity search instead of an exact similarity search as shown in the following example:

```
SELECT bookId, paragraphId, sentence
FROM books
ORDER BY vector_distance(sentence_embedding, :sentence_query_vector)
FETCH APPROXIMATE FIRST 2 PARTITIONS BY bookId, 3 PARTITIONS BY paragraphId,
4 ROWS ONLY
WITH TARGET ACCURACY 90;
```

> **Note:**
>
> All the rows returned are ordered by `VECTOR_DISTANCE()` and not grouped by the partition clause.

Semantically, the previous SQL statement is interpreted as:

- Sort all records in the books table in descending order of the vector distance between the sentences and the query vector.
- For each record in this order, check its `bookId` and `paragraphId`. This record is produced if the following three conditions are met:
  1. Its `bookId` is one of the first two distinct `bookId` in the sorted order.
  2. Its `paragraphId` is one of the first three distinct `paragraphId` in the sorted order within the same `bookId`.
  3. Its record is one of the first four records within the same `bookId` and `paragraphId` combination.
- Otherwise, this record is filtered out.

Multi-vector similarity search is not just for documents and can be used to answer the following questions too:

- Return the top K closest matching photos but ensure that they are photos of different people.
- Find the top K songs with two or more audio segments that best match this sound snippet.

> **Note:**
>
> - This partition row-limiting clause extension is a generic extension of the SQL language. It does not have to apply just to vector searches.
> - Multi-vector search with the partitioning row-limit clause does not use vector indexes.

> **See Also:**
>
> *Oracle Database SQL Language Reference* for the full syntax of the `ROW_LIMITING_CLAUSE`

Perform Hybrid Search

Hybrid search is an advanced information retrieval technique that lets you search documents by *keywords* and *vectors*, to achieve more relevant search results.

- Understand Hybrid Search
  With hybrid search, you can search through your documents by performing a combination of full-text queries and vector-based similarity queries, using out-of-the-box or custom scoring techniques.
- Query Hybrid Vector Indexes End-to-End Example
  In this example, you can see an end-to-end hybrid search workflow. First, you run the `CREATE HYBRID VECTOR INDEX` SQL statement that prepares, chunks, embeds, stores, and indexes your input data. You then perform vector search alongside keyword search using the `DBMS_HYBRID_VECTOR.SEARCH` PL/SQL query API.

Understand Hybrid Search

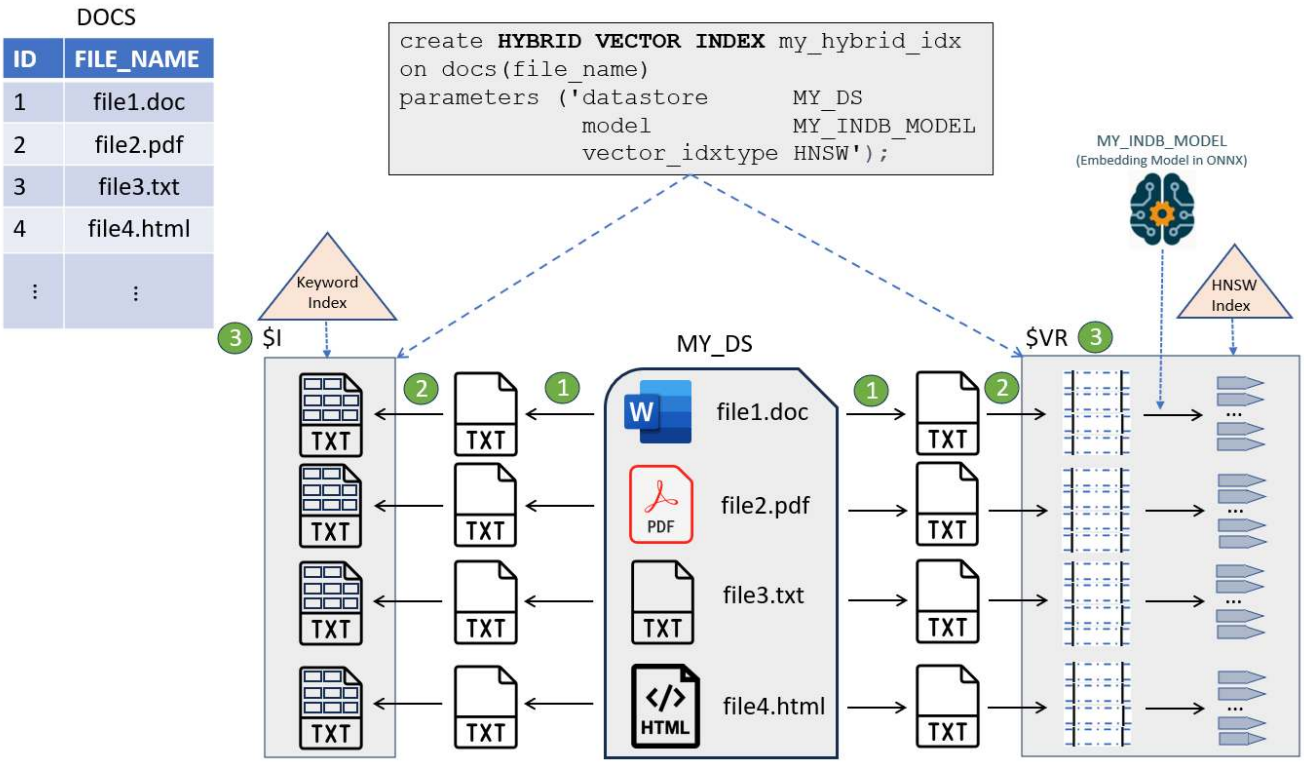
With hybrid search, you can search through your documents by performing a combination of full-text queries and vector-based similarity queries, using out-of-the-box or custom scoring techniques.

Here are the key points to note when using hybrid search:

- Hybrid searches are run against hybrid vector indexes (as explained in Manage Hybrid Vector Indexes). You use the `DBMS_HYBRID_VECTOR.SEARCH` PL/SQL function to query a hybrid vector index.
- After you create a hybrid vector index, you have five possibilities to use the index by querying it in multiple **search modes** summarized here:
  - Pure Semantic in Document Mode
  - Pure Semantic in Chunk Mode
  - Pure Keyword in Document Mode
  - Keyword and Semantic in Document Mode
  - Keyword and Semantic in Chunk Mode
- When performing hybrid search by both keywords and vectors, the results are combined (or fused) into a single result set. The results of such a hybrid search are based on a scoring mechanism that is important to understand to be able to interpret the results.

  Similarity search uses the notion of `VECTOR_DISTANCE` values to decide on the ranking of chunks, whereas the traditional Oracle Text search uses the notion of **keyword score** also known as `CONTAINS` score. These two metrics are very different and one cannot be used directly to compare to the other. Hence, similarity search distances are converted or normalized into a `CONTAINS`-score equivalent that is called **semantic score** so its value will range between 100 (best) to 0 (worse). That way, keyword score and semantic score are comparable when running a hybrid search.

For the sake of the following explanations, we are using the same use case as seen in Understand Hybrid Vector Indexes:



Pure Semantic in Document Mode

The pure semantic in document mode performs vector-only search to fetch document-level results.

The following SQL statement queries a hybrid vector index in this mode:

```
select json_Serialize(
  DBMS_HYBRID_VECTOR.SEARCH(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "vector":
          {
             "search_text"   : "galaxies formation and massive black holes",
             "search_mode"   : "DOCUMENT",
             "aggregator"    : "AVG"
          },
         "return":
          {
             "values"        : [ "rowid", "score", "vector_score" ],
             "topN"          : 10
          }
      }'
    )
  ) RETURNING CLOB pretty);
```

The result of this query may look like the following, where you see the `ROWID`s of the `DOCS` table rows corresponding to documents as well as its vector score (`vector_score`). Here, the final score (`score`) is the same as the vector score because there is no keyword score in this case.

Here is an excerpt from the results:

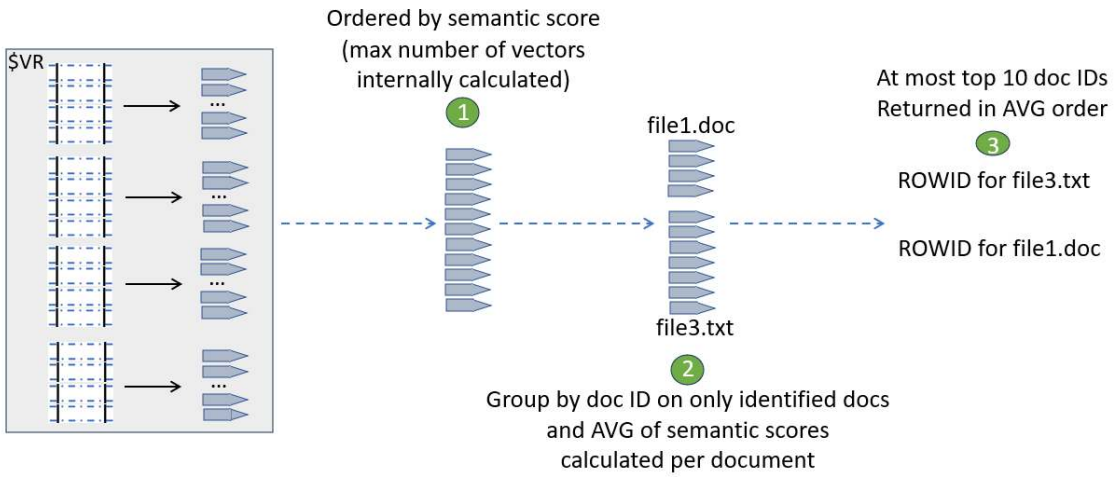
```
[
{
    "rowid"        : "AAASBEAAEAAAUpqAAB",
    "score"        : 71.04,
    "vector_score" : 71.04
  },
  {
    "rowid"        : "AAASBEAAEAAAWBKAAE",
    "score"        : 67.82,
    "vector_score" : 67.82
  },
]
```

Here is how to interpret this statement:

1. The system runs a similarity search on all the vectors and extracts the top k ones at most. The value k is internally calculated. Each is given a vector score.
2. These k vectors (at most) are then grouped by document IDs and for each identified document, the semantic score of each associated vector found for that document is used to compute the average (in this example) of these scores for that particular document.
3. The top 10 documents (at most) with the highest averages are returned.

This is illustrated here:

```
DBMS_HYBRID_VECTOR.SEARCH(json('{
    "hybrid_index name" : "my_hybrid_idx",
    "vector" : { "search_text" : "galaxies formation and massive black holes",
                 "search_mode" : "DOCUMENT",
                 "aggregator"  : "AVG" },
    "return" : { "values"      : [ "rowid", "score", "vector_score" ],
                 "topN"        : 10 }}'))
```



Pure Semantic in Chunk Mode

The pure semantic in chunk mode is the SQL semantic search equivalent. This mode performs a vector-only search to fetch chunk results.

The following SQL statement queries a hybrid vector index in this mode:

```
select json_Serialize(
  dbms_hybrid_vector.search(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "vector":
          {
             "search_text"   : "galaxies formation and massive black holes",
             "search_mode"   : "CHUNK"
          },
         "return":
          {
             "values"        : [ "score", "chunk_text", "chunk_id" ],
             "topN"          : 3
          }
      }'
    )
  ) RETURNING CLOB pretty);
```

Here is how to interpret this statement:

1. The system runs a similarity search on all the vectors and extracts the top k ones at most. The value k is internally calculated. Each is given a vector score.
2. The top 3 chunks (at most) with the highest scores are returned.

The result of this query may look like the following, where you can see the chunks corresponding to their semantic scores:

```
[
  {
    "score"      : 61,
    "chunk_text" : "Galaxies form through a complex process that begins with
small fluctuations in the density of matter in the early
universe. Massive black holes, typically found at the centers of galaxies,
are believed to play a crucial role in their formation and evolution.",
    "chunk_id"   : "1"
  },
  {
    "score"      : 56.64,
    "chunk_text" : "The presence of massive black holes in galaxies is
closely linked to their morphological characteristics and star formation
rates.
Observations suggest that as galaxies evolve, their central black holes grow
in tandem with their host galaxy's mass.",
    "chunk_id"   : "3"
  },
  {
    "score"      : 55.75,
    "chunk_text" : "Black holes grow by accreting gas and merging with other
black holes. Their gravitational influence can regulate star
formation and drive powerful jets of energy, which can impact the surrounding
galaxy.",
    "chunk_id"   : "2"
```

```
  }
]
```

Pure Keyword in Document Mode

The pure keyword search in document mode is equivalent to the traditional `CONTAINS` query using Oracle Text. This mode performs a text-only search to fetch document-level results.

The following SQL statement queries a hybrid vector index in this mode:

```
select json_Serialize(
  dbms_hybrid_vector.search(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "text":
          {
             "contains"      : "galaxies, black holes"
          },
         "return":
          {
             "values"        : [ "rowid", "score" ],
             "topN"          : 3
          }
      }'
    )
  ) RETURNING CLOB pretty);
```

Here is how to interpret this statement:

1. The system runs a `CONTAINS` query that returns a maximum number of documents. This maximum number is internally calculated. Each document is given a keyword score.
2. The top 3 documents (at most) with the highest scores are returned.

The result of this query may look like the following, where you can see the `ROWID`s of the `DOCS` table rows corresponding to documents as well as their keyword scores:

```
[
  {
    "rowid" : "AAAR9jAABAAAQeaAAB",
    "score" : 68
  },
  {
    "rowid" : "AAAR9jAABAAAQeaAAA",
    "score" : 35
  },
  {
    "rowid" : "AAAR9jAABAAAQeaAAD",
    "score" : 2
  }
]
```

Keyword and Semantic in Document Mode

Let us examine a non-pure case of hybrid search where keyword scores and semantic scores are combined.

The following SQL statement performs a keyword and semantic search to fetch document-level results:

```
select json_Serialize(
  DBMS_HYBRID_VECTOR.SEARCH(
    json(
      '{
         "hybrid_index_name" : "my_hybrid_idx",
         "search_scorer"     : "rsf",
         "search_fusion"     : "UNION",
         "vector":
          {
             "search_text"   : "How can I search with hybrid vector indexes?",
             "search_mode"   : "DOCUMENT",
             "aggregator"    : "MAX",
             "score_weight"  : 1,
             "rank_penalty"  : 5
          },
         "text":
          {
             "contains"      : "hybrid AND vector AND index"
             "score_weight"  : 10,
             "rank_penalty"  : 1
          },
         "return":
          {
             "values"        : [ "rowid", "score", "vector_score",
"text_score" ],
             "topN"          : 10
          }
      }'
    )
  ) RETURNING CLOB pretty);
```

Here is how to interpret this statement:

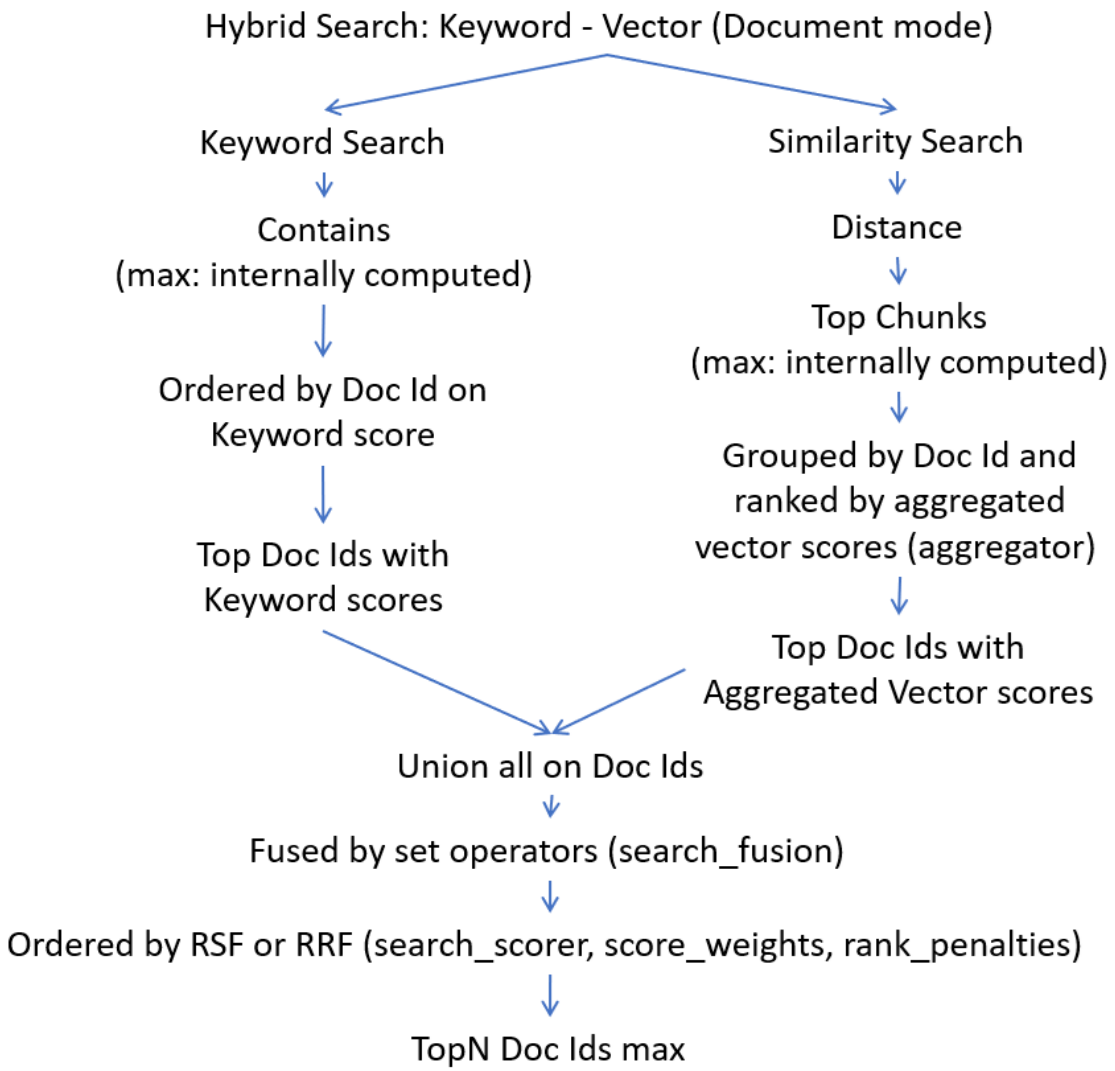
In document mode, the result of your search is a list of `ROWID`s from your base table corresponding to the list of best files identified.

To get to this list, two searches are conducted:

- **Keyword search**: During this search, the system uses the `CONTAINS` string representing the Oracle Text `CONTAINS` search expression for the searched keywords. The result of this operation is a list of document identifiers satisfying your `CONTAINS` expression. The numbers of document identifiers to retrieve, at max, is internally calculated.
- **Similarity search**: During this search, the system performs a similarity search with the query vector (created from the `SEARCH_TEXT` string) against the vector index of all the chunks of all your documents. The maximum number of chunks to retrieve is also internally calculated. It then assigns a vector score to each chunk retrieved. Because you want to run this search in `DOCUMENT SEARCH_MODE`, the result of this similarity search is first grouped by document identifier. The process now aggregates the vector scores for each document, identified using the `AGGREGATOR` function. The result of this similarity search is a list of document identifiers satisfying your `SEARCH_TEXT` similarity query string.

After the searches complete, the system needs to merge the results and score them as illustrated here:

Figure 8-2 Scoring for Keyword and Semantic Search in Document Mode

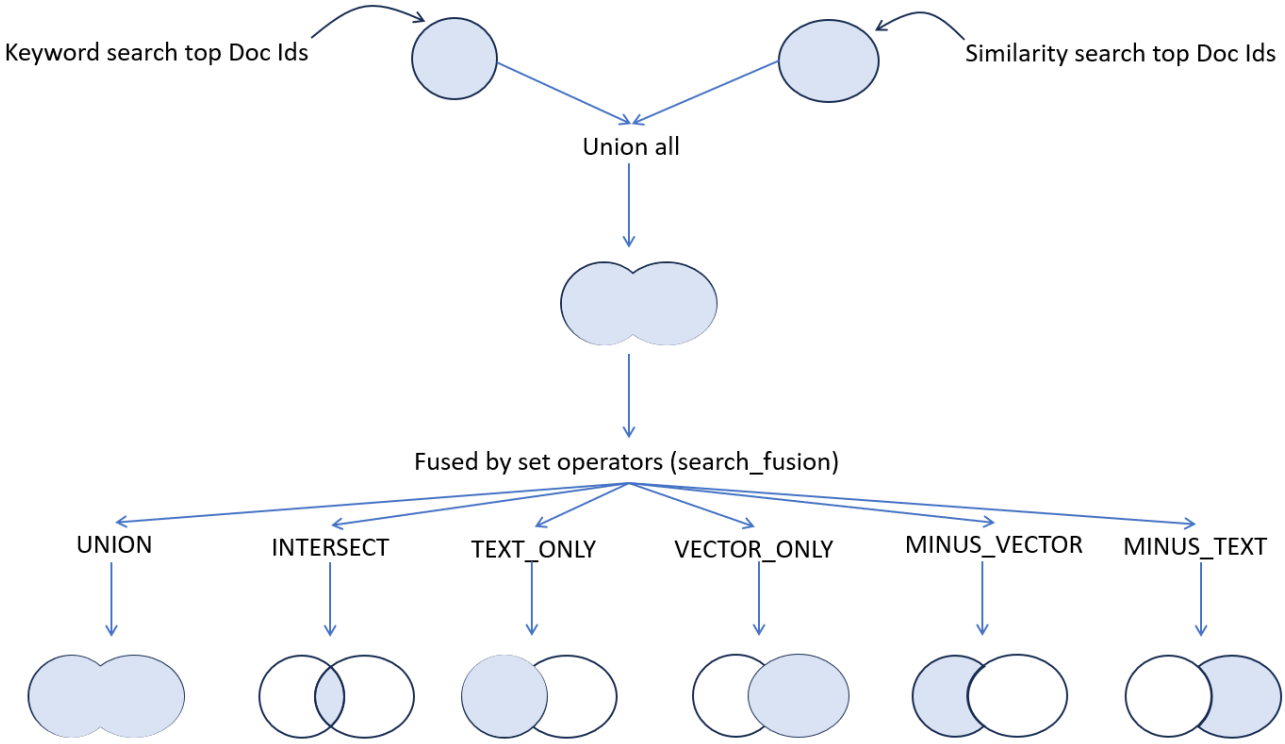


As outlined in the preceding diagram,

- First, both search results are added using a `UNION ALL` operation.
- Before final scoring, you have the possibility to define what you want to retain from this intermediate result set by specifying the `SEARCH_FUSION` operation.
- Then comes the time where final scoring is computed using the defined `SEARCH_SCORER` algorithm such as Reciprocal Rank Fusion (RRF) or Relative Score Fusion (RSF). The final scoring can use the specified `SCORE_WEIGHT` and `RANK_PENALTY` values for each retrieved document identifier, depending from which sort operation they are coming from.
- Finally, the defined `topN` document identifiers are returned at most.

The possible fuse operators are illustrated here:

Figure 8-3 Fusion Operation in Document Search Mode



Keyword and Semantic in Chunk Mode

Let us examine another non-pure case of hybrid search where keyword scores and semantic scores are combined to fetch chunk results.

The following SQL statement performs a keyword and semantic search in chunk mode:

```
select json_Serialize(
  DBMS_HYBRID_VECTOR.SEARCH(
    json(
         '{
            "hybrid_index_name" : "my_hybrid_vector_idx",
            "search_scorer"     : "rsf",
            "search_fusion"     : "UNION",
            "vector":
                      {
                        "search_text"   : "How can I search with hybrid
vector indexes?",
                        "search_mode"   : "CHUNK",
                        "score_weight"  : 1
                      },
            "text":
                      {
                       "contains"       : "hybrid AND vector AND index",
                       "score_weight"   : 1
                      },
            "return":
                      {
```

```
                        "values"        : [ "chunk_id", "score",
"vector_score", "text_score" ],
                        "topN"          : 10
                      }
          }'
    )
  ) RETURNING CLOB pretty);
```

Here is how to interpret this statement:

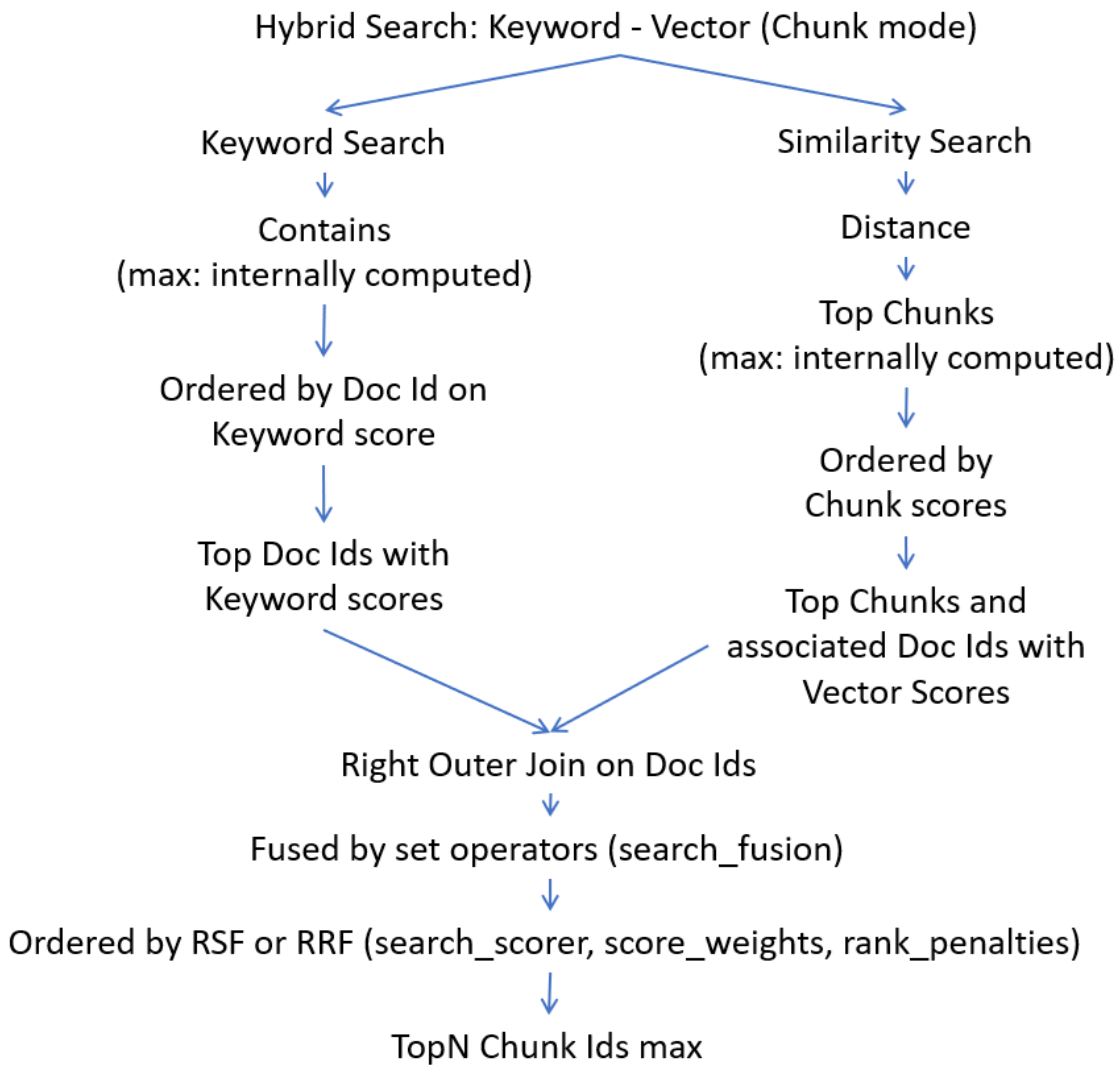
In chunk mode, the result of your search is a list of best chunk identifiers from the files stored using your base table.

To get to this list, two searches are conducted:

- **Keyword search**: During this search, the system uses the `CONTAINS` string representing the Oracle Text `CONTAINS` search expression for the searched keywords. The result of this operation is a list of document identifiers satisfying your `CONTAINS` expression. The numbers of document identifiers to retrieve, at max, is internally calculated.
- **Similarity search**: During this search, the system performs a similarity search with the query vector (created from the `SEARCH_TEXT` string) against the vector index of all the chunks of all your documents. The maximum number of chunks to retrieve is also internally calculated. It then assigns a vector score to each chunk retrieved. Because you want to do this search in `CHUNK SEARCH_MODE`, the result of this similarity search is ordered by chunk vector scores. The result of this similarity search is a list of chunk identifiers and associated document identifiers satisfying your `SEARCH_TEXT` similarity query string.

After the searches complete, the system needs to merge the results and score them as illustrated here:

Figure 8-4 Scoring for Keyword and Semantic Search in Chunk Mode

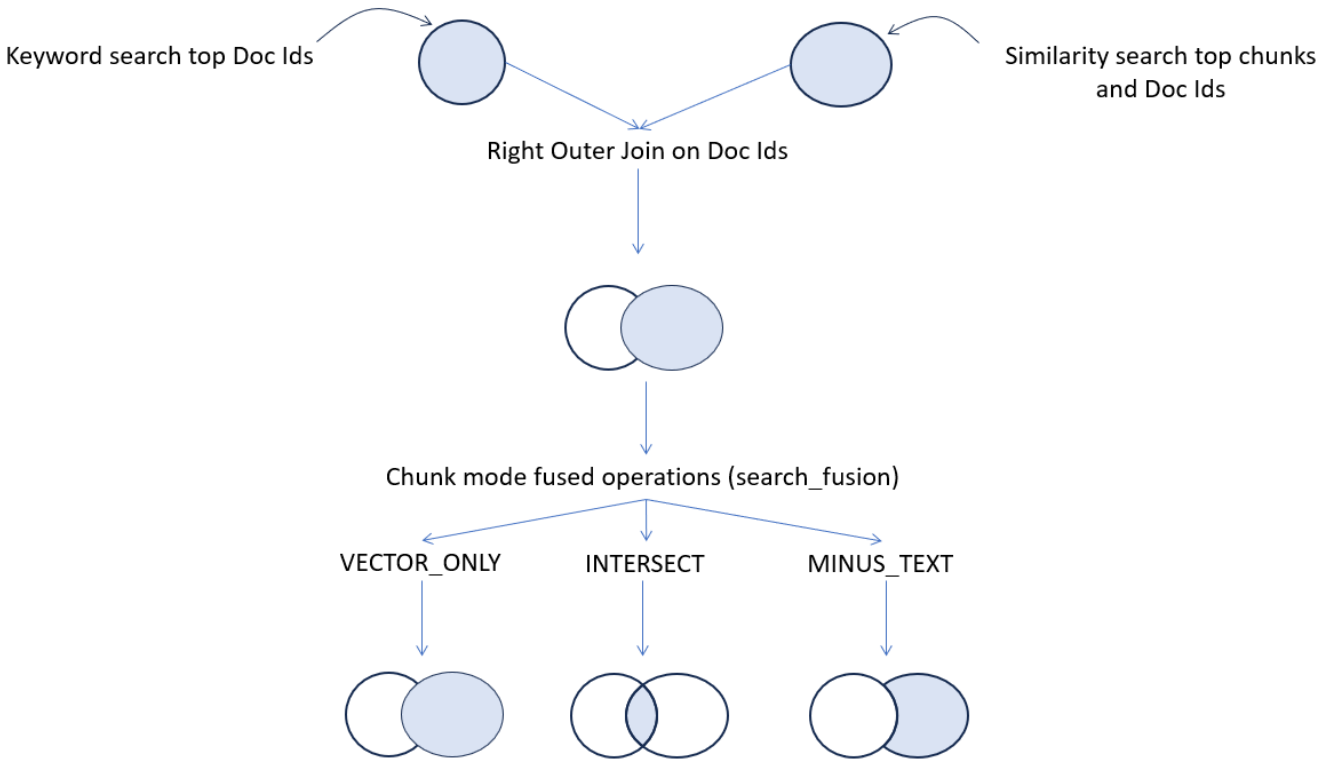


As outlined in the preceding diagram,

- First, both search results are submitted to a `RIGHT OUTER JOIN` operation on the document identifiers.
- Before final scoring, you have the possibility to define what you want to retain from this intermediate result set by specifying the `SEARCH_FUSION` operation.
- Then comes the time where final scoring is computed using the defined `SEARCH_SCORER` algorithm such as Reciprocal Rank Fusion (RRF) or Relative Score Fusion (RSF). The final scoring can use the specified `SCORE_WEIGHT` and `RANK_PENALTY` values for each retrieved document identifier, depending from which sort operation they are coming from.
- Finally, the defined `topN` chunk identifiers are returned at most.

The possible fuse operators are illustrated here:

Figure 8-5 Fusion Operation in Chunk Search Mode



`MINUS_VECTOR`, `UNION`, and `TEXT_ONLY` are ignored. `MINUS_VECTOR` would eliminate all results. `TEXT_ONLY` and `UNION` are not possible because the right outer join excludes the non-overlapping text results.

Related Topics

- Understand Hybrid Vector Indexes
  A hybrid vector index inherits all the information retrieval capabilities of Oracle Text search indexes and leverages the semantic search capabilities of Oracle AI Vector Search vector indexes.
- SEARCH
  Use the `DBMS_HYBRID_VECTOR.SEARCH` PL/SQL function to run textual queries, vector similarity queries, or hybrid queries against hybrid vector indexes.

Query Hybrid Vector Indexes End-to-End Example

In this example, you can see an end-to-end hybrid search workflow. First, you run the `CREATE HYBRID VECTOR INDEX` SQL statement that prepares, chunks, embeds, stores, and indexes your input data. You then perform vector search alongside keyword search using the `DBMS_HYBRID_VECTOR.SEARCH` PL/SQL query API.

Here, you can see how to use hybrid search in an HR Recruitment scenario, where you want to hire employees with specific technical skills (keyword search for "C, Python, and Database") but who also display certain personality and leadership traits (semantic search for "prioritize teamwork and leadership experience"). You can also see how to perform different kinds of pure and non-pure hybrid search queries in multiple search modes.

1. Connect to Oracle Database as a local user.

a. Log in to SQL*Plus as the `SYS` user, connecting as `SYSDBA`:

```
conn sys/password as sysdba


CREATE TABLESPACE tbs1
DATAFILE 'tbs5.dbf' SIZE 20G AUTOEXTEND ON
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;


SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 10000
SET LONG 10000
```

b. Create a local user (`docuser`) and grant necessary privileges:

```
DROP USER docuser cascade;


CREATE USER docuser identified by docuser DEFAULT TABLESPACE tbs1 quota
unlimited on tbs1;


GRANT DB_DEVELOPER_ROLE, create credential to docuser;
```

c. Create a local directory on your server (`DEMO_DIR`) to store your input data and model files. Grant necessary privileges:

```
create or replace directory DEMO_DIR as '/my_local_dir/';


grant read, write on directory DEMO_DIR to docuser;

commit;
```

d. Connect as the local user (`docuser`):

```
CONN docuser/password
```

2. Load an ONNX format embedding model into Oracle Database by calling the `load_onnx_model` procedure.

This embedding model is internally used to generate vector embeddings from your input data.

```
EXECUTE dbms_vector.drop_onnx_model(model_name => 'doc_model', force =>
true);


EXECUTE dbms_vector.load_onnx_model(
  'DEMO_DIR', 'my_embedding_model.onnx', 'doc_model',
   JSON('{"function" : "embedding", "embeddingOutput" : "embedding" ,
"input": {"input": ["DATA"]}}'));
```

In this example, the procedure loads an ONNX model file, named `my_embedding_model.onnx` from the `DEMO_DIR` directory, into the database as `doc_model`. You must replace `my_embedding_model.onnx` with an ONNX export of your embedding model and `doc_model` with the name under which the imported model is stored in the database.

> **Note:**
>
> If you do not have an embedding model in ONNX format, then perform the steps listed in ONNX Pipeline Models : Text Embedding.

3. Create a relational table (for example, `doc_tab`) and insert all your documents in a textual column (for example, `text`).

```
DROP TABLE doc_tab purge;


CREATE TABLE doc_tab (id number, text varchar2(500));

insert into t1 values(1, 'Candidate-1: C Master. Optimizes low-level
system (i.e. Database) performance with C. Strong leadership skills in
guiding teams to deliver complex projects.');
insert into t1 values(2, 'Candidate-2: Full-Stack Developer. Skilled in
Database, C, HTML, JavaScript, and Python with experience in building
responsive web applications. Thrives in collaborative team environments.');
insert into t1 values(3, 'Candidate-3: DevOps Engineer. Manages CI/CD
pipelines (Jenkins, Gitlab) with expertise in cloud infrastructure (OCI,
AWS, GCP). Proven track record of streamlining deployments and ensuring
high availability.');
insert into t1 values(4, 'Candidate-4: Database Administrator (DBA).
Maintains and secures enterprise database (Oracle, MySql, SQL Server).
Passionate about data integrity and optimization. Strong mentor for junior
DBA(s).');
insert into t1 values(5, 'Candidate-5: C, Java, Python, and Database (DBA)
Guru. Develops scalable applications. Strong leadership, teamwork and
collaborative.');

commit;
```

4. Create a hybrid vector index (`my_hybrid_idx`) on the `text` column of the `doc_tab` table.

```
CREATE HYBRID VECTOR INDEX my_hybrid_idx on
  doc_tab(text)
  parameters('model doc_model');
```

Here, `model` specifies the embedding model in ONNX format that you have imported into the database for generating embeddings. This is the only required indexing parameter to create a hybrid vector index.

An index named `my_hybrid_idx` is created on the `text` column of the `doc_tab` table. The default vector index type is set to Inverted File Flat (IVF).

5. Query the hybrid vector index.

In the next steps, we will explore all possible ways in which you can query the hybrid vector index using multiple search modes described in Understand Hybrid Search. You can then compare the results to examine the differences in scores and ranking for a comprehensive understanding.

a. **Simple default query**:

By default, hybrid search offers a simplified query with predefined fields. The minimum input parameters required are `hybrid_index_name` and `search_text`.

The same search text (`C, Python, Database`) is used to query the vectorized chunk index and the text document index. This search text is transformed into a `CONTAINS` query for keyword search, and is vectorized for a `VECTOR_DISTANCE` query for semantic search.

```
select json_Serialize(
  dbms_hybrid_vector.search(
    json(
         '{
           "hybrid_index_name" : "my_hybrid_idx",
           "search_text"       : "C, Python, Database"
          }'
        )
  ) pretty)
from dual;
```

This query returns the top 3 rows, ordered by score relevance. The highest final score (`69.54`) of Candidate-2 indicates the best match. All the return attributes are shown by default in this query.

```
[
  {
    "rowid"        : "AAAR9jAABAAAQeaAAB",
    "score"        : 69.54,
    "vector_score" : 69.69,
    "text_score"   : 68,
    "vector_rank"  : 1,
    "text_rank"    : 1,
    "chunk_text"   : "Candidate-2: Full-Stack Developer. Skilled in
Database, C, HTM
L, JavaScript, and Python with experience in building responsive web
application
```

```
s. Thrives in collaborative team environments.",
    "chunk_id"     : "1"
  },
  {
    "rowid"        : "AAAR9jAABAAAQeaAAA",
    "score"        : 62.77,
    "vector_score" : 65.55,
    "text_score"   : 35,
    "vector_rank"  : 4,
    "text_rank"    : 2,
    "chunk_text"   : "Candidate-1: C Master. Optimizes low-level system
(i.e. Da
tabase) performance with C. Strong leadership skills in guiding teams
to deliver
 complex projects.",
    "chunk_id"     : "1"
  },
  {
    "rowid"        : "AAAR9jAABAAAQeaAAD",
    "score" : 62.15,
    "vector_score" : 68.17,
    "text_score"   : 2,
    "vector_rank"  : 2,
    "text_rank"    : 3,
    "chunk_text"   : "Candidate-4: Database Administrator (DBA).
Maintains and
secures enterprise database (Oracle, MySql, SQL Server). Passionate
about data i
ntegrity and optimization. Strong mentor for junior DBA(s).",
    "chunk_id"     : "1"
  }
]
```

b. **Pure semantic in document mode (text as query input)**:

This is a pure vector-based similarity query to fetch document-level search results. Here, the `search_text` string is vectorized into a query vector for a `VECTOR_DISTANCE` semantic query.

A vector search operates at the chunk level, where the query first extracts the top candidates of vectorized chunks, aggregates them by document using the `aggregator` (`MAX`) function, produces a combined score (according to the `aggregator`), and finally returns the top n documents. To know more about how this works, see Pure Semantic in Document Mode.

```
select json_Serialize(
  dbms_hybrid_vector.search(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "vector":
            {
             "search_text"   : "prioritize teamwork and leadership
experience",
             "search_mode"   : "DOCUMENT",
             "aggregator"    : "MAX"
            },
         "return":
```

```
            {
             "values"        : [ "rowid", "score" ],
             "topN": 3
            }
      }'
    )
  ) pretty)
from dual;
```

The top 3 documents are returned with the corresponding `ROWID`s of the `doc_tab` table rows as well as their vector scores.

```
[
  {
    "rowid" : "AAAR9jAABAAAQeaAAA",
    "score" : 61
  },
  {
    "rowid" : "AAAR9jAABAAAQeaAAD",
    "score" : 56.64
  },
  {
    "rowid" : "AAAR9jAABAAAQeaAAB",
    "score" : 55.75
  }
]
```

c. **Pure semantic in document mode (embedding as query input)**:

As compared to the previous pure semantic query in document mode (where you used the `search_text` parameter to specify the query text for vector search), here you use the `search_vector` parameter to directly pass a vector embedding (query vector) as input for your vector search.

In the same manner, this query retrieves the top vectorized chunk results, aggregates them by document using the `aggregator` (`MAX`) function, and finally returns the top n documents.

```
select json_Serialize(
  dbms_hybrid_vector.search(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "vector":
          {
           "search_vector"   : vector_serialize(
                                          vector_embedding(doc_model
                                                using
                                                "C, Python,
Database"
                                                as data)
                                         RETURNING CLOB),
           "search_mode"     : "DOCUMENT",
           "aggregator"      : "MAX"
          },
         "return":
          {
```

```
           "values"     : [ "rowid", "score" ],
           "topN": 3
          }
      }'
    )
  ) pretty)
from dual;
```

The top 3 documents are returned with the corresponding `ROWID`s of the `doc_tab` table rows as well as their vector scores.

```
[
  {
    "rowid" : "AAAR9jAABAAAQeaAAB",
    "score" : 69.69
  },
  {
    "rowid" : "AAAR9jAABAAAQeaAAD",
    "score" : 68.17
  },
  {
    "rowid" : "AAAR9jAABAAAQeaAAC",
    "score" : 67.48
  }
]
```

d. **Pure semantic in chunk mode**:

This is a pure vector-based similarity query to retrieve chunk results. Note that this is the Oracle AI Vector Search semantic search equivalent. Here, vector search retrieves the best vectorized chunks from the same or different set of documents and then internally computes their vector scores. The top n chunks with the highest vector scores are returned. To know more about how this works, see Pure Semantic in Chunk Mode.

As compared to the previous pure semantic search in document mode, aggregation of chunk results is not performed in chunk mode, so the `aggregator` function is not needed.

```
select json_Serialize(
  dbms_hybrid_vector.search(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "vector":
          {
             "search_text"   : "prioritize teamwork and leadership
experience",
             "search_mode"   : "CHUNK"
          },
         "return":
          {
             "values"        : [ "rowid", "score", "chunk_text",
"chunk_id" ],
             "topN"          : 3
          }
      }'
```

```
    )
  ) pretty)
from dual;
```

The top 3 chunks with the highest scores are returned along with chunk IDs and chunk texts. You can also see the corresponding `ROWID`s of the `doc_tab` table rows. The highest score (`61`) of Candidate-1 with chunk ID 1 indicates the top match:

```
[
  {
    "rowid"      : "AAAR9jAABAAAQeaAAA",
    "score"      : 61,
    "chunk_text" : "Candidate-1: C Master. Optimizes low-level system
(i.e. Da
tabase) performance with C. Strong leadership skills in guiding teams
to deliver
 complex projects.",
    "chunk_id"   : "1"
  },
  {
    "rowid"      : "AAAR9jAABAAAQeaAAD",
    "score"      : 56.64,
    "chunk_text" : "Candidate-4: Database Administrator (DBA).
Maintains and
secures enterprise database (Oracle, MySql, SQL Server). Passionate
about data i
ntegrity and optimization. Strong mentor for junior DBA(s).",
    "chunk_id"   : "1"
  },
  {
    "rowid"      : "AAAR9jAABAAAQeaAAB",
    "score"      : 55.75,
    "chunk_text" : "Candidate-2: Full-Stack Developer. Skilled in
Database, C, HTM
L, JavaScript, and Python with experience in building responsive web
application
s. Thrives in collaborative team environments.",
    "chunk_id"   : "1"
  }
]
```

e. **Pure keyword in document mode**:

This is a pure text-based keyword query to fetch document-level results. Note that this is equivalent to the traditional `CONTAINS` query using Oracle Text. Here, full-text search retrieves the best documents and then internally computes their keyword scores. The top n documents with the highest keyword scores are returned. To know more about how this works, see Pure Keyword in Document Mode.

```
select json_Serialize(
  dbms_hybrid_vector.search(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "text":
          {
           "contains"        : "C, Python, Database"
```

```
          },
         "return":
          {
           "values"          : [ "rowid", "score" ],
           "topN": 3
          }
      }'
    )
  ) pretty)
from dual;
```

The top 3 documents with the highest scores are returned, where you can see the corresponding `ROWID`s of the `doc_tab` table as well as their keyword scores:

```
[
  {
    "rowid" : "AAAR9jAABAAAQeaAAB",
    "score" : 68
  },
  {
    "rowid" : "AAAR9jAABAAAQeaAAA",
    "score" : 35
  },
  {
    "rowid" : "AAAR9jAABAAAQeaAAD",
    "score" : 2
  }
]
```

f. **Keyword and semantic in document mode (common query string)**:

This is a hybrid query that conducts both keyword search and vector search on the data, and then combines the keyword scores and semantic scores to fetch document-level results.

Here, the same text string (`C, Python, Database`) is used for both:

- a keyword query on the document text index by converting the `SEARCH_TEXT` string into a `CONTAINS ACCUM` operator syntax
- and a semantic query on the vectorized chunk index by vectorizing the `SEARCH_TEXT` string for a `VECTOR_DISTANCE` query

```
select json_Serialize(
  dbms_hybrid_vector.search(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "search_text"       : "C, Python, Database",
         "search_fusion"     : "INTERSECT",
         "search_scorer"     : "rsf",
         "vector":
            {
             "search_mode"   : "DOCUMENT",
             "aggregator"    : "MAX"
            },
         "return":
            {
```

```
            "values"        : [ "rowid", "score" ],
            "topN"          : 3
           }
      }'
    )
  ) pretty)
from dual;
```

For vector search, this query searches the query vector (created from the `SEARCH_TEXT` string) against the vector index. A vector search operates at the chunk level, where the query first extracts the top candidates of vectorized chunks, aggregates them by document using the `aggregator` (`MAX`) function, produces a combined vector score (according to the `aggregator`), and finally returns the top n documents.

For scoring, first both search results are added using a `UNION ALL` operation. Then the results are fused using the `SEARCH_FUSION` operator, `INTERSECT` that combines all distinct rows selected by both the searches. The final scoring is computed using the defined `SEARCH_SCORER` algorithm, RSF. Finally, the defined `topN` doc IDs are returned at most.

To know more about how the scores are calculated, see Keyword and Semantic in Document Mode.

The top 3 documents with the highest scores are returned, where you can see a list of `ROWID`s from your base table (`doc_tab`) corresponding to the list of best files identified.

```
[
  {
    "rowid" : "AAAR9jAABAAAQeaAAB",
    "score" : 69.54
  },
  {
    "rowid" : "AAAR9jAABAAAQeaAAA",
    "score" : 62.77
  },
  {
    "rowid" : "AAAR9jAABAAAQeaAAD",
    "score" : 62.15
  }
]
```

g. **Keyword and semantic in document mode (separate query strings)**:

As compared to the previous keyword and semantic search in document mode (where you used the same `search_text` string for both text search and vector search), here you specify two separate search texts for both the search types.

```
select json_Serialize(
  dbms_hybrid_vector.search(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "search_fusion"     : "INTERSECT",
         "search_scorer"     : "rsf",
         "vector":
          {
             "search_text"   : "prioritize teamwork and leadership
experience",
```

```
            "search_mode"    : "DOCUMENT",
            "aggregator"     : "MAX"
          },
         "text":
          {
            "contains"       : "C, Python, Database"
          },
         "return":
          {
            "values"         : [ "rowid", "score" ],
            "topN"           : 3
          }
      }'
    )
  ) pretty)
from dual;
```

In the same manner, the keyword search retrieves a list of doc IDs satisfying your `CONTAINS` text query string (`C, Python, Database`). The vector search retrieves a list of doc IDs satisfying your `SEARCH_TEXT` similarity query string (`prioritize teamwork and leadership experience`).

For scoring, first both search results are added using a `UNION ALL` operation. Then the results are fused using the `SEARCH_FUSION` operator, `INTERSECT`. The final scoring is computed using the defined `SEARCH_SCORER` algorithm, RSF. Finally, the defined `topN` doc IDs are returned at most.

To know more about how the scores are calculated, see Keyword and Semantic in Document Mode.

The top 3 documents with the highest scores are returned, where you can see a list of `ROWID`s from your base table (`doc_tab`) corresponding to the list of best files identified.

```
[
  {
    "rowid" : "AAAR9jAABAAAQeaAAA",
    "score" : 58.64
  },
  {
    "rowid" : "AAAR9jAABAAAQeaAAB",
    "score" : 56.86
  },
  {
    "rowid" : "AAAR9jAABAAAQeaAAD",
    "score" : 51.67
  }
]
```

h. **Keyword and semantic in chunk mode**:

This is a hybrid query that conducts both keyword search and vector search on the data, and then combines the keyword scores and semantic scores to fetch chunk-level results.

```
select json_Serialize(
  dbms_hybrid_vector.search(
    json(
```

```
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "vector":
          {
             "search_text"   : "prioritize teamwork and leadership
experience",
             "search_mode"   : "CHUNK"
          },
         "text":
          {
             "contains"      : "C, Python, Database"
          },
         "return":
          {
             "values"        : [ "rowid", "score", "chunk_text",
"chunk_id" ],
             "topN": 3
          }
      }'
    )
  ) pretty)
from dual;
```

The keyword search retrieves a list of doc IDs satisfying your `CONTAINS` text query string (`C, Python, Database`).

The vector search performs a similarity query with the query vector (created from the `SEARCH_TEXT` string: `prioritize teamwork and leadership experience`) against the vector index of all the chunks of all your documents. It retrieves a list of chunk IDs and associated doc IDs satisfying your `SEARCH_TEXT` similarity query string.

Aggregation of chunk results is not performed in chunk mode, so the `AGGREGATOR` function is not applicable.

For scoring, first the text document results are added into the chunk results using the `RIGHT OUTER JOIN` operation on doc IDs. Any text document results that are not in the vector candidate chunks are not returned. Then the results are fused using the `SEARCH_FUSION` operator, `INTERSECT` that combines all distinct rows selected by both the searches. The final scoring is computed using the defined `SEARCH_SCORER` algorithm, RSF. Finally, the defined `topN` chunk IDs are returned at most.

To know how the scores are calculated, see Keyword and Semantic in Chunk Mode.

A list of top 3 chunk IDs are returned from the files stored in your base table (`doc_tab`), along with chunk texts and the corresponding `ROWID`s. The highest final score (`58.64`) of Candidate-1 with chunk ID 1 indicates the top match:

```
[
  {
    "rowid"      : "AAAR9jAABAAAQeaAAA",
    "score"      : 58.64,
    "chunk_text" : "Candidate-1: C Master. Optimizes low-level system
(i.e. Da
tabase) performance with C. Strong leadership skills in guiding teams
to deliver
 complex projects.",
    "chunk_id"   : "1"
  },
```

```
  {
    "rowid"      : "AAAR9jAABAAAQeaAAB",
    "score"      : 56.86,
    "chunk_text" : "Candidate-2: Full-Stack Developer. Skilled in
Database, C, HTM
L, JavaScript, and Python with experience in building responsive web
application
s. Thrives in collaborative team environments.",
    "chunk_id"   : "1"
  },
  {
    "rowid"      : "AAAR9jAABAAAQeaAAD",
    "score"      : 51.67,
    "chunk_text" : "Candidate-4: Database Administrator (DBA).
Maintains and
secures enterprise database (Oracle, MySql, SQL Server). Passionate
about data i
ntegrity and optimization. Strong mentor for junior DBA(s).",
    "chunk_id"   : "1"
  }
]
```

Related Topics

- Manage Hybrid Vector Indexes
  Learn how to manage a hybrid vector index, which is a single index for searching by *similarity* and *keywords*, to enhance the accuracy of your search results.
- SEARCH
  Use the `DBMS_HYBRID_VECTOR.SEARCH` PL/SQL function to run textual queries, vector similarity queries, or hybrid queries against hybrid vector indexes.

9

Work with LLM-Powered APIs and Retrieval Augmented Generation

You can use Vector Utility PL/SQL APIs for prompting Large Language Models (LLMs) with textual prompts and images, using LLM-powered interfaces. You can also communicate with LLMs through the implementation of Retrieval Augmented Generation (RAG), which helps to generate more accurate and informative responses.

- Use LLM-Powered APIs to Generate Summary and Text
  Run these end-to-end examples to see how you can summarize or describe textual inputs and images.
- Use Retrieval Augmented Generation to Complement LLMs
  RAG lets you mitigate the inaccuracies and hallucinations faced when using LLMs. Oracle AI Vector Search enables RAG within Oracle Database using the `DBMS_VECTOR_CHAIN` PL/SQL package or through popular frameworks (such as LangChain).
- Supported Third-Party Provider Operations and Endpoints
  Review a list of third-party REST providers and REST endpoints that are supported for various vector generation, summarization, text generation, and reranking operations.

Use LLM-Powered APIs to Generate Summary and Text

Run these end-to-end examples to see how you can summarize or describe textual inputs and images.

- Generate Summary
  In these examples, you can see how to summarize a given textual extract.
- Generate Text Response
  In these examples, you can see how to generate a textual answer, description, or summary based on the specified task in a given prompt.
- Describe Image Content
  In these examples, you can see how to generate a textual analysis or description of the contents of a given image.

Generate Summary

In these examples, you can see how to summarize a given textual extract.

A summary is a short and concise extract with key features of a document that best represents what the document is about as a whole. A summary can be free-form paragraphs or bullet points based on the format that you specify.

> **Note:**
>
> You can also generate a summary using Oracle Database as the service provider. In this case, Oracle Text is internally used to generate a summary. However, that is outside the scope of these examples. See UTL_TO_SUMMARY.

- Generate Summary Using Public REST Providers
  Perform a text-to-summary transformation, using publicly hosted third-party text summarization models by Cohere, Generative AI, Google AI, Hugging Face, OpenAI, or Vertex AI.
- Generate Summary Using the Local REST Provider Ollama
  Perform a text-to-summary transformation by accessing open LLMs, using the local host REST endpoint provider Ollama.

Generate Summary Using Public REST Providers

Perform a text-to-summary transformation, using publicly hosted third-party text summarization models by Cohere, Generative AI, Google AI, Hugging Face, OpenAI, or Vertex AI.

Here, you call the chainable utility function `UTL_TO_SUMMARY`.

> **WARNING:**
>
> Certain features of the database may allow you to access services offered separately by third-parties, for example, through the use of JSON specifications that facilitate your access to REST APIs.
>
> Your use of these features is solely at your own risk, and you are solely responsible for complying with any terms and conditions related to use of any such third-party services. Notwithstanding any other terms and conditions related to the third-party services, your use of such database features constitutes your acceptance of that risk and express exclusion of Oracle's responsibility or liability for any damages resulting from such access.

To summarize a textual extract, using an external LLM:

1. Connect to Oracle Database as a local user.
   a. Log in to SQL*Plus as the `SYS` user, connecting as `SYSDBA`:

```
conn sys/password as sysdba


CREATE TABLESPACE tbs1
DATAFILE 'tbs5.dbf' SIZE 20G AUTOEXTEND ON
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;


SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
```

```
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 10000
SET LONG 10000
```

b. Create a local user (`docuser`) and grant necessary privileges:

```
DROP USER docuser cascade;


CREATE USER docuser identified by docuser DEFAULT TABLESPACE tbs1 quota
unlimited on tbs1;


GRANT DB_DEVELOPER_ROLE, create credential to docuser;
```

c. Connect as the local user (`docuser`):

```
CONN docuser/password
```

2. Set proxy if one exists.

```
EXEC UTL_HTTP.SET_PROXY('<proxy-hostname>:<proxy-port>');
```

3. Grant connect privilege to `docuser` for allowing connection to the host, using the `DBMS_NETWORK_ACL_ADMIN` procedure.

   This example uses `*` to allow any host. However, you can explicitly specify the host that you want to connect to.

```
BEGIN
  DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE(
    host => '*',
    ace => xs$ace_type(privilege_list => xs$name_list('connect'),
                       principal_name => 'docuser',
                       principal_type => xs_acl.ptype_db));
END;
/
```

4. Set up your credentials for the REST provider that you want to access and then call `UTL_TO_SUMMARY`.

   - **Using Generative AI**:

     > **Note:**
     >
     > Currently, `UTL_TO_SUMMARY` does not work for Generative AI because the model and summary endpoint supported for Generative AI have been retired. It will be available in a subsequent release.

     a. Run `DBMS_VECTOR_CHAIN.CREATE_CREDENTIAL` to create and store an OCI credential (`OCI_CRED`).

Generative AI requires the following authentication parameters:

```
{
"user_ocid"       : "<user ocid>",
"tenancy_ocid"    : "<tenancy ocid>",
"compartment_ocid": "<compartment ocid>",
"private_key"     : "<private key>",
"fingerprint"     : "<fingerprint>"
}
```

You will later refer to this credential name when declaring JSON parameters for the `UTL_TO_SUMMARY` call.

> **Note:**
>
> The generated private key may appear as:
>
> ```
> -----BEGIN RSA PRIVATE KEY-----
> <private key string>
> -----END RSA PRIVATE KEY-----
> ```
>
> You pass the *<private key string>* value (excluding the `BEGIN` and `END` lines), either as a single line or as multiple lines.

```
exec dbms_vector_chain.drop_credential('OCI_CRED');


declare
  jo json_object_t;
begin
  jo := json_object_t();
  jo.put('user_ocid','<user ocid>');
  jo.put('tenancy_ocid','<tenancy ocid>');
  jo.put('compartment_ocid','<compartment ocid>');
  jo.put('private_key','<private key>');
  jo.put('fingerprint','<fingerprint>');
  dbms_vector_chain.create_credential(
    credential_name   => 'OCI_CRED',
    params            => json(jo.to_string));
end;
/
```

Replace all the authentication parameter values.

For example:

```
declare
  jo json_object_t;
begin
  jo := json_object_t();
```

```
jo.put('user_ocid','ocid1.user.oc1..aabbalbbaa1112233aabbaabb1111222
aa1111bb');

jo.put('tenancy_ocid','ocid1.tenancy.oc1..aaaaalbbbb1112233aaaabbaa1
111222aaa111a');

jo.put('compartment_ocid','ocid1.compartment.oc1..ababalabab1112233a
babababb1111222aba11ab');
  jo.put('private_key','AAAaaaBBB11112222333...AAA111AAABBB222aaa1a/
+');

jo.put('fingerprint','01:1a:a1:aa:12:a1:12:1a:ab:12:01:ab:a1:12:ab:1
a');
  dbms_vector_chain.create_credential(
    credential_name   => 'OCI_CRED',
    params            => json(jo.to_string));
end;
/
```

b. Run `DBMS_VECTOR_CHAIN.UTL_TO_SUMMARY`:

Here, the cohere.command-r-16k model is used. You can replace the `model` value, as required.

> **Note:**
>
> For a list of all REST endpoint URLs and models that are supported to use with Generative AI, see Supported Third-Party Provider Operations and Endpoints.

```
-- select example

var params clob;
exec :params := '
{
  "provider"       : "ocigenai",
  "credential_name": "OCI_CRED",
  "url"            : "https://inference.generativeai.us-
chicago-1.oci.oraclecloud.com/20231130/actions/chat",
  "model"          : "cohere.command-r-16k"
}';

select dbms_vector_chain.utl_to_summary(
  'A transaction is a logical, atomic unit of work that contains
one or more SQL
    statements.
    An RDBMS must be able to group SQL statements so that they are
either all
    committed, which means they are applied to the database, or all
rolled back, which
    means they are undone.
    An illustration of the need for transactions is a funds
transfer from a savings account to
    a checking account. The transfer consists of the following
```

```
separate operations:
    1. Decrease the savings account.
    2. Increase the checking account.
    3. Record the transaction in the transaction journal.
    Oracle Database guarantees that all three operations succeed or
fail as a unit. For
    example, if a hardware failure prevents a statement in the
transaction from executing,
    then the other statements must be rolled back.
    Transactions set Oracle Database apart from a file system. If
you
    perform an atomic operation that updates several files, and if
the system fails halfway
    through, then the files will not be consistent. In contrast, a
transaction moves an
    Oracle database from one consistent state to another. The basic
principle of a
    transaction is all or nothing: an atomic operation succeeds or
fails as a whole.',
  json(:params)) from dual;

-- PL/SQL example

declare
  input clob;
  params clob;
  output clob;
begin
  input := 'A transaction is a logical, atomic unit of work that
contains one or more SQL
    statements.
    An RDBMS must be able to group SQL statements so that they are
either all
    committed, which means they are applied to the database, or all
rolled back, which
    means they are undone.
    An illustration of the need for transactions is a funds
transfer from a savings account to
    a checking account. The transfer consists of the following
separate operations:
    1. Decrease the savings account.
    2. Increase the checking account.
    3. Record the transaction in the transaction journal.
    Oracle Database guarantees that all three operations succeed or
fail as a unit. For
    example, if a hardware failure prevents a statement in the
transaction from executing,
    then the other statements must be rolled back.
    Transactions set Oracle Database apart from a file system. If
you
    perform an atomic operation that updates several files, and if
the system fails halfway
    through, then the files will not be consistent. In contrast, a
transaction moves an
    Oracle database from one consistent state to another. The basic
principle of a
```

```
    transaction is all or nothing: an atomic operation succeeds or
fails as a whole.';

  params := '
{
  "provider"       : "ocigenai",
  "credential_name": "OCI_CRED",
  "url"            : "https://inference.generativeai.us-
chicago-1.oci.oraclecloud.com/20231130/actions/chat",
  "model"          : "cohere.command-r-16k"
}';

  output := dbms_vector_chain.utl_to_summary(input, json(params));
  dbms_output.put_line(output);
  if output is not null then
    dbms_lob.freetemporary(output);
  end if;
exception
  when OTHERS THEN
    DBMS_OUTPUT.PUT_LINE (SQLERRM);
    DBMS_OUTPUT.PUT_LINE (SQLCODE);
end;
/
```

Optionally, you can specify additional REST provider-specific parameters. For example:

```
{
  "provider"       : "ocigenai",
  "credential_name": "OCI_CRED",
  "url"            : "https://inference.generativeai.us-
chicago-1.oci.oraclecloud.com/20231130/actions/chat",
  "model"          : "cohere.command-r-16k",
  "length"         : "MEDIUM",
  "format"         : "PARAGRAPH",
  "temperature"    : 1.0
}
```

- **Using Cohere, Google AI, Hugging Face, OpenAI, and Vertex AI**:

  a. Run `DBMS_VECTOR_CHAIN.CREATE_CREDENTIAL` to create and store a credential.

  Cohere, Google AI, Hugging Face, OpenAI, and Vertex AI require the following authentication parameter:

  ```
  { "access_token": "<access token>" }
  ```

  You will later refer to this credential name when declaring JSON parameters for the `UTL_TO_SUMMARY` call.

  ```
  exec dbms_vector_chain.drop_credential('<credential name>');


  declare
    jo json_object_t;
  begin
    jo := json_object_t();
  ```

```
  jo.put('access_token', '<access token>');
  dbms_vector_chain.create_credential(
    credential_name   => '<credential name>',
    params            => json(jo.to_string));
end;
/
```

Replace the `access_token` and `credential_name` values. For example:

```
declare
  jo json_object_t;
begin
  jo := json_object_t();
  jo.put('access_token', 'AbabA1B123aBc123AbabAb123a1a2ab');
  dbms_vector_chain.create_credential(
    credential_name   => 'HF_CRED',
    params            => json(jo.to_string));
end;
/
```

b. Run `DBMS_VECTOR_CHAIN.UTL_TO_SUMMARY`:

```
-- select example

var params clob;
exec :params := '
{
  "provider": "<REST provider>",
  "credential_name": "<credential name>",
  "url": "<REST endpoint URL for text summarization service>",
  "model": "<REST provider text summarization model name>"
}';

select dbms_vector_chain.utl_to_summary(
  'A transaction is a logical, atomic unit of work that contains
one or more SQL
    statements.
    An RDBMS must be able to group SQL statements so that they are
either all
    committed, which means they are applied to the database, or all
rolled back, which
    means they are undone.
    An illustration of the need for transactions is a funds
transfer from a savings account to
    a checking account. The transfer consists of the following
separate operations:
    1. Decrease the savings account.
    2. Increase the checking account.
    3. Record the transaction in the transaction journal.
    Oracle Database guarantees that all three operations succeed or
fail as a unit. For
    example, if a hardware failure prevents a statement in the
transaction from executing,
    then the other statements must be rolled back.
    Transactions set Oracle Database apart from a file system. If
```

```
you
    perform an atomic operation that updates several files, and if
the system fails halfway
    through, then the files will not be consistent. In contrast, a
transaction moves an
    Oracle database from one consistent state to another. The basic
principle of a
    transaction is "all or nothing": an atomic operation succeeds
or fails as a whole.',
  json(:params)) from dual;

-- PL/SQL example

declare
  input clob;
  params clob;
  output clob;
begin
  input := 'A transaction is a logical, atomic unit of work that
contains one or more SQL
    statements.
    An RDBMS must be able to group SQL statements so that they are
either all
    committed, which means they are applied to the database, or all
rolled back, which
    means they are undone.
    An illustration of the need for transactions is a funds
transfer from a savings account to
    a checking account. The transfer consists of the following
separate operations:
    1. Decrease the savings account.
    2. Increase the checking account.
    3. Record the transaction in the transaction journal.
    Oracle Database guarantees that all three operations succeed or
fail as a unit. For
    example, if a hardware failure prevents a statement in the
transaction from executing,
    then the other statements must be rolled back.
    Transactions set Oracle Database apart from a file system. If
you
    perform an atomic operation that updates several files, and if
the system fails halfway
    through, then the files will not be consistent. In contrast, a
transaction moves an
    Oracle database from one consistent state to another. The basic
principle of a
    transaction is "all or nothing": an atomic operation succeeds
or fails as a whole.';

  params := '
{
  "provider": "<REST provider>",
  "credential_name": "<credential name>",
  "url": "<REST endpoint URL for text summarization service>",
  "model": "<REST provider text summarization model name>"
}';
```

```
  output := dbms_vector_chain.utl_to_summary(input, json(params));
  dbms_output.put_line(output);
  if output is not null then
    dbms_lob.freetemporary(output);
  end if;
exception
  when OTHERS THEN
    DBMS_OUTPUT.PUT_LINE (SQLERRM);
    DBMS_OUTPUT.PUT_LINE (SQLCODE);
end;
/
```

> **Note:**
>
> For a list of all supported REST endpoint URLs, see Supported Third-Party Provider Operations and Endpoints.

Replace `provider`, `credential_name`, `url`, and `model` with your own values. Optionally, you can specify additional REST provider parameters. This is shown in the following examples:

Cohere example:

```
{
  "provider"        : "cohere",
  "credential_name" : "COHERE_CRED",
  "url"             : "https://api.cohere.ai/v1/chat",
  "model"           : "command",
  "length"          : "medium",
  "format"          : "paragraph",
  "temperature"     : 1.0
}
```

Google AI example:

```
{
  "provider"         : "googleai",
  "credential_name"  : "GOOGLEAI_CRED",
  "url"              : "https://generativelanguage.googleapis.com/
v1beta/models/",
  "model"            : "gemini-pro:generateContent",
  "generation_config": {
                        "temperature"    : 0.9,
                        "topP"           : 1,
                        "candidateCount" : 1,
                        "maxOutputTokens": 256
                       }
}
```

Hugging Face example:

```
{
  "provider"         : "huggingface",
  "credential_name"  : "HF_CRED",
  "url"              : "https://api-inference.huggingface.co/
models/",
  "model"            : "facebook/bart-large-cnn"
}
```

OpenAI example:

```
{
  "provider"        : "openai",
  "credential_name" : "OPENAI_CRED",
  "url"             : "https://api.openai.com/v1/chat/completions",
  "model"           : "gpt-4o-mini",
  "max_tokens"      : 256,
  "temperature"     : 1.0
}
```

Vertex AI example:

```
{
  "provider"         : "vertexai",
  "credential_name"  : "VERTEXAI_CRED",
  "url"              : "https://LOCATION-
aiplatform.googleapis.com/v1/projects/PROJECT/locations/LOCATION/
publishers/google/models/",
  "model"            : "gemini-1.0-pro:generateContent",
  "generation_config": {
                        "temperature"    : 0.9,
                        "topP"           : 1,
                        "candidateCount" : 1,
                        "maxOutputTokens": 256
                       }
}
```

A generated summary may appear as:

```
A transaction is a logical unit of work that groups one or more SQL
statements
that must be executed as a unit, with all statements succeeding, or all
statements being rolled back. Transactions are a fundamental concept in
relational database management systems (RDBMS), and Oracle Database is
specifically designed to manage transactions, ensuring database
consistency and
integrity. Transactions differ from file systems in that they maintain
atomicity, ensuring that all related operations succeed or fail as a whole,
maintaining database consistency regardless of intermittent failures.
Transactions move a database from one consistent state to another, and the
fundamental principle is that a transaction is committed or rolled back as
a
whole, upholding the "all or nothing" principle.
```

```
PL/SQL procedure successfully completed.
```

This example uses the default settings for each provider. For detailed information on additional parameters, refer to your third-party provider's documentation.

Related Topics

- About Chainable Utility Functions and Common Use Cases
  These are intended to be a set of chainable and flexible "stages" through which you pass your input data to transform into a different representation, including vectors.
- UTL_TO_SUMMARY
  Use the `DBMS_VECTOR_CHAIN.UTL_TO_SUMMARY` chainable utility function to generate a summary for textual documents.

Generate Summary Using the Local REST Provider Ollama

Perform a text-to-summary transformation by accessing open LLMs, using the local host REST endpoint provider Ollama.

Ollama is a free and open-source command-line interface tool that allows you to run open LLMs (such as Llama 3, Phi 3, Mistral, or Gemma 2) locally and privately on your Linux, Windows, or macOS systems. You can access Ollama as a service using SQL and PL/SQL commands.

Here, you call the chainable utility function `UTL_TO_SUMMARY`.

> **WARNING:**
>
> Certain features of the database may allow you to access services offered separately by third-parties, for example, through the use of JSON specifications that facilitate your access to REST APIs.
>
> Your use of these features is solely at your own risk, and you are solely responsible for complying with any terms and conditions related to use of any such third-party services. Notwithstanding any other terms and conditions related to the third-party services, your use of such database features constitutes your acceptance of that risk and express exclusion of Oracle's responsibility or liability for any damages resulting from such access.

To generate a concise and informative summary of a textual extract, by calling a local LLM using Ollama:

1. Connect to Oracle Database as a local user.
   a. Log in to SQL*Plus as the `SYS` user, connecting as `SYSDBA`:

```
conn sys/password as sysdba


CREATE TABLESPACE tbs1
DATAFILE 'tbs5.dbf' SIZE 20G AUTOEXTEND ON
```

```
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;


SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 10000
SET LONG 10000
```

b. Create a local user (`docuser`) and grant necessary privileges:

```
DROP USER docuser cascade;


CREATE USER docuser identified by docuser DEFAULT TABLESPACE tbs1 quota
unlimited on tbs1;


GRANT DB_DEVELOPER_ROLE, create credential to docuser;
```

c. Connect as the local user (`docuser`):

```
CONN docuser/password
```

2. Install Ollama and run a model locally.

   a. Download and run the Ollama application from https://ollama.com/download.

   You can either install Ollama as a service that runs in the background or as a standalone binary with a manual install. For detailed installation-specific steps, see Quick Start in the Ollama Documentation.

   Note the following:

   - The Ollama server needs to be able to connect to the internet so that it can download the models. If you require a proxy server to access the internet, remember to set the appropriate environment variables before running the Ollama server. For example, to set for Linux:

     ```
     -- set a proxy if you require one

     export https_proxy=<proxy-hostname>:<proxy-port>
     export http_proxy=<proxy-hostname>:<proxy-port>
     export no_proxy=localhost,127.0.0.1,.example.com
     export ftp_proxy=<proxy-hostname>:<proxy-port>
     ```

   - If you are running Ollama and the database on different machines, then on the database machine, you must change the `URL` to refer to the host name or IP address that is running Ollama instead of the local host.
   - You may need to change your firewall settings on the machine that is running Ollama to allow the port through.

b. If running Ollama as a standalone binary from a manual install, then start the server:

```
ollama serve
```

c. Run a model using the `ollama run <model_name>` command.

For example, to call the Llama 3 model:

```
ollama run llama3
```

For detailed information on this step, see Ollama Readme.

d. Verify that Ollama is running locally by using a cURL command.

For example:

```
-- get summary

curl http://localhost:11434/api/generate -d '{
  "model" : "llama3",
  "prompt": "What is Oracle AI Vector Search?"
  "stream": false}'
```

3. Set proxy if one exists.

```
EXEC UTL_HTTP.SET_PROXY('<proxy-hostname>:<proxy-port>');
```

4. Grant connect privilege to `docuser` for allowing connection to the host, using the `DBMS_NETWORK_ACL_ADMIN` procedure.

This example uses `*` to allow any host. However, you can explicitly specify the host that you want to connect to.

```
BEGIN
  DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE(
    host => '*',
    ace => xs$ace_type(privilege_list => xs$name_list('connect'),
                       principal_name => 'docuser',
                       principal_type => xs_acl.ptype_db));
END;
/
```

5. Call `UTL_TO_SUMMARY`.

The Ollama service has a REST API endpoint for summarizing text. Specify the URL and other configuration parameters in a JSON object.

```
var gent_ollama_params clob;
exec :gent_ollama_params := '{
  "provider": "ollama",
  "host"    : "local",
  "url"     : "http://localhost:11434/api/generate",
  "model"   : "llama3"
}';

select dbms_vector_chain.utl_to_summary(
  'A transaction is a logical, atomic unit of work that contains one or
```

```
more SQL
    statements.
    An RDBMS must be able to group SQL statements so that they are either
all
    committed, which means they are applied to the database, or all rolled
back, which
    means they are undone.
    An illustration of the need for transactions is a funds transfer from
a savings account to
    a checking account. The transfer consists of the following separate
operations:
    1. Decrease the savings account.
    2. Increase the checking account.
    3. Record the transaction in the transaction journal.
    Oracle Database guarantees that all three operations succeed or fail
as a unit. For
    example, if a hardware failure prevents a statement in the transaction
from executing,
    then the other statements must be rolled back.
    Transactions set Oracle Database apart from a file system. If you
    perform an atomic operation that updates several files, and if the
system fails halfway
    through, then the files will not be consistent. In contrast, a
transaction moves an
    Oracle database from one consistent state to another. The basic
principle of a
    transaction is "all or nothing": an atomic operation succeeds or fails
as a whole.',
  json(:gent_ollama_params)) from dual;
```

You can replace the `url` and `model` with your own values, as required.

> **Note:**
>
> For a complete list of all supported REST endpoint URLs, see Supported Third-Party Provider Operations and Endpoints.

A generated summary may appear as:

```
A transaction in an RDBMS (Relational Database Management System) is a
self-contained unit of work that
consists of one or more SQL statements. This ensures that all changes made
by the transaction are either
committed (applied to the database) or rolled back (undone). Oracle
Database is specifically designed to
manage transactions, ensuring database consistency and integrity.
Transactions differ from file systems in that they maintain atomicity,
ensuring that all related operations
succeed or fail as a whole, maintaining database consistency regardless of
intermittent failures.
Transactions move a database from one consistent state to another, and the
fundamental principle is that a
```

```
transaction is committed or rolled back as a whole, upholding the "all or
nothing" principle.
```

Related Topics

- About Chainable Utility Functions and Common Use Cases
  These are intended to be a set of chainable and flexible "stages" through which you pass your input data to transform into a different representation, including vectors.
- UTL_TO_SUMMARY
  Use the `DBMS_VECTOR_CHAIN.UTL_TO_SUMMARY` chainable utility function to generate a summary for textual documents.

Generate Text Response

In these examples, you can see how to generate a textual answer, description, or summary based on the specified task in a given prompt.

A prompt can be an input text string, such as a question that you ask an LLM. For example, "`What is Oracle Text?`". A prompt can also be a set of instructions or a command, such as "`Summarize the following ...`", "`Draft an email asking for ...`", or "`Rewrite the following ...`", and can include results from a search.

- Generate Text Using Public REST Providers
  Perform a text-to-text transformation, using publicly hosted third-party text generation models by Cohere, Generative AI, Google AI, Hugging Face, OpenAI, or Vertex AI. The input is a textual prompt, and the generated output is a textual answer or description based on the specified task in that prompt.
- Generate Text Using the Local REST Provider Ollama
  Perform a text-to-text transformation by accessing open LLMs, using the local host REST endpoint provider Ollama. The input is a textual prompt, and the generated output is a textual answer or description based on the specified task in that prompt.

Generate Text Using Public REST Providers

Perform a text-to-text transformation, using publicly hosted third-party text generation models by Cohere, Generative AI, Google AI, Hugging Face, OpenAI, or Vertex AI. The input is a textual prompt, and the generated output is a textual answer or description based on the specified task in that prompt.

A prompt can be a text string (such as a question that you ask an LLM or a command), and can include results from a search.

Here, you can use the `UTL_TO_GENERATE_TEXT` function from either the `DBMS_VECTOR` or the `DBMS_VECTOR_CHAIN` package, depending on your use case.

> **WARNING:**
>
> Certain features of the database may allow you to access services offered separately by third-parties, for example, through the use of JSON specifications that facilitate your access to REST APIs.
>
> Your use of these features is solely at your own risk, and you are solely responsible for complying with any terms and conditions related to use of any such third-party services. Notwithstanding any other terms and conditions related to the third-party services, your use of such database features constitutes your acceptance of that risk and express exclusion of Oracle's responsibility or liability for any damages resulting from such access.

To generate a text response for the prompt "`What is Oracle Text?`", using an external LLM:

1. Connect to Oracle Database as a local user.
   a. Log in to SQL*Plus as the `SYS` user, connecting as `SYSDBA`:

```
conn sys/password as sysdba


CREATE TABLESPACE tbs1
DATAFILE 'tbs5.dbf' SIZE 20G AUTOEXTEND ON
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;


SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 10000
SET LONG 10000
```

   b. Create a local user (`docuser`) and grant necessary privileges:

```
DROP USER docuser cascade;


CREATE USER docuser identified by docuser DEFAULT TABLESPACE tbs1 quota
unlimited on tbs1;


GRANT DB_DEVELOPER_ROLE, create credential to docuser;
```

   c. Connect as the local user (`docuser`):

```
CONN docuser/password
```

2. Set proxy if one exists.

```
EXEC UTL_HTTP.SET_PROXY('<proxy-hostname>:<proxy-port>');
```

3. Grant connect privilege to `docuser` for allowing connection to the host, using the `DBMS_NETWORK_ACL_ADMIN` procedure.

   This example uses `*` to allow any host. However, you can explicitly specify the host that you want to connect to.

```
BEGIN
  DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE(
    host => '*',
    ace => xs$ace_type(privilege_list => xs$name_list('connect'),
                       principal_name => 'docuser',
                       principal_type => xs_acl.ptype_db));
END;
/
```

4. Set up your credentials for the REST provider that you want to access and then call `UTL_TO_GENERATE_TEXT`:

   - **Using Generative AI**:

     a. Call `CREATE_CREDENTIAL` to create and store an OCI credential (`OCI_CRED`).

        Generative AI requires the following authentication parameters:

```
{
"user_ocid"       : "<user ocid>",
"tenancy_ocid"    : "<tenancy ocid>",
"compartment_ocid": "<compartment ocid>",
"private_key"     : "<private key>",
"fingerprint"     : "<fingerprint>"
}
```

        You will later refer to this credential name when declaring JSON parameters for the `UTL_TO_GENERATE_TEXT` call.

> **Note:**
>
> The generated private key may appear as:
>
> ```
> -----BEGIN RSA PRIVATE KEY-----
> <private key string>
> -----END RSA PRIVATE KEY-----
> ```
>
> You pass the *<private key string>* value (excluding the `BEGIN` and `END` lines), either as a single line or as multiple lines.

```
exec dbms_vector_chain.drop_credential('OCI_CRED');


declare
  jo json_object_t;
begin
  jo := json_object_t();
  jo.put('user_ocid','<user ocid>');
  jo.put('tenancy_ocid','<tenancy ocid>');
  jo.put('compartment_ocid','<compartment ocid>');
  jo.put('private_key','<private key>');
  jo.put('fingerprint','<fingerprint>');
  dbms_vector_chain.create_credential(
    credential_name   => 'OCI_CRED',
    params            => json(jo.to_string));
end;
/
```

Replace all the authentication parameter values. For example:

```
declare
  jo json_object_t;
begin

  -- create an OCI credential
  jo := json_object_t();

jo.put('user_ocid','ocid1.user.oc1..aabbalbbaa1112233aabbaabb1111222
aa1111bb');

jo.put('tenancy_ocid','ocid1.tenancy.oc1..aaaaalbbbb1112233aaaabbaa1
111222aaa111a');

jo.put('compartment_ocid','ocid1.compartment.oc1..ababalabab1112233a
babababb1111222aba11ab');
  jo.put('private_key','AAAaaaBBB11112222333...AAA111AAABBB222aaa1a/
+');

jo.put('fingerprint','01:1a:a1:aa:12:a1:12:1a:ab:12:01:ab:a1:12:ab:1
a');
  dbms_vector_chain.create_credential(
```

```
    credential_name   => 'OCI_CRED',
    params            => json(jo.to_string));
end;
/
```

b. Call `UTL_TO_GENERATE_TEXT`:

Here, the cohere.command-r-16k model is used. You can replace `model` with your own value, as required.

> **Note:**
>
> For a list of all REST endpoint URLs and models that are supported to use with Generative AI, see Supported Third-Party Provider Operations and Endpoints.

```
-- select example

var params clob;
exec :params := '
{
  "provider"       : "ocigenai",
  "credential_name": "OCI_CRED",
  "url"            : "https://inference.generativeai.us-
chicago-1.oci.oraclecloud.com/20231130/actions/chat",
  "model"          : "cohere.command-r-16k",
  "chatRequest"    : {
                      "maxTokens": 256
                     }
}';

select dbms_vector_chain.utl_to_generate_text(
 'What is Oracle Text?',json(:params)) from dual;

-- PL/SQL example

declare
  input clob;
  params clob;
  output clob;
begin
  input := 'What is Oracle Text?';

  params := '
{
  "provider"       : "ocigenai",
  "credential_name": "OCI_CRED",
  "url"            : "https://inference.generativeai.us-
chicago-1.oci.oraclecloud.com/20231130/actions/chat",
  "model"          : "cohere.command-r-16k",
  "chatRequest"    : {
                      "maxTokens": 256
                     }
}';
```

```
  output := dbms_vector_chain.utl_to_generate_text(input,
json(params));
  dbms_output.put_line(output);
  if output is not null then
    dbms_lob.freetemporary(output);
  end if;
exception
  when OTHERS THEN
    DBMS_OUTPUT.PUT_LINE (SQLERRM);
    DBMS_OUTPUT.PUT_LINE (SQLCODE);
end;
/
```

Optionally, you can specify additional REST provider-specific parameters.

> **Note:**
>
> If you want to pass any additional REST provider-specific parameters, then you must enclose those in `chatRequest`.

- **Using Cohere, Google AI, Hugging Face, OpenAI, and Vertex AI**:

  a. Call `CREATE_CREDENTIAL` to create and store a credential.

  Cohere, Google AI, Hugging Face, OpenAI, and Vertex AI require the following authentication parameter:

  ```
  { "access_token": "<access token>" }
  ```

  You will later refer to this credential name when declaring JSON parameters for the `UTL_TO_GENERATE_TEXT` call.

  ```
  exec dbms_vector_chain.drop_credential('<credential name>');


  declare
    jo json_object_t;
  begin
    jo := json_object_t();
    jo.put('access_token', '<access token>');
    dbms_vector_chain.create_credential(
      credential_name   => '<credential name>',
      params            => json(jo.to_string));
  end;
  /
  ```

  Replace the `access_token` and `credential_name` values. For example:

  ```
  declare
    jo json_object_t;
  begin
    jo := json_object_t();
    jo.put('access_token', 'AbabA1B123aBc123AbabAb123a1a2ab');
    dbms_vector_chain.create_credential(
  ```

```
    credential_name   => 'HF_CRED',
    params            => json(jo.to_string));
end;
/
```

b. Call `UTL_TO_GENERATE_TEXT`:

```
-- select example

var params clob;
exec :params := '
{
  "provider"       : "<REST provider>",
  "credential_name": "<credential name>",
  "url"            : "<REST endpoint URL for text generation
service>",
  "model"          : "<REST provider text generation model name>"
}';

select dbms_vector_chain.utl_to_generate_text(
 'What is Oracle Text?',json(:params)) from dual;

-- PL/SQL example

declare
  input clob;
  params clob;
  output clob;
begin
  input := 'What is Oracle Text?';

  params := '
{
  "provider"       : "<REST provider>",
  "credential_name": "<credential name>",
  "url"            : "<REST endpoint URL for text generation
service>",
  "model"          : "<REST provider text generation model name>"
}';

  output := dbms_vector_chain.utl_to_generate_text(input,
json(params));
  dbms_output.put_line(output);
  if output is not null then
    dbms_lob.freetemporary(output);
  end if;
exception
  when OTHERS THEN
    DBMS_OUTPUT.PUT_LINE (SQLERRM);
    DBMS_OUTPUT.PUT_LINE (SQLCODE);
end;
/
```

> **Note:**
>
> For a list of all supported REST endpoint URLs, see Supported Third-Party Provider Operations and Endpoints.

Replace `provider`, `credential_name`, `url`, and `model` with your own values. Optionally, you can specify additional REST provider parameters. This is shown in the following examples:

Cohere example:

```
{
  "provider"       : "Cohere",
  "credential_name": "COHERE_CRED",
  "url"            : "https://api.cohere.ai/v1/chat",
  "model"          : "command"
}
```

Google AI example:

```
{
  "provider"       : "googleai",
  "credential_name": "GOOGLEAI_CRED",
  "url"            : "https://generativelanguage.googleapis.com/
v1beta/models/",
  "model"          : "gemini-pro:generateContent"
}
```

Hugging Face example:

```
{
  "provider"       : "huggingface",
  "credential_name": "HF_CRED",
  "url"            : "https://api-inference.huggingface.co/models/",
  "model"          : "gpt2"
}
```

OpenAI example:

```
{
  "provider"       : "openai",
  "credential_name": "OPENAI_CRED",
  "url"            : "https://api.openai.com/v1/chat/completions",
  "model"          : "gpt-4o-mini",
  "max_tokens"     : 60,
  "temperature"    : 1.0
}
```

Vertex AI example:

```
{
  "provider"         : "vertexai",
```

```
  "credential_name"  : "VERTEXAI_CRED",
  "url"              : "https://LOCATION-
aiplatform.googleapis.com/v1/projects/PROJECT/locations/LOCATION/
publishers/google/models/",
  "model"            : "gemini-1.0-pro:generateContent",
  "generation_config": {
                        "temperature"    : 0.9,
                        "topP"           : 1,
                        "candidateCount" : 1,
                        "maxOutputTokens": 256
                       }
}
```

A response to your prompt may appear as:

```
BMS_VECTOR_CHAIN.UTL_TO_GENERATE_TEXT(:INPUT,JSON(:PARAMS))
--------------------------------------------------------------------------------
-----
Oracle Text is a powerful tool that enhances Oracle Database with
integrated
text mining and text analytics capabilities.

It enables users to extract valuable insights and make informed decisions
by
analyzing unstructured text data stored within the database.

Here are some enhanced capabilities offered by Oracle Text:

1. Full-Text Search: Enables powerful and rapid full-text searches across
large
collections of documents. This helps users find relevant information
quickly
and effectively, even within massive datasets.

2. Natural Language Processing: Implements advanced language processing
techn
iques to analyze text and extract meaningful information. This includes
capabilities
like tokenization, stemming, lemmatization, and part-of-speech tagging,
which collectively facilitate efficient text processing and understanding.

3. Sentiment Analysis: Provides a deeper understanding of sentiment
expressed
 in text. It enables businesses to automatically analyze customer
opinions, feed
back, and reviews, helping them gain valuable insights into customer
sentiment,
satisfaction levels, and potential trends.

4. Entity Recognition: Automatically identifies and categorizes entities
with
in text, such as names of people, organizations, locations, or any other
specific
terms of interest. This is useful in applications like customer
relationship
management, where linking relevant information to individuals or
```

```
organizations is
crucial.

5. Contextual Analysis: Delivers insights into the context and
relationships
between entities and concepts in textual data. It helps organizations
better und
erstand the broader implications and associations between entities,
facilitating
 a deeper understanding of their data.

These features collectively empower various applications, enhancing the
function
ality of the Oracle Database platform to allow businesses and
organizations to
derive maximum value from their unstructured text data.

Let me know if you'd like to dive deeper into any of these specific
capabilities
, or if there are other aspects of Oracle Text you'd like to explore
further.
```

This example uses the default settings for each provider. For detailed information on additional parameters, refer to your third-party provider's documentation.

Related Topics

- UTL_TO_GENERATE_TEXT
  Use the `DBMS_VECTOR_CHAIN.UTL_TO_GENERATE_TEXT` chainable utility function to generate a text response for a given prompt or an image, by accessing third-party text generation models.
- DBMS_VECTOR
  The `DBMS_VECTOR` package simplifies common operations with Oracle AI Vector Search, such as extracting chunks or embeddings from user data, generating text for a given prompt or an image, creating a vector index, or reporting on index accuracy.
- DBMS_VECTOR_CHAIN
  The `DBMS_VECTOR_CHAIN` package enables advanced operations with Oracle AI Vector Search, such as chunking and embedding data along with text generation and summarization capabilities. It is more suitable for text processing with similarity search and hybrid search, using functionality that can be pipelined together for an end-to-end search.

Generate Text Using the Local REST Provider Ollama

Perform a text-to-text transformation by accessing open LLMs, using the local host REST endpoint provider Ollama. The input is a textual prompt, and the generated output is a textual answer or description based on the specified task in that prompt.

A prompt can be a text string (such as a question that you ask an LLM or a command), and can include results from a search.

Ollama is a free and open-source command-line interface tool that allows you to run open LLMs (such as Llama 3, Phi 3, Mistral, or Gemma 2) locally and privately on your Linux, Windows, or macOS systems. You can access Ollama as a service using SQL and PL/SQL commands.

Here, you can use the `UTL_TO_GENERATE_TEXT` function from either the `DBMS_VECTOR` or the `DBMS_VECTOR_CHAIN` package, depending on your use case.

> **WARNING:**
>
> Certain features of the database may allow you to access services offered separately by third-parties, for example, through the use of JSON specifications that facilitate your access to REST APIs.
>
> Your use of these features is solely at your own risk, and you are solely responsible for complying with any terms and conditions related to use of any such third-party services. Notwithstanding any other terms and conditions related to the third-party services, your use of such database features constitutes your acceptance of that risk and express exclusion of Oracle's responsibility or liability for any damages resulting from such access.

To generate a descriptive answer for the prompt "`What is Oracle Text?`", by calling a local LLM using Ollama:

1. Connect to Oracle Database as a local user.
   a. Log in to SQL*Plus as the `SYS` user, connecting as `SYSDBA`:

```
conn sys/password as sysdba


CREATE TABLESPACE tbs1
DATAFILE 'tbs5.dbf' SIZE 20G AUTOEXTEND ON
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;


SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 10000
SET LONG 10000
```

   b. Create a local user (`docuser`) and grant necessary privileges:

```
DROP USER docuser cascade;


CREATE USER docuser identified by docuser DEFAULT TABLESPACE tbs1 quota
unlimited on tbs1;


GRANT DB_DEVELOPER_ROLE, create credential to docuser;
```

c. Connect as the local user (`docuser`):

```
CONN docuser/password
```

2. Install Ollama and run a model locally.

a. Download and run the Ollama application from https://ollama.com/download.

You can either install Ollama as a service that runs in the background or as a standalone binary with a manual install. For detailed installation-specific steps, see Quick Start in the Ollama Documentation.

Note the following:

- The Ollama server needs to be able to connect to the internet so that it can download the models. If you require a proxy server to access the internet, remember to set the appropriate environment variables before running the Ollama server. For example, to set for Linux:

```
-- set a proxy if you require one

export https_proxy=<proxy-hostname>:<proxy-port>
export http_proxy=<proxy-hostname>:<proxy-port>
export no_proxy=localhost,127.0.0.1,.example.com
export ftp_proxy=<proxy-hostname>:<proxy-port>
```

- If you are running Ollama and the database on different machines, then on the database machine, you must change the `URL` to refer to the host name or IP address that is running Ollama instead of the local host.
- You may need to change your firewall settings on the machine that is running Ollama to allow the port through.

b. If running Ollama as a standalone binary from a manual install, then start the server:

```
ollama serve
```

c. Run a model using the `ollama run <model_name>` command.

For example, to call the Llama 3 model:

```
ollama run llama3
```

For detailed information on this step, see Ollama Readme.

d. Verify that Ollama is running locally by using a cURL command.

For example:

```
-- generate text

curl -X POST http://localhost:11434/api/generate -d '{
  "model" : "llama3",
  "prompt": "Why is the sky blue?",
  "stream": false }'
```

3. Set the HTTP proxy server, if configured.

```
EXEC UTL_HTTP.SET_PROXY('<proxy-hostname>:<proxy-port>');
```

4. Grant connect privilege to `docuser` for allowing connection to the host, using the `DBMS_NETWORK_ACL_ADMIN` procedure.

   This example uses `*` to allow any host. However, you can explicitly specify the host that you want to connect to.

```
BEGIN
  DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE(
    host => '*',
    ace => xs$ace_type(privilege_list => xs$name_list('connect'),
                       principal_name => 'docuser',
                       principal_type => xs_acl.ptype_db));
END;
/
```

5. Call `UTL_TO_GENERATE_TEXT`.

   The Ollama service has a REST API endpoint for generating text. Specify the URL and other configuration parameters in a JSON object.

```
var gent_ollama_params clob;
exec :gent_ollama_params := '{
   "provider": "ollama",
   "host"    : "local",
   "url"     : "http://localhost:11434/api/generate",
   "model"   : "llama3"
}';

select dbms_vector.utl_to_generate_text('What is Oracle Text?',
json(:gent_ollama_params)) from dual;
```

   You can replace the `url` and `model` with your own values, as required.

> **Note:**
>
> For a complete list of all supported REST endpoint URLs, see Supported Third-Party Provider Operations and Endpoints.

   The response to your prompt may appear as:

```
Oracle Text (formerly known as Oracle InterMedia) is a suite of text
search and retrieval tools within Oracle Database. It allows you to index
and query
unstructured text data, such as documents, emails, and other text-based
content.

With Oracle Text, you can:

1. Index text: Create indexes on text columns or external files, making it
possible to efficiently search and retrieve relevant text data.
2. Query text: Use SQL syntax to query the indexed text data, allowing you
to find specific words, phrases, or patterns within large volumes of
text.
3. Full-text search: Perform full-text searches on unstructured text data,
```

```
returning relevant results based on keyword matches, proximity, and
relevance.

Oracle Text supports various indexing schemes, including:

1. Basic Indexing: A simple, fast index for searching exact keywords.
2. Phrase Indexing: An index that allows you to search for phrases (e.g.,
"John Smith").
3. Thesaurus Indexing: An index that enables searches based on synonyms
and related words.

Oracle Text also includes various text analysis and processing features,
such as:

1. Tokenization: Breaking down text into individual words or tokens.
2. Stemming: Reducing words to their base form (e.g., "running" becomes
"run").
3. Stopword removal: Eliminating common words like "the," "and," and "a"
that don't add much value to the search.

Oracle Text is particularly useful in scenarios where you need to search,
analyze, or retrieve unstructured text data, such as:

1. Content management: Searching and retrieving documents, articles, or
other content.
2. Email archiving: Indexing and searching email messages.
3. Search engines: Building custom search solutions for specific domains
or industries.

In summary, Oracle Text is a powerful tool within the Oracle Database that
enables you to index, query, and retrieve unstructured text data with ease.
```

Related Topics

- UTL_TO_GENERATE_TEXT
  Use the `DBMS_VECTOR.UTL_TO_GENERATE_TEXT` chainable utility function to generate a text response for a given prompt or an image, by accessing third-party text generation models.
- DBMS_VECTOR
  The `DBMS_VECTOR` package simplifies common operations with Oracle AI Vector Search, such as extracting chunks or embeddings from user data, generating text for a given prompt or an image, creating a vector index, or reporting on index accuracy.
- DBMS_VECTOR_CHAIN
  The `DBMS_VECTOR_CHAIN` package enables advanced operations with Oracle AI Vector Search, such as chunking and embedding data along with text generation and summarization capabilities. It is more suitable for text processing with similarity search and hybrid search, using functionality that can be pipelined together for an end-to-end search.

Describe Image Content

In these examples, you can see how to generate a textual analysis or description of the contents of a given image.

Here, you supply an image along with a text question as the prompt (for example, "`What is this image about?`" or "`How many birds are there in this image?`"). The LLM responds

with a textual answer or description based on the specified task in the prompt, which can then be used for image classification, object detection, or similarity search.

- Describe Images Using Public REST Providers
  Perform an image-to-text transformation by supplying an image along with a text question as the prompt, using publicly hosted third-party LLMs by Google AI, Hugging Face, OpenAI, or Vertex AI.
- Describe Images Using the Local REST Provider Ollama
  Perform an image-to-text transformation by supplying an image along with a text question as the prompt by accessing open LLMs, using the local host REST endpoint provider Ollama.

Describe Images Using Public REST Providers

Perform an image-to-text transformation by supplying an image along with a text question as the prompt, using publicly hosted third-party LLMs by Google AI, Hugging Face, OpenAI, or Vertex AI.

Here, you can use the `UTL_TO_GENERATE_TEXT` function from either the `DBMS_VECTOR` or the `DBMS_VECTOR_CHAIN` package, depending on your use case.

> **WARNING:**
>
> Certain features of the database may allow you to access services offered separately by third-parties, for example, through the use of JSON specifications that facilitate your access to REST APIs.
>
> Your use of these features is solely at your own risk, and you are solely responsible for complying with any terms and conditions related to use of any such third-party services. Notwithstanding any other terms and conditions related to the third-party services, your use of such database features constitutes your acceptance of that risk and express exclusion of Oracle's responsibility or liability for any damages resulting from such access.

To generate a text response by prompting with the following image and the text question "`Describe this image?`", using an external LLM:

1. Connect to Oracle Database as a local user.

a. Log in to SQL*Plus as the `SYS` user, connecting as `SYSDBA`:

```
conn sys/password as sysdba


CREATE TABLESPACE tbs1
DATAFILE 'tbs5.dbf' SIZE 20G AUTOEXTEND ON
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;


SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 10000
SET LONG 10000
```

b. Create a local user (`docuser`) and grant necessary privileges:

```
DROP USER docuser cascade;


CREATE USER docuser identified by docuser DEFAULT TABLESPACE tbs1 quota
unlimited on tbs1;


GRANT DB_DEVELOPER_ROLE, create credential to docuser;
```

c. Connect as the local user (`docuser`):

```
CONN docuser/password
```

2. Set the HTTP proxy server, if configured.

```
EXEC UTL_HTTP.SET_PROXY('<proxy-hostname>:<proxy-port>');
```

3. Grant connect privilege to `docuser` for allowing connection to the host, using the `DBMS_NETWORK_ACL_ADMIN` procedure.

   This example uses `*` to allow any host. However, you can explicitly specify the host that you want to connect to.

```
BEGIN
  DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE(
    host => '*',
    ace => xs$ace_type(privilege_list => xs$name_list('connect'),
                       principal_name => 'docuser',
                       principal_type => xs_acl.ptype_db));
END;
/
```

4. Create a local directory (`DEMO_DIR`) to store your image file:

```
create or replace directory DEMO_DIR as '/my_local_dir/';

create or replace function load_blob_from_file(directoryname varchar2,
filename varchar2)
  return blob
is
  filecontent blob := null;
  src_file bfile := bfilename(directoryname, filename);
  offset number := 1;
begin
  dbms_lob.createtemporary(filecontent, true, dbms_lob.session);
  dbms_lob.fileopen(src_file, dbms_lob.file_readonly);
  dbms_lob.loadblobfromfile(filecontent, src_file,
    dbms_lob.getlength(src_file), offset, offset);
  dbms_lob.fileclose(src_file);
  return filecontent;
end;
/
```

   Upload the image file (for example, `bird.jpg`) to the directory.

5. Set up your credentials for the REST provider that you want to access and then call `UTL_TO_GENERATE_TEXT`:

   a. Call `CREATE_CREDENTIAL` to create and store a credential.

      Google AI, Hugging Face, OpenAI, and Vertex AI require the following authentication parameter:

      ```
      { "access_token": "<access token>" }
      ```

      You will later refer to this credential name when declaring JSON parameters for the `UTL_TO_GENERATE_TEXT` call.

      ```
      exec dbms_vector_chain.drop_credential('<credential name>');


      declare
        jo json_object_t;
      begin
        jo := json_object_t();
        jo.put('access_token', '<access token>');
        dbms_vector_chain.create_credential(
          credential_name   => '<credential name>',
          params            => json(jo.to_string));
      end;
      /
      ```

      Replace `access_token` and `credential_name` with your own values. For example:

      ```
      declare
        jo json_object_t;
      begin
        jo := json_object_t();
        jo.put('access_token', 'AbabA1B123aBc123AbabAb123a1a2ab');
      ```

```
  dbms_vector_chain.create_credential(
    credential_name   => 'HF_CRED',
    params            => json(jo.to_string));
end;
/
```

b. Call `UTL_TO_GENERATE_TEXT`:

```
-- select example

var input clob;
var media_data blob;
var media_type clob;
var params clob;

begin
  :input := 'Describe this image';
  :media_data := load_blob_from_file('DEMO_DIR', 'bird.jpg');
  :media_type := 'image/jpeg';
  :params := '
{
  "provider"       : "<REST provider>",
  "credential_name": "<credential name>",
  "url"            : "<REST endpoint URL for text generation service>",
  "model"          : "<REST provider text generation model name>",
  "max_tokens"     : <maximum number of tokens in the output text>
}';
end;
/

select
dbms_vector_chain.utl_to_generate_text(:input, :media_data, :media_type,
 json(:params));

-- PL/SQL example

declare
  input clob;
  media_data blob;
  media_type varchar2(32);
  params clob;
  output clob;
begin
  input := 'Describe this image';
  media_data := load_blob_from_file('DEMO_DIR', 'bird.jpg');
  media_type := 'image/jpeg';
  params := '
{
  "provider"       : "<REST provider>",
  "credential_name": "<credential name>",
  "url"            : "<REST endpoint URL for text generation service>",
  "model"          : "<REST provider text generation model name>",
  "max_tokens"     : <maximum number of tokens in the output text>
}';

  output := dbms_vector_chain.utl_to_generate_text(
```

```
    input, media_data, media_type, json(params));
  dbms_output.put_line(output);

  if output is not null then
    dbms_lob.freetemporary(output);
  end if;
  if media_data is not null then
    dbms_lob.freetemporary(media_data);
  end if;
exception
  when OTHERS THEN
    DBMS_OUTPUT.PUT_LINE (SQLERRM);
    DBMS_OUTPUT.PUT_LINE (SQLCODE);
end;
/
```

> **Note:**
>
> For a list of all supported REST endpoint URLs, see Supported Third-Party Provider Operations and Endpoints.

Replace `provider`, `credential_name`, `url`, and `model` with your own values. Optionally, you can specify additional REST provider parameters. This is shown in the following examples:

Google AI example:

```
{
  "provider"       : "googleai",
  "credential_name": "GOOGLEAI_CRED",
  "url"            : "https://generativelanguage.googleapis.com/v1beta/
models/",
  "model"          : "gemini-pro:generateContent"
}
```

Hugging Face example:

```
{
  "provider"       : "huggingface",
  "credential_name": "HF_CRED",
  "url"            : "https://api-inference.huggingface.co/models/",
  "model"          : "gpt2"
}
```

> **Note:**
>
> Hugging Face uses an image captioning model, which does not require a prompt. If you input a prompt along with an image, then the prompt will be ignored.

OpenAI example:

```
{
  "provider"       : "openai",
  "credential_name": "OPENAI_CRED",
  "url"            : "https://api.openai.com/v1/chat/completions",
  "model"          : "gpt-4o-mini",
  "max_tokens"     : 60
}
```

Vertex AI example:

```
{
  "provider"         : "vertexai",
  "credential_name"  : "VERTEXAI_CRED",
  "url"              : "https://LOCATION-aiplatform.googleapis.com/v1/
projects/PROJECT/locations/LOCATION/publishers/google/models/",
  "model"            : "gemini-1.0-pro:generateContent",
  "generation_config": {
                        "temperature"    : 0.9,
                        "topP"           : 1,
                        "candidateCount" : 1,
                        "maxOutputTokens": 256
                       }
}
```

A generated text response to your question may appear as:

```
This image showcases a stylized, artistic depiction of a hummingbird in
mid-flight, set against a vibrant red background. The bird is illustrated
with a
mix of striking colors and details - its head and belly are shown in
white, with
a black patterned detailing that resembles stripes or scales. Its long,
slender
beak is depicted in a darker color, extending forwards. The hummingbird's
wings
and tail are rendered in eye-catching shades of orange, purple, and red,
with
texture that suggests a rough, perhaps brush-like stroke.

The background features abstract shapes resembling clouds or wind currents
in a white line
pattern, which adds a sense of motion or air dynamics around the bird. The
overall use of vivid colors and dynamic patterns gives the image an
energetic
and modern feel.
```

This example uses the default settings for each provider. For detailed information on additional parameters, refer to your third-party provider's documentation.

Related Topics

- UTL_TO_GENERATE_TEXT
  Use the `DBMS_VECTOR_CHAIN.UTL_TO_GENERATE_TEXT` chainable utility function to generate a text response for a given prompt or an image, by accessing third-party text generation models.
- DBMS_VECTOR
  The `DBMS_VECTOR` package simplifies common operations with Oracle AI Vector Search, such as extracting chunks or embeddings from user data, generating text for a given prompt or an image, creating a vector index, or reporting on index accuracy.
- DBMS_VECTOR_CHAIN
  The `DBMS_VECTOR_CHAIN` package enables advanced operations with Oracle AI Vector Search, such as chunking and embedding data along with text generation and summarization capabilities. It is more suitable for text processing with similarity search and hybrid search, using functionality that can be pipelined together for an end-to-end search.

Describe Images Using the Local REST Provider Ollama

Perform an image-to-text transformation by supplying an image along with a text question as the prompt by accessing open LLMs, using the local host REST endpoint provider Ollama.

Ollama is a free and open-source command-line interface tool that allows you to run open LLMs (such as Llama 3, Llava, Phi 3, Mistral, or Gemma 2) locally and privately on your Linux, Windows, or macOS systems. You can access Ollama as a service using SQL and PL/SQL commands.

Here, you can use the `UTL_TO_GENERATE_TEXT` function from either the `DBMS_VECTOR` or the `DBMS_VECTOR_CHAIN` package, depending on your use case.

> **WARNING:**
>
> Certain features of the database may allow you to access services offered separately by third-parties, for example, through the use of JSON specifications that facilitate your access to REST APIs.
>
> Your use of these features is solely at your own risk, and you are solely responsible for complying with any terms and conditions related to use of any such third-party services. Notwithstanding any other terms and conditions related to the third-party services, your use of such database features constitutes your acceptance of that risk and express exclusion of Oracle's responsibility or liability for any damages resulting from such access.

To describe the contents of an image, by prompting with the following image of a bird and the text question "`Describe this image?`", by calling a local LLM using Ollama:

1. Connect to Oracle Database as a local user.
   a. Log in to SQL*Plus as the `SYS` user, connecting as `SYSDBA`:

```
conn sys/password as sysdba


CREATE TABLESPACE tbs1
DATAFILE 'tbs5.dbf' SIZE 20G AUTOEXTEND ON
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;


SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 10000
SET LONG 10000
```

   b. Create a local user (`docuser`) and grant necessary privileges:

```
DROP USER docuser cascade;


CREATE USER docuser identified by docuser DEFAULT TABLESPACE tbs1 quota
unlimited on tbs1;


GRANT DB_DEVELOPER_ROLE, create credential to docuser;
```

   c. Connect as the local user (`docuser`):

```
CONN docuser/password
```

2. Install Ollama and run a model locally.
   a. Download and run the Ollama application from https://ollama.com/download.

You can either install Ollama as a service that runs in the background or as a standalone binary with a manual install. For detailed installation-specific steps, see Quick Start in the Ollama Documentation.

Note the following:

- The Ollama server needs to be able to connect to the internet so that it can download the models. If you require a proxy server to access the internet, remember to set the appropriate environment variables before running the Ollama server. For example, to set for Linux:

```
-- set a proxy if you require one

export https_proxy=<proxy-hostname>:<proxy-port>
export http_proxy=<proxy-hostname>:<proxy-port>
export no_proxy=localhost,127.0.0.1,.example.com
export ftp_proxy=<proxy-hostname>:<proxy-port>
```

- If you are running Ollama and the database on different machines, then on the database machine, you must change the `URL` to refer to the host name or IP address that is running Ollama instead of the local host.
- You may need to change your firewall settings on the machine that is running Ollama to allow the port through.

b. If running Ollama as a standalone binary from a manual install, then start the server:

```
ollama serve
```

c. Run a model using the `ollama run <model_name>` command.

For example, to call the llava model:

```
ollama run llava
```

For detailed information on this step, see Ollama Readme.

d. Verify that Ollama is running locally by using a cURL command.

For example:

```
-- generate text

curl -X POST http://localhost:11434/api/generate -d '{
  "model" : "llava",
  "prompt": "Why is the sky blue?",
  "stream": false }'
```

3. Set the HTTP proxy server, if configured.

```
EXEC UTL_HTTP.SET_PROXY('<proxy-hostname>:<proxy-port>');
```

4. Grant connect privilege to `docuser` for allowing connection to the host, using the `DBMS_NETWORK_ACL_ADMIN` procedure.

This example uses * to allow any host. However, you can explicitly specify the host that you want to connect to.

```
BEGIN
  DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE(
    host => '*',
    ace => xs$ace_type(privilege_list => xs$name_list('connect'),
                       principal_name => 'docuser',
                       principal_type => xs_acl.ptype_db));
END;
/
```

5. Create a local directory (DEMO_DIR) to store your image file:

```
create or replace directory DEMO_DIR as '/my_local_dir/';

create or replace function load_blob_from_file(directoryname varchar2,
filename varchar2)
  return blob
is
  filecontent blob := null;
  src_file bfile := bfilename(directoryname, filename);
  offset number := 1;
begin
  dbms_lob.createtemporary(filecontent, true, dbms_lob.session);
  dbms_lob.fileopen(src_file, dbms_lob.file_readonly);
  dbms_lob.loadblobfromfile(filecontent, src_file,
    dbms_lob.getlength(src_file), offset, offset);
  dbms_lob.fileclose(src_file);
  return filecontent;
end;
/
```

Upload the image file (for example, bird.jpg) to the directory.

6. Call UTL_TO_GENERATE_TEXT.

The Ollama service has a REST API endpoint for generating text. Specify the URL and other configuration parameters in a JSON object.

```
var input clob;
var media_data blob;
var media_type clob;
var gent_ollama_params clob;

  :input := 'Describe this image';
  :media_data := load_blob_from_file('DEMO_DIR', 'bird.jpg');
  :media_type := 'image/jpeg';
  :gent_ollama_params := '{
     "provider": "ollama",
     "host"    : "local",
     "url"     : "http://localhost:11434/api/generate",
     "model"   : "llava"
  }';

select
```

```
dbms_vector_chain.utl_to_generate_text(:input, :media_data, :media_type,
json(:gent_ollama_params)) from dual;
```

You can replace `url` and `model` with your own values, as required.

> **Note:**
>
> For a complete list of all supported REST endpoint URLs, see Supported Third-Party Provider Operations and Endpoints.

A generated text response to your question may appear as:

```
This is an image of a stylized, artistic depiction of a hummingbird in
mid-flight, set against a vibrant red background. The bird is illustrated
with a
mix of striking colors and details - its head and belly are shown in
white, with
a black patterned detailing that resembles stripes or scales. Its long,
slender
beak is depicted in a darker color, extending forwards. The hummingbird's
wings
and tail are rendered in eye-catching shades of orange, purple, and red,
with
texture that suggests a rough, perhaps brush-like stroke.

The background features abstract shapes resembling clouds or wind currents
in a white line
pattern, which adds a sense of motion or air dynamics around the bird. The
overall use of vivid colors and dynamic patterns gives the image an
energetic
and modern feel.
```

Related Topics

- UTL_TO_GENERATE_TEXT
  Use the `DBMS_VECTOR_CHAIN.UTL_TO_GENERATE_TEXT` chainable utility function to generate a text response for a given prompt or an image, by accessing third-party text generation models.
- DBMS_VECTOR
  The `DBMS_VECTOR` package simplifies common operations with Oracle AI Vector Search, such as extracting chunks or embeddings from user data, generating text for a given prompt or an image, creating a vector index, or reporting on index accuracy.
- DBMS_VECTOR_CHAIN
  The `DBMS_VECTOR_CHAIN` package enables advanced operations with Oracle AI Vector Search, such as chunking and embedding data along with text generation and summarization capabilities. It is more suitable for text processing with similarity search and hybrid search, using functionality that can be pipelined together for an end-to-end search.

Use Retrieval Augmented Generation to Complement LLMs

RAG lets you mitigate the inaccuracies and hallucinations faced when using LLMs. Oracle AI Vector Search enables RAG within Oracle Database using the `DBMS_VECTOR_CHAIN` PL/SQL package or through popular frameworks (such as LangChain).

- About Retrieval Augmented Generation
  Oracle AI Vector Search supports Enterprise Retrieval Augmented Generation (RAG) to enable sophisticated queries that can combine vectors with relational data, graph data, spatial data, and JSON collections.
- SQL RAG Example
  This scenario allows you to run a similarity search for specific documentation content based on a user query. Once documentation chunks are retrieved, they are concatenated and a prompt is generated to ask an LLM to answer the user question using retrieved chunks.
- Oracle AI Vector Search Integration with LangChain
  LangChain is a powerful and flexible open source orchestration framework that helps developers build applications that leverage the advanced capabilities of large language models (LLMs).
- Oracle AI Vector Search Integration with LlamaIndex
  LlamaIndex is an open-source data framework designed to simplify the process of building applications that leverage large language models (LLMs) with custom data. Basically, LlamaIndex acts as a bridge between custom data sources and LLMs such as Cohere Command models or OpenAI GPTs models.
- Use Reranking for Better RAG Results
  Reranking models are primarily used to reassess and reorder an initial set of search results. This helps to improve the relevance and quality of search results in both similarity search and Retrieval Augmented Generation (RAG) scenarios.

About Retrieval Augmented Generation

Oracle AI Vector Search supports Enterprise Retrieval Augmented Generation (RAG) to enable sophisticated queries that can combine vectors with relational data, graph data, spatial data, and JSON collections.

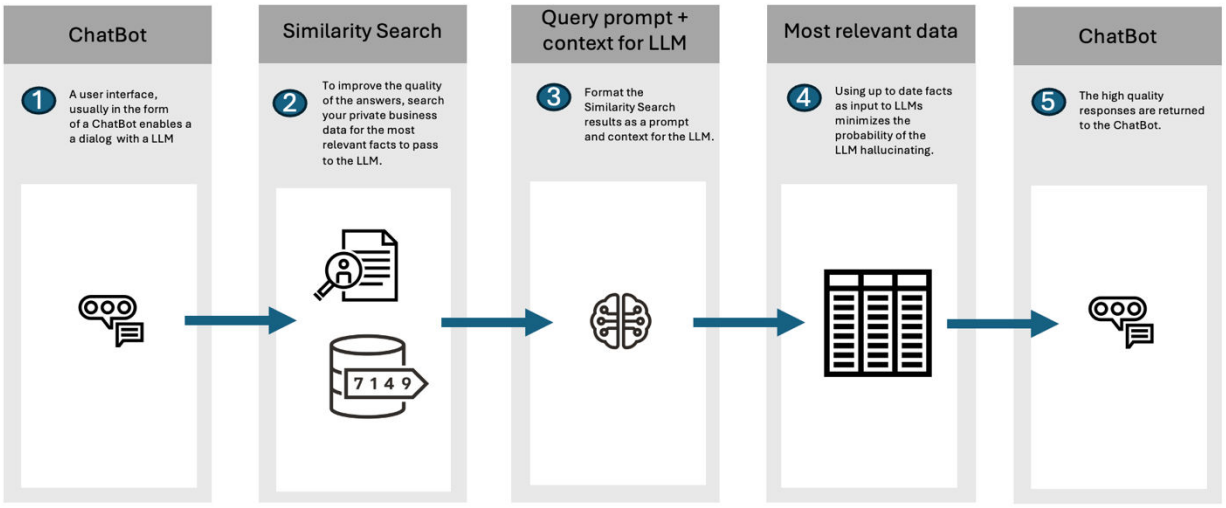
Retrieval Augmented Generation is an approach developed to address the limitations of LLMs. RAG combines the strengths of pretrained language models with the ability to retrieve recent and accurate information from a dataset or database in real-time during the generation of responses.

By communicating with LLMs through the implementation of RAG, the knowledge of LLMs is increased with business data found through AI Vector Search.

The primary problem with LLMs, such as GPT (Generative Pretrained Transformer), is that they generate responses based solely on the patterns and data they were trained on up to the point of their last update. This means that they inherently lack the ability to access or incorporate new, real-time information after their training is cut off, potentially limiting their responses to outdated or incomplete information. LLMs do not know about your private company data. Consequently, LLMs can make up answers (hallucinate) when they do not have enough relevant and up-to-date facts.

By providing your LLM with up-to-date facts from your company, you can minimize the probability that an LLM will hallucinate.

Figure 9-1 Example RAG Workflow



Here is how RAG improves upon the issues with traditional LLMs:

- **Access to External and Private Information:** RAG can pull in data from external and private sources during its response generation process. This allows it to provide answers that are up-to-date and grounded in the latest available information, which is crucial for queries requiring current knowledge or specific details not included in its original training data.
- **Factually More Accurate and Detailed Responses:** While traditional LLMs are trained on older data, RAG incorporates real-time retrieved information, meaning that generated responses are not only contextually rich but also factually more up-to-date and accurate as time goes on. This is particularly beneficial for queries that require precision and detail, such as scientific facts, historical data, or specific statistics.
- **Reduced Hallucination:** LLMs can sometimes "hallucinate" information, as in generate plausible but false or unverified content. RAG mitigates this by grounding responses in retrieved documents, thereby enhancing the reliability of the information provided.

Oracle AI Vector Search enables RAG within Oracle Database using the `DBMS_VECTOR_CHAIN` PL/SQL package. Alternatively, you can implement RAG externally by using popular frameworks such as LangChain.

LangChain is a popular open source framework that encapsulates popular LLMs, vector databases, document stores, and embedding models. `DBMS_VECTOR_CHAIN` is a PL/SQL package that provides the ability to create RAG solutions, all within the database. With `DBMS_VECTOR_CHAIN`, your data never needs to leave the security of Oracle Database.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for details about the `DBMS_VECTOR_CHAIN` package

SQL RAG Example

This scenario allows you to run a similarity search for specific documentation content based on a user query. Once documentation chunks are retrieved, they are concatenated and a prompt is generated to ask an LLM to answer the user question using retrieved chunks.

1. Start SQL*Plus and connect to Oracle Database as a local test user.
   a. Log in to SQL*Plus as the `sys` user, connecting as `sysdba`:

      ```
      conn sys/password AS sysdba


      SET SERVEROUTPUT ON;
      SET ECHO ON;
      SET LONG 100000;
      ```

   b. Create a local test user (`vector`) and grant necessary privileges:

      ```
      DROP USER vector cascade;


      CREATE USER vector identified by <my vector password>


      GRANT DB_DEVELOPER_ROLE, CREATE CREDENTIAL TO vector;
      ```

   c. Set the proxy if one exists:

      ```
      EXEC UTL_HTTP.SET_PROXY('<my proxy full name>:<my proxy port>');
      ```

   d. Grant connect privilege for a host using the `DBMS_NETWORK_ACL_ADMIN` procedure. This example uses `*` to allow any host. However, you can explicitly specify each host that you want to connect to.

      ```
      BEGIN
        DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE(
          host => '*',
          ace => xs$ace_type(privilege_list => xs$name_list('connect'),
                             principal_name => 'VECTOR',
                             principal_type => xs_acl.ptype_db));
      END;
      /
      ```

   e. Connect to Oracle Database as the test user.

      ```
      conn docuser/password;
      ```

2. Create a credential for Oracle Cloud Infrastructure Generative AI:
   a. Run `DBMS_VECTOR_CHAIN.CREATE_CREDENTIAL` to create and store an OCI credential (`OCI_CRED`).

OCIGenAI requires the following parameters:

```
{
"user_ocid"       : "<user ocid>",
"tenancy_ocid"    : "<tenancy ocid>",
"compartment_ocid": "<compartment ocid>",
"private_key"     : "<private key>",
"fingerprint"     : "<fingerprint>"
}
```

> **Note:**
>
> The generated private key may appear as:
>
> ```
> -----BEGIN RSA PRIVATE KEY-----
> <private key string>
> -----END RSA PRIVATE KEY-----
> ```
>
> You pass the `<private key string>` value (excluding the `BEGIN` and `END` lines), either as a single line or as multiple lines.

```
BEGIN
  DBMS_VECTOR_CHAIN.DROP_CREDENTIAL(credential_name => 'OCI_CRED');
EXCEPTION
  WHEN OTHERS THEN NULL;
END;
/


DECLARE
  jo json_object_t;
BEGIN
  jo := json_object_t();
  jo.put('user_ocid', '<user ocid>');
  jo.put('tenancy_ocid', '<tenancy ocid>');
  jo.put('compartment_ocid', '<compartment ocid>');
  jo.put('private_key', '<private key>');
  jo.put('fingerprint', '<fingerprint>');
  DBMS_VECTOR_CHAIN.CREATE_CREDENTIAL(
    credential_name => 'OCID_CRED',
    params          => json(jo.to_string));
END;
/
```

b. Check credential creation:

```
col owner format a15
col credential_name format a20
col username format a20


SELECT owner, credential_name, username
FROM all_credentials
ORDER BY owner, credential_name, username;
```

3. Generate a prompt using similarity search results:

> **Note:**
>
> For information about loading the ONNX format model into the database as `doc_model`, see Import ONNX Models into Oracle Database End-to-End Example.

```
SET SERVEROUTPUT ON;

VAR prompt CLOB;
VAR user_question CLOB;
VAR context CLOB;

BEGIN
  -- initialize the concatenated string
  :context := '';

  -- read this question from the user
  :user_question := 'what are vector indexes?';

  -- cursor to fetch chunks relevant to the user's query
  FOR rec IN (SELECT EMBED_DATA
              FROM doc_chunks
              WHERE DOC_ID = 'Vector User Guide'
              ORDER BY vector_distance(embed_vector, vector_embedding(
                  doc_model using :user_question as data), COSINE)
              FETCH EXACT FIRST 10 ROWS ONLY)
  LOOP
    -- concatenate each value to the string
    :context := :context || rec.embed_data;
  END LOOP;

  -- concatenate strings and format it as an enhanced prompt to the LLM
  :prompt := 'Answer the following question using the supplied context
               assuming you are a subject matter expert. Question: '
               || :user_question || ' Context: ' || :context;

  DBMS_OUTPUT.PUT_LINE('Generated prompt: ' || :prompt);
END;
/
```

4. Issue the GenAI call:

> **Note:**
>
> For a list of all supported REST endpoints, see Supported Third-Party Provider Operations and Endpoints.

```
DECLARE
  input CLOB;
  params CLOB;
  output CLOB;
BEGIN
  input := :prompt;
  params := '{
    "provider" : "ocigenai",
    "credential_name" : "OCI_CRED",
    "url" : "https://inference.generativeai.us-chicago-1.oci.
            oraclecloud.com/20231130/actions/generateText",
    "model" : "cohere.command"
  }';

  output := DBMS_VECTOR_CHAIN.UTL_TO_GENERATE_TEXT(input, json(params));
  DBMS_OUTPUT.PUT_LINE(output);
  IF output IS NOT NULL THEN
    DBMS_LOB.FREETEMPORARY(output);
  END IF;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE(SQLERRM);
    DBMS_OUTPUT.PUT_LINE(SQLCODE);
END;
/
```

Oracle AI Vector Search Integration with LangChain

LangChain is a powerful and flexible open source orchestration framework that helps developers build applications that leverage the advanced capabilities of large language models (LLMs).

LangChain provides essential tools for managing workflows, maintaining context, and integrating with external systems. For example, the LangChain framework allows you to create Chatbots and agent applications. LangChain primarily supports Python but also has support for JavaScript and TypeScript.

Oracle AI Vector Search integrates with LangChain at various levels:

- Document Loaders
- Text Splitter
- Embeddings
- Summary
- Vector Store

For more information about each of these components, see LangChain Oracle AI Vector Search documentation.

> **See Also:**
>
> - LangChain documentation for an introduction to the LangChain framwork
> - LangChain component documentation for a list of LangChain components
> - LangChain installation documentation to learn how to install LangChain packages in Python
> - Oracle AI Vector Search integration demo for an end-to-end tutorial that demonstrates how Oracle AI Vector Search can be used with LangChain to serve as an end-to-end RAG pipeline

Oracle AI Vector Search Integration with LlamaIndex

LlamaIndex is an open-source data framework designed to simplify the process of building applications that leverage large language models (LLMs) with custom data. Basically, LlamaIndex acts as a bridge between custom data sources and LLMs such as Cohere Command models or OpenAI GPTs models.

Oracle AI Vector Search is integrated with LlamaIndex in several ways to enable powerful semantic search and retrieval capabilities.

Here are the key aspects of this integration:

- **Embedding Generation**

  Oracle AI Vector Search provides embedding capabilities that can be used with LlamaIndex:

  - The `OracleEmbeddings` class from LlamaIndex can be used to generate embeddings using Oracle's embedding models.
  - Multiple embedding methods are supported, including locally-hosted ONNX models and third-party APIs such as Generative AI and Hugging Face.
  - Embeddings can be generated for documents and queries to enable semantic similarity search.

  For more information on how to use this integration for generating embeddings, see Oracle AI Vector Search: Use Embedding Generation Capabilities.

- **Vector Storage**

  LlamaIndex can leverage Oracle Database as a vector store:

  - Vector embeddings can be stored alongside business data in Oracle Database tables using the `VECTOR` data type.
  - This allows combining semantic search on unstructured data with relational queries on structured data in a single system.

  For more information on how to use this integration for vector storage, see Oracle AI Vector Search: Use Vector Storage Capabilities.

- **Indexing and Search**

You can utilize Oracle's vector indexing and search capabilities:

- Vector indexes can be created on the embeddings to enable fast similarity search.
- LlamaIndex can use Oracle's native SQL operations for similarity search to retrieve relevant data.
- Various distance metrics, such as dot product, cosine similarity, Euclidean distance, and more are supported.

For more information about how to use this integration for indexing and search, see Oracle AI Vector Search: Use Document Processing Capabilities and Oracle AI Vector Search: End-to-End Pipeline with Document Processing.

- **RAG Pipeline Integration**

  The integration enables building end-to-end Retrieval Augmented Generation (RAG) pipelines. You can embed and store unstructured data in Oracle Database. LlamaIndex can query the vector store to retrieve relevant context. The retrieved information can be used to generate prompts for LLMs.

  LlamaIndex provides several libraries and classes to integrate Oracle AI Vector Search capabilities. Here are the key components available:

  - `OracleEmbeddings`: Supports multiple embedding methods, including locally hosted ONNX models and third-party APIs such as Generative AI and Hugging Face.
  - `OracleReader`: Used for loading documents from various sources, including Oracle Database.
  - `OracleSummary`: Provides functionality for summarizing documents within or outside the database.
  - `OracleTextSplitter`: Offers advanced Oracle capabilities for chunking documents according to different requirements.
  - `OraLlamaVS`: Used for storing, indexing, and querying vector embeddings.

  For more information about how to use this integration for building end-to-end RAG pipelines, see Oracle AI Vector Search: End-to-End Pipeline with Document Processing.

Benefits

The Oracle AI Vector Search integration with LlamaIndex provides a powerful foundation for developing sophisticated AI applications that can leverage both structured and unstructured data within the Oracle ecosystem.

By integrating Oracle AI Vector Search with LlamaIndex, developers can:

- Leverage Oracle Database's enterprise features such as scalability, security, and transactions.
- Combine semantic search with relational queries in one single system.
- Utilize Oracle's optimized vector operations for efficient similarity search.
- Build AI-powered applications using familiar SQL and PL/SQL interfaces.

Use Reranking for Better RAG Results

Reranking models are primarily used to reassess and reorder an initial set of search results. This helps to improve the relevance and quality of search results in both similarity search and Retrieval Augmented Generation (RAG) scenarios.

In a RAG scenario, reranking plays a crucial role in improving the quality of information ingested into an LLM by ensuring that the most relevant documents or chunks are prioritized. This can reduce hallucinations and improve the accuracy of generated outputs. The reranking step is typically performed after an initial search that uses a faster but less precise embedding model. The reranker helps to identify the most pertinent information for a given query, but is more expensive in terms of resources because it often employs sophisticated matching methods that go beyond simple vector comparisons.

Here, you can use the `RERANK` function from either the `DBMS_VECTOR` or the `DBMS_VECTOR_CHAIN` package, depending on your use case.

Oracle AI Vector Search supports reranking models provided by Cohere and Vertex AI.

> **WARNING:**
>
> Certain features of the database may allow you to access services offered separately by third-parties, for example, through the use of JSON specifications that facilitate your access to REST APIs.
>
> Your use of these features is solely at your own risk, and you are solely responsible for complying with any terms and conditions related to use of any such third-party services. Notwithstanding any other terms and conditions related to the third-party services, your use of such database features constitutes your acceptance of that risk and express exclusion of Oracle's responsibility or liability for any damages resulting from such access.

To rerank results for the query "`What are some interesting characteristics of the Jovian satellites?`", using a third-party reranking model:

1. Log in to SQL*Plus as the `SYS` user, connecting as `SYSDBA`:

   ```
   conn sys/password as sysdba
   ```

2. Create a local user (`vector`) and grant necessary privileges:

   ```
   DROP USER vector cascade;


   CREATE USER vector identified by <my vector password>


   GRANT DB_DEVELOPER_ROLE, CREATE CREDENTIAL TO vector;
   ```

3. Set proxy if one exists.

   ```
   EXEC UTL_HTTP.SET_PROXY('<my proxy full name>:<my proxy port>');
   ```

4. Grant connect privilege for a host using the `DBMS_NETWORK_ACL_ADMIN` procedure. This example uses `*` to allow any host. However, you can explicitly specify each host that you want to connect to:

```
BEGIN
  DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE(
    host => '*',
    ace => xs$ace_type(privilege_list => xs$name_list('connect'),
                       principal_name => 'VECTOR',
                       principal_type => xs_acl.ptype_db));
END;
/
```

5. Connect to Oracle Database as the test user:

```
conn vector/password;
```

6. Simulate the initial retrieval step that returns, in the specified order, potentially relevant documents for the following query: "`What are some interesting characteristics of the Jovian satellites?`":

```
set echo on

set serveroutput on

var query clob;

var initial_retrieval_docs clob;

exec :query := 'What are some interesting characteristics of the Jovian
satellites?';

begin
  :initial_retrieval_docs := '
{
   "documents": [
                "Jupiter boasts an impressive system of 95 known moons,
including the four largest Galilean satellites.",
                "Jupiter's immense mass, 318 times that of Earth,
significantly influences the orbits of other bodies in the Solar System.",
                "Io, one of Jupiter's Galilean moons, is the most
volcanically active body in our solar system.",
                "The gas giant completes one orbit around the Sun in just
under 12 years, traveling at an average speed of 13 kilometers per
second.",
                "Jupiter's composition is similar to that of the Sun, and
it could have become a brown dwarf if its mass had been 80 times greater."
               ]
}';
end;
/
```

7. Set up your credentials for the REST provider that you want to access, that is, Cohere or Vertex AI. Here, replace *<access token>* with your own values:

- Cohere example:

```
EXEC DBMS_VECTOR.DROP_CREDENTIAL('COHERE_CRED');

DECLARE
   jo json_object_t;
BEGIN
   jo := json_object_t();
   jo.put('access_token', '<access token>');
   DBMS_VECTOR.CREATE_CREDENTIAL(
          credential_name => 'COHERE_CRED',
          params => json(jo.to_string));
END;
/
```

- Vertex AI example:

```
begin
   dbms_vector_chain.drop_credential(credential_name  =>
'VERTEXAI_CRED');
exception
   when others then null;
end;
/

declare
   jo json_object_t;
begin
   jo := json_object_t();
   jo.put('access_token', '<access token>');
   dbms_vector_chain.create_credential(
          credential_name   =>  'VERTEXAI_CRED',
          params            => json(jo.to_string));
end;
/
```

8. Rerank the initial retrieval:

> **Note:**
>
> For a list of all supported REST endpoints, see Supported Third-Party Provider Operations and Endpoints.

- Using the Cohere `rerank-english-v3.0` model:

```
declare
  params clob;
  reranked_output json;
begin
  params := '
{
  "provider": "cohere",
  "credential_name": "COHERE_CRED",
  "url": "https://api.cohere.com/v1/rerank",
```

```
  "model": "rerank-english-v3.0",
  "return_documents": true,
  "top_n": 3
}';

  reranked_output := dbms_vector_chain.rerank(:query,
json(:initial_retrieval_docs), json(params));
  dbms_output.put_line(json_serialize(reranked_output));
end;
/
```

- Using the Vertex AI `semantic-ranker-512` model:

```
declare
  params clob;
  reranked_output json;
begin
  params := '
{
  "provider": "vertexai",
  "credential_name": "VERTEXAI_CRED",
  "url": "https://discoveryengine.googleapis.com/v1/projects/
1085581009881/locations/global/rankingConfigs/
default_ranking_config:rank",
  "model": "semantic-ranker-512@latest",
  "ignoreRecordDetailsInResponse": false,
  "topN": 3
}';

  reranked_output := dbms_vector_chain.rerank(:query,
json(:initial_retrieval_docs), json(params));
  dbms_output.put_line(json_serialize(reranked_output));
end;
/
```

Using the above Cohere model, the reranked results appear as follows:

```
[
   {
     "index" : "0",
     "score" : "0.059319142",
     "content" : "Jupiter boasts an impressive system of 95 known moons,
including the four largest Galilean satellites."
   },
   {
     "index" : "2",
     "score" : "0.04352814",
     "content" : "Io, one of Jupiter's Galilean moons, is the most
volcanically active body in our solar system."
   },
   {
     "index" : "4",
     "score" : "0.04138472",
     "content" : "Jupiter's composition is similar to that of the Sun, and it
could have become a brown dwarf if its mass had been 80 times greater."
```

```
    }
]
```

Related Topics

- RERANK
  Use the `DBMS_VECTOR_CHAIN.RERANK` function to reassess and reorder an initial set of results to retrieve more relevant search output.
- DBMS_VECTOR
  The `DBMS_VECTOR` package simplifies common operations with Oracle AI Vector Search, such as extracting chunks or embeddings from user data, generating text for a given prompt or an image, creating a vector index, or reporting on index accuracy.
- DBMS_VECTOR_CHAIN
  The `DBMS_VECTOR_CHAIN` package enables advanced operations with Oracle AI Vector Search, such as chunking and embedding data along with text generation and summarization capabilities. It is more suitable for text processing with similarity search and hybrid search, using functionality that can be pipelined together for an end-to-end search.

Supported Third-Party Provider Operations and Endpoints

Review a list of third-party REST providers and REST endpoints that are supported for various vector generation, summarization, text generation, and reranking operations.

- Public or Remote REST Endpoint Providers
- Local REST Endpoint Provider
- REST Operations
- REST Endpoints
- Models Supported for Generative AI

Public or Remote REST Endpoint Providers

The supported publicly-hosted, third-party REST endpoint providers are:

- Cohere
- Google AI
- Hugging Face
- Oracle Cloud Infrastructure (OCI) Generative AI
- OpenAI
- Vertex AI

Local REST Endpoint Provider

You can use Ollama as a third-party REST endpoint provider, locally and privately on your Linux, Windows, and macOS systems.

Ollama is a free and open-source command-line interface tool that allows you to run open LLMs (such as Llama 3, Phi 3, Mistral, and Gemma 2) and embedding models (such as mxbai-embed-large, nomic-embed-text, or all-minilm). You can access Ollama using SQL and PL/SQL commands.

You can download and run the Ollama application from https://ollama.com/download. You can either install Ollama as a service that runs in the background or as a standalone binary with a

manual install. For detailed installation-specific steps, see Quick Start in the Ollama Documentation.

REST Operations

These are the supported third-party REST operations and APIs along with their corresponding REST providers:

Operation	Provider	API
Generate embedding: Convert textual documents and images to one or more vector embeddings	• For text input: All supported public and local providers • For image input: Vertex AI	• DBMS_VECTOR.UTL_TO_EMBEDDING and DBMS_VECTOR.UTL_TO_EMBEDDINGS • DBMS_VECTOR_CHAIN.UTL_TO_EMBEDDING and DBMS_VECTOR_CHAIN.UTL_TO_EMBEDDINGS
Generate summary: Extract a brief and comprehensive summary from textual documents	All supported public and local providers	• DBMS_VECTOR_CHAIN.UTL_TO_SUMMARY
Generate text: Retrieve a descriptive response for textual prompts and images through conversations with LLMs	• For text input: All supported public and local providers • For image input: Google AI, Hugging Face, OpenAI, Ollama, and Vertex AI	• DBMS_VECTOR.UTL_TO_GENERATE_TEXT • DBMS_VECTOR_CHAIN.UTL_TO_GENERATE_TEXT
Rerank results: Reassess and reorder search results to retrieve a more relevant output	Cohere and Vertex AI	• DBMS_VECTOR.RERANK • DBMS_VECTOR_CHAIN.RERANK

REST Endpoints

These are the supported REST endpoints for all third-party REST providers:

API	Provider	Endpoint
`UTL_TO_EMBEDDING` and `UTL_TO_EMBEDDINGS`	Cohere	`https://api.cohere.ai/v1/embed`
	Generative AI	`https://inference.generativeai.us-chicago-1.oci.oraclecloud.com/20231130/actions/embedText`
	Google AI	`https://generativelanguage.googleapis.com/v1beta/models/`
	Hugging Face	`https://api-inference.huggingface.co/pipeline/feature-extraction/`
	Ollama	`http://localhost:11434/api/embeddings`
	OpenAI	`https://api.openai.com/v1/embeddings`
	Vertex AI	`https://LOCATION-aiplatform.googleapis.com/v1/projects/PROJECT/locations/LOCATION/publishers/google/models/`

API	Provider	Endpoint
UTL_TO_SUMMARY	Cohere	https://api.cohere.ai/v1/chat https://api.cohere.ai/v1/summarize
	Generative AI	https://inference.generativeai.us-chicago-1.oci.oraclecloud.com/20231130/actions/chat
	Google AI	https://generativelanguage.googleapis.com/v1beta/models/
	Hugging Face	https://api-inference.huggingface.co/models/
	Ollama	http://localhost:11434/api/generate
	OpenAI	https://api.openai.com/v1/chat/completions https://api.openai.com/v1/completions
	Vertex AI	https://LOCATION-aiplatform.googleapis.com/v1/projects/PROJECT/locations/LOCATION/publishers/google/models/
UTL_TO_GENERATE_TEXT	Cohere	https://api.cohere.ai/v1/chat https://api.cohere.ai/v1/generate
	Generative AI	https://inference.generativeai.us-chicago-1.oci.oraclecloud.com/20231130/actions/chat
	Google AI	https://generativelanguage.googleapis.com/v1beta/models/
	Hugging Face	https://api-inference.huggingface.co/models/
	Ollama	http://localhost:11434/api/generate
	OpenAI	https://api.openai.com/v1/chat/completions https://api.openai.com/v1/completions
	Vertex AI	https://LOCATION-aiplatform.googleapis.com/v1/projects/PROJECT/locations/LOCATION/publishers/google/models/
RERANK	Cohere	https://api.cohere.com/v1/rerank
	Vertex AI	https://discoveryengine.googleapis.com/v1/projects/PROJECT/locations/global/rankingConfigs/default_ranking_config:rank

Models Supported for Generative AI

These are the third-party models that are supported to use with Generative AI corresponding to each REST API:

API	Model
UTL_TO_EMBEDDING and UTL_TO_EMBEDDINGS	cohere.embed-english-v3.0 cohere.embed-multilingual-v3.0 cohere.embed-english-light-v3.0 cohere.embed-multilingual-light-v3.0

API	Model
UTL_TO_SUMMARY	cohere.command-r-16k cohere.command-r-plus meta.llama-3.1-70b-instruct meta.llama-3.1-405b-instruct
UTL_TO_GENERATE_TEXT	cohere.command-r-16k cohere.command-r-plus meta.llama-3.1-70b-instruct meta.llama-3.1-405b-instruct

10

Supported Clients and Languages

For more information about Oracle AI Vector Search support using some of Oracle's available clients and languages, see the included reference material.

Clients and Languages	Reference Material
PL/SQL	*Oracle Database PL/SQL Language Reference*
MLE JavaScript	*Oracle Database JavaScript Developer's Guide*
JDBC	*Oracle Database JDBC Developer's Guide*
Node.js	node-oracledb documentation
Python	python-oracledb documentation
Oracle Call Interface	*Oracle Call Interface Developer's Guide*
ODP.NET	*Oracle Data Provider for .NET Developer's Guide*
SQL*Plus	*SQL*Plus User's Guide and Reference*

Oracle Database 23ai supports binding with native VECTOR types for all Oracle clients. Applications that use earlier Oracle Client 23ai libraries can connect to Oracle Database 23ai in the following ways:

- Using the `TO_VECTOR()` SQL function to insert vector data, as shown in the following example:

```
INSERT INTO vecTab VALUES(TO_VECTOR('[1.1, 2.9, 3.14]'));
```

- Using the `FROM_VECTOR()` SQL function to fetch vector data, as shown in the following example:

```
SELECT FROM_VECTOR(dataVec) FROM vecTab;
```

11

Vector Diagnostics

AI Vector Search includes several views, statistics, and parameters that can be used to help understand how vector search is performing for your workload.

- Oracle AI Vector Search Views
  These are a set of data dictionary views related to Oracle AI Vector Search.
- Oracle AI Vector Search Statistics
  These are a set of statistics related to Oracle AI Vector Search.
- Oracle AI Vector Search Parameters
  This is a set of parameters related to Oracle AI Vector Search.

Oracle AI Vector Search Views

These are a set of data dictionary views related to Oracle AI Vector Search.

- Text Processing Views
  These views display language-specific data (abbreviation token details) and vocabulary data related to the Oracle AI Vector Search SQL and PL/SQL utilities.
- Vector Memory Pool Views
  Review the various vector memory pool views.
- Vector Index and Hybrid Vector Index Views
  These views allow you to query tables related to vector indexes and hybrid vector indexes.

Text Processing Views

These views display language-specific data (abbreviation token details) and vocabulary data related to the Oracle AI Vector Search SQL and PL/SQL utilities.

- ALL_VECTOR_ABBREV_TOKENS
  The `ALL_VECTOR_ABBREV_TOKENS` view displays a list of abbreviation tokens from all supported languages.
- ALL_VECTOR_LANG
  The `ALL_VECTOR_LANG` view displays a list of all supported languages, distributed by default.
- DBA_VECTOR_HITCOUNTS
  The `DBA_VECTOR_HITCOUNTS` view tracks calls to third parties for observability.
- USER_VECTOR_ABBREV_TOKENS
  The `USER_VECTOR_ABBREV_TOKENS` view displays a list of abbreviation tokens from all languages loaded by the current user.
- USER_VECTOR_HITCOUNTS
  The `USER_VECTOR_HITCOUNTS` view tracks calls to third parties for observability for the current user.
- USER_VECTOR_LANG
  The `USER_VECTOR_LANG` view displays all languages loaded by the current user.

- USER_VECTOR_VOCAB
  The `USER_VECTOR_VOCAB` view displays all custom token vocabularies created by the current user.
- USER_VECTOR_VOCAB_TOKENS
  The `USER_VECTOR_VOCAB_TOKENS` view displays tokens from all custom token vocabularies created by the current user.
- ALL_VECTOR_VOCAB
  The `ALL_VECTOR_VOCAB` view displays all custom token vocabularies.
- ALL_VECTOR_VOCAB_TOKENS
  The `ALL_VECTOR_VOCAB_TOKENS` view displays tokens from all custom token vocabularies.

Related Topics

- Configure Chunking Parameters
  Oracle AI Vector Search provides many parameters for chunking text data, such as `SPLIT [BY]`, `OVERLAP`, or `NORMALIZE`. In these examples, you can see how to configure these parameters to define your own chunking specifications and strategies, so that you can create meaningful chunks.

ALL_VECTOR_ABBREV_TOKENS

The `ALL_VECTOR_ABBREV_TOKENS` view displays a list of abbreviation tokens from all supported languages.

Column Name	Data Type	Description
`ABBREV_OWNER`	`VARCHAR2(128)`	Owner of the abbreviation token (for example, `PUBLIC`)
`ABBREV_LANGUAGE`	`NUMBER`	Language ID for the language (for example, `1` for American)
`ABBREV_TOKEN`	`NVARCHAR2(255)`	List of all abbreviation tokens corresponding to each language

ALL_VECTOR_LANG

The `ALL_VECTOR_LANG` view displays a list of all supported languages, distributed by default.

Column Name	Data Type	Description
`LANG_OWNER`	`VARCHAR2(128)`	Owner of the language (for example, `PUBLIC`)
`LANG_LANG`	`NUMBER`	Language ID for the language (for example, `1` for American)
`LANG_NAME`	`VARCHAR2(128)`	Name of the language (for example, `AMERICAN`)

DBA_VECTOR_HITCOUNTS

The DBA_VECTOR_HITCOUNTS view tracks calls to third parties for observability.

Column Name	Data Type	Description
USER_ID	NUMBER	ID of the current user
CREDENTIAL_ID	NUMBER	Credential ID for the third party call used by the user
PROVIDER_NAME	VARCHAR2(128)	Name of the third-party provider
METHOD_NAME	VARCHAR2(128)	Name of the method calling the third-party API(s) (such as UTL_TO_SUMMARY, UTL_TO_EMBEDDING, and so on)
TOTAL_COUNT	NUMBER	Total number of calls made to the third-party provider
LAST_HIT	TIMESTAMP(6)	Time of the last call to the third-party provider

USER_VECTOR_ABBREV_TOKENS

The USER_VECTOR_ABBREV_TOKENS view displays a list of abbreviation tokens from all languages loaded by the current user.

Column Name	Data Type	Description
ABBREV_LANGUAGE	NUMBER	Language ID for the language (for example, 1 for American)
ABBREV_TOKEN	NVARCHAR2(255)	List of all abbreviation tokens corresponding to each language

USER_VECTOR_HITCOUNTS

The USER_VECTOR_HITCOUNTS view tracks calls to third parties for observability for the current user.

Column Name	Data Type	Description
USER_ID	NUMBER	ID of the current user
CREDENTIAL_ID	NUMBER	Credential ID for the third-party call used by the user
CREDENTIAL_NAME	VARCHAR2(128)	Credential name for the third-party call used by the user
PROVIDER_NAME	VARCHAR2(128)	Name of the third-party provider
METHOD_NAME	VARCHAR(128)	Name of the method calling the third-party API(s) (such as UTL_TO_SUMMARY, UTL_TO_EMBEDDING, and so on)
TOTAL_COUNT	NUMBER	Total number of calls made to the third-party provider.

Column Name	Data Type	Description
LAST_HIT	TIMESTAMP(6)	Time of the last call to the third-party provider

USER_VECTOR_LANG

The USER_VECTOR_LANG view displays all languages loaded by the current user.

Column Name	Data Type	Description
LANG_LANG	NUMBER	Language ID for the language (for example, 1 for American)
LANG_NAME	VARCHAR2(128)	Name of the language (for example, AMERICAN)

USER_VECTOR_VOCAB

The USER_VECTOR_VOCAB view displays all custom token vocabularies created by the current user.

Column Name	Data Type	Description
VOCAB_NAME	VARCHAR2(128)	User-specified name of the vocabulary (for example, DOC_VOCAB)
FORMAT	VARCHAR2(4)	Format of the vocabulary, such as XLM, BERT, or GPT2
CASED	VARCHAR2(7)	Character-casing of the vocabulary, that is, vocabulary to be treated as cased or uncased

USER_VECTOR_VOCAB_TOKENS

The USER_VECTOR_VOCAB_TOKENS view displays tokens from all custom token vocabularies created by the current user.

Column Name	Data Type	Description
VOCAB_NAME	VARCHAR2(128)	User-specified name of the vocabulary (for example, DOC_VOCAB)
VOCAB_TOKEN	VARCHAR2(255)	Tokens contained in the vocabulary

ALL_VECTOR_VOCAB

The ALL_VECTOR_VOCAB view displays all custom token vocabularies.

Column Name	Data Type	Description
VOCAB_OWNER	VARCHAR2(128)	Owner of the vocabulary (for example, SYS)

Column Name	Data Type	Description
VOCAB_NAME	VARCHAR2(128)	User-specified name of the vocabulary (for example, DOC_VOCAB)
FORMAT	VARCHAR2(4)	Format of the vocabulary, such as XLM, BERT, or GPT2
CASED	VARCHAR2(7)	Character-casing of the vocabulary, that is, vocabulary to be treated as cased or uncased

ALL_VECTOR_VOCAB_TOKENS

The ALL_VECTOR_VOCAB_TOKENS view displays tokens from all custom token vocabularies.

Column Name	Data Type	Description
VOCAB_OWNER	VARCHAR2(128)	Owner of the vocabulary (for example, SYS)
VOCAB_NAME	VARCHAR2(128)	User-specified name of the vocabulary (for example, DOC_VOCAB)
VOCAB_TOKEN	VARCHAR2(255)	Tokens contained in the vocabulary

Vector Memory Pool Views

Review the various vector memory pool views.

- V$VECTOR_MEMORY_POOL
  This view contains information about the space allocation for Vector Memory.

V$VECTOR_MEMORY_POOL

This view contains information about the space allocation for Vector Memory.

The Vector Memory Pool area is used primarily to maintain in-memory vector indexes or metadata useful for vector-related operations. The Vector Memory Pool is subdivided into two pools: a 1MB pool used to store In-Memory Neighbor Graph Index allocations; and a 64K pool used to store metadata. The amount of available memory in each pool is visible in the V$VECTOR_MEMORY_POOL view. The relative size of the two pools is determined by internal heuristics. The size of the Vector Memory Pool is controlled by the vector_memory_size parameter. Area in the Vector Memory Pool is also allocated to accelerate Neighbor Partition Index access by storing centroid vectors.

Column Name	Data Type	Description
POOL	VARCHAR2(26)	Name of the pools in the Vector Memory Pool (64K or 1MB)
ALLOC_BYTES	NUMBER	Total amount of memory allocated to this pool
USED_BYTES	NUMBER	Amount of memory currently used in this pool

Column Name	Data Type	Description
POPULATE_STATUS	VARCHAR2(26)	Status of the vector memory store—whether it is being populated, is done populating etc.
CON_ID	NUMBER	The ID of the container to which the data pertains. Possible values are: • 0: This value is used for rows containing data that pertain to the entire multitenant container database (CDB). This value is also used for rows in non-CDBs. • 1: This value is used for rows containing data that pertain to only the root. • n: Where n is the applicable container ID for the rows containing data.

Example

```
select CON_ID, POOL, ALLOC_BYTES/1024/1024 as ALLOC_BYTES_MB,
USED_BYTES/1024/1024 as USED_BYTES_MB
from V$VECTOR_MEMORY_POOL order by 1,2;


    CON_ID POOL            ALLOC_BYTES_MB USED_BYTES_MB
---------- --------------- -------------- -------------
         1 1MB POOL                   319             0
         1 64KB POOL                  144             0
         1 IM POOL METADATA            32            32
         2 1MB POOL                   320             0
         2 64KB POOL                  144             0
         2 IM POOL METADATA            16            16
         3 1MB POOL                   320             0
         3 64KB POOL                  144             0
         3 IM POOL METADATA            16            16
         4 1MB POOL                   320             0
         4 64KB POOL                  144             0
         4 IM POOL METADATA            16            16

12 rows selected.

SQL>
```

Vector Index and Hybrid Vector Index Views

These views allow you to query tables related to vector indexes and hybrid vector indexes.

- VECSYS.VECTOR$INDEX
  This dictionary table contains detailed information about vector indexes.

- V$VECTOR_INDEX
  This fixed view provides diagnostic information about vector indexes and is available with Autonomous Database Serverless (ADB-S).
- V$VECTOR_GRAPH_INDEX
  This fixed view provides diagnostic information about In-Memory Neighbor Graph vector indexes and is available with Autonomous Database Serverless (ADB-S).
- V$VECTOR_PARTITIONS_INDEX
  This fixed view provides diagnostic information about Inverted File Flat indexes and is available with Autonomous Database Serverless (ADB-S).
- VECSYS.VECTOR$INDEX$CHECKPOINTS
  This dictionary table provides detailed information about Hierarchical Navigable Small World (HNSW) full checkpoints at the database level.
- <index name>$VECTORS
  This dictionary table provides information about the vector index part of a hybrid vector index, showing the contents of the `$VR` table with row ids, chunks, and embeddings.

VECSYS.VECTOR$INDEX

This dictionary table contains detailed information about vector indexes.

Column Name	Data Type	Description
IDX_OBJN	NUMBER	Object number of the vector index
IDX_OBJD	NUMBER	ID of the vector index object. This ID can be used to rebuild the vector index.
IDX_OWNER#	NUMBER	Owner ID of the vector index. Refer user$ entry
IDX_NAME	VARCHAR2(128)	Name of the vector index.
IDX_BASE_TABLE_OBJN	NUMBER	Base table object number
IDX_PARAMS	JSON	Vector index creation parameters such as vector column indexed, index distance, vector dimension datatype, number of dimensions, efConstruction, and number of neighbors for in-memory neighbor graph HNSW index or the number of centroids for Inverted File Flat vector indexes.
IDX_AUXILIARY_TABLES	JSON	Names and object IDs of auxiliary tables used to support rowid-to-vertexid conversion information or names of a centroid table and its associated partitions table for Inverted File Flat vector indexes.

Example

```
SQL> SELECT JSON_SERIALIZE(IDX_PARAMS returning varchar2 PRETTY)
  2* FROM VECSYS.VECTOR$INDEX where IDX_NAME = 'DOCS_HNSW_IDX';

JSON_SERIALIZE(IDX_PARAMSRETURNINGVARCHAR2PRETTY)
{
```

```
  "type" : "HNSW",
  "num_neighbors" : 32,
  "efConstruction" : 300,
  "upcast_dtype" : 1,
  "distance" : "COSINE",
  "accuracy" : 95,
  "vector_type" : "FLOAT32",
  "vector_dimension" : 384,
  "degree_of_parallelism" : 1,
  "indexed_col" : "EMBED_VECTOR"
}
SQL>
SQL> select * from vecsys.vector$index;

IDX_OBJN IDX_OBJD  IDX_OWNER# IDX_NAME       IDX_BASE_TABLE_OBJN
IDX_PARAMS
IDX_AUXILIARY_TABLES
                          IDX_SPARE1 IDX_SPARE2
-------- --------  ---------- -------------- -------------------
-------------------------
--------------------
                          ----------  ---------
   74051     143               DOCS_HNSW_IDX              73497
{"type":"HNSW",                         {"rowid_vid_map_objn":74052,

"num_neighbors":32,
"shared_journal_transaction_commits_objn":74054,

"efConstruction":300,
"shared_journal_change_log_objn":74057,

"upcast_dtype":1,
"rowid_vid_map_name":"VECTOR$DOCS_HNSW_IDX$HNSW_ROWID_VID_MAP",

"distance":"COSINE",
"shared_journal_transaction_commits_name":"VECTOR$DOCS_HNSW_IDX$HNSW_SHARED_JO
URNAL_TRANSACTION_COMMITS",

"accuracy":95,
"shared_journal_change_log_name":"VECTOR$DOCS_HNSW_IDX$HNSW_SHARED_JOURNAL_CHA
NGE_LOG"}

"vector_type":"FLOAT32",

"vector_dimension":384,

"degree_of_parallelism":1,

"indexed_col":"EMBED_VECTOR"}

   74072      143             GALAXIES_HNSW_IDX           74069
{"type":"HNSW",                           {"rowid_vid_map_objn":74073,

"num_neighbors":32,
"shared_journal_transaction_commits_objn":74075,
```

```
"efConstruction":300,
"shared_journal_change_log_objn":74078,

"upcast_dtype":0,
"rowid_vid_map_name":"VECTOR$GALAXIES_HNSW_IDX$HNSW_ROWID_VID_MAP",

"distance":"COSINE",
"shared_journal_transaction_commits_name":"VECTOR$GALAXIES_HNSW_IDX$HNSW_SHARE
D_JOURNAL_TRANSACTION_COMMITS",

"accuracy":95,
"shared_journal_change_log_name":"VECTOR$GALAXIES_HNSW_IDX$HNSW_SHARED_JOURNAL
_CHANGE_LOG"}

"vector_type":"INT8",

"vector_dimension":5,

"degree_of_parallelism":1,

"indexed_col":"EMBEDDING"}


SQL>
```

V$VECTOR_INDEX

This fixed view provides diagnostic information about vector indexes and is available with Autonomous Database Serverless (ADB-S).

Column Name	Data Type	Description
OWNER	VARCHAR(129)	User name of the vector index owner
INDEX_NAME	VARCHAR(129)	Name of the vector index
PARTITION_NAME	VARCHAR(129)	Object partition name (set to NULL for non-partitioned objects)
INDEX_ORGANIZATION	VARCHAR(129)	The vector index organization. The value can be one of the following: • NEIGHBOR PARTITIONS • INMEMORY NEIGHBOR GRAPH
INDEX_OBJN	NUMBER	Index object number
ALLOCATED_BYTES	NUMBER	Total amount of memory allocated to this vector index, in bytes. When the vector index is an IVF index, the value is 0.
USED_BYTES	NUMBER	Total amount of memory currently used by the vector index, in bytes. When the vector index is an IVF index, the value is 0.

Column Name	Data Type	Description
NUM_VECTORS	NUMBER	The number of vectors currently indexed in the main-memory storage of the index as of the latest snapshot
NUM_REPOP	NUMBER	Number of times this index has been repopulated
INDEX_USED_COUNT	NUMBER	Number of times this vector index has been used by queries
DISTANCE_TYPE	VARCHAR2(129)	Possible distance values: • EUCLIDEAN • EUCLIDEAN_SQUARED • COSINE • DOT • MANHATTAN • HAMMING
INDEX_DIMENSIONS	NUMBER	Number of dimensions of the indexed vector
INDEX_DIM_TYPE	VARCHAR2(129)	Type of the dimensions of the index vector. Possible values: • FLOAT16 • FLOAT32 • FLOAT64 • INT8
DEFAULT_ACCURACY	NUMBER	The accuracy to achieve when performing approximate search on this vector index if a target query accuracy is not provided
CON_ID	NUMBER	The ID of the container to which the data pertains. Possible values include the following: • 0: This value is used for rows containing data that pertain to the entire multitenant container database (CDB). This value is also used for rows in non-CDBs. • 1: This value is used for rows containing data that pertain to only the root. • n: Where n is the applicable container ID for the rows containing data.

V$VECTOR_GRAPH_INDEX

This fixed view provides diagnostic information about In-Memory Neighbor Graph vector indexes and is available with Autonomous Database Serverless (ADB-S).

Column Name	Data Type	Description
OWNER	VARCHAR2(129)	User name of the vector graph index owner
INDEX_NAME	VARCHAR2(129)	Name of the vector graph index
PARTITION_NAME	VARCHAR2(129)	Object partition name (set to NULL for non-partitioned objects)
INDEX_OBJN	NUMBER	Index object number
ANCHOR_ADDRESS	RAW(8)	The address of the anchor structure of the in-memory neighbor graph index in the vector pool
INDEX_GRAPH_TYPE	VARCHAR2(129)	Type of the constructed graph of this index. Possible values: • HNSW: Hierarchical Navigable Small Worlds. Currently only this graph is supported.
NUM_LAYERS	NUMBER	Number of layers in the constructed graph
NUM_VECTORS	NUMBER	Number of indexed vectors in the vector graph index
SPARSE_LAYER_VECTORS	NUMBER	The total number of vectors in the sparse layers. Note that a sparse layer is defined as any layer above the bottom most layer.
NUM_NEIGHBORS	NUMBER	Maximum number of neighbors a vector can have in the sparse layers
EFCONSTRUCTION	NUMBER	Maximum number of closest vector candidates considered at each step of the search during insertion
TOTAL_EDGES	NUMBER	Number of total edges in the constructed graph as of the latest snapshot
REF_COUNT	NUMBER	Number used to track readers of the inmemory neighbor graph index. The inmemory neighbor graph index can only be dropped once the refcount is 0.
QUERY_DIST_COUNT	NUMBER	Number of distance computations done as part of queries that used the inmemory neighbor graph index

Column Name	Data Type	Description
CREATION_DIST_COUNT	NUMBER	Number of distance computations done as part of the inmemory neighbor graph index creation
PRUNED_NEIGHBORS	NUMBER	Number of neighbors pruned out during the neighbor selection phase of the inmemory neighbor graph index creation
NUM_SNAPSHOTS	NUMBER	Number of snapshots currently tracked for the inmemory neighbor graph index
MAX_SNAPSHOT	NUMBER	The total number of snapshots created so far for the inmemory graph index
ALLOCATED_BYTES	NUMBER	Total amount of memory allocated to this vector graph index, in bytes
USED_BYTES	NUMBER	Total amount of memory currently used by the vector graph index, in bytes
CON_ID	NUMBER	The ID of the container to which the data pertains. Possible values include the following: • 0: This value is used for rows containing data that pertain to the entire multitenant container database (CDB). This value is also used for rows in non-CDBs. • 1: This value is used for rows containing data that pertain to only the root • n: Where n is the applicable container ID for the rows containing data

V$VECTOR_PARTITIONS_INDEX

This fixed view provides diagnostic information about Inverted File Flat indexes and is available with Autonomous Database Serverless (ADB-S).

Column Name	Data Type	Description
OWNER	VARCHAR2(129)	User name of the vector partition index owner
INDEX_NAME	VARCHAR2(129)	Name of the vector partition index
PARTITION_NAME	VARCHAR2(129)	Object partition name (set to NULL for non-partitioned objects)
INDEX_OBJN	NUMBER	Index object number

Column Name	Data Type	Description
CENTROIDS_ANCHOR_ADDRESS	RAW(8)	The address of the anchor structure hosting the centroids for the in-memory neighbor partitions index in the vector pool. CENTROIDS_ANCHOR_ADDRESS is only enabled if in-memory centroids are enabled on the index. Set to 0 if in-memory centroids are unavailable for the index.
INDEX_PARTITIONS_TYPE	VARCHAR2(129)	The partition type of this neighbor partitions vector index. Possible values include: • IVF: Inverted File Index
NUM_NEIGHBOR_PARTITIONS	NUMBER	Number of neighbor partitions created for the neighbor partitions vector index
NUM_VECTORS	NUMBER	Number of indexed vectors in this vector index
MIN_PARTITION_VECTORS_COUNT	NUMBER	Number of vectors in the smallest neighbor partition
MAX_PARTITION_VECTORS_COUNT	NUMBER	Number of vectors in the largest neighbor partition
INDEX_USED_COUNT	NUMBER	Number of times this vector partition index has been used by queries
AVG_PARTITIONS_SCANNED	NUMBER	Average number of neighbor partitions scanned in order to answer queries. This is an average over the NEIGHBOR PROBES query parameter supported for an Inverted File neighbor partitions index
CON_ID	NUMBER	The ID of the container to which the data pertains. Possible values include the following: • 0: This value is used for rows containing data that pertain to the entire multitenant container database (CDB). This value is also used for rows in non-CDBs. • 1: This value is used for rows containing data that pertain to only the root • n: Where n is the applicable container ID for the rows containing data

VECSYS.VECTOR$INDEX$CHECKPOINTS

This dictionary table provides detailed information about Hierarchical Navigable Small World (HNSW) full checkpoints at the database level.

Column Name	Data Type	Description
INDEX_OBJN	NUMBER	The index object number to uniquely identify the HNSW index.
INDEX_OWNER_ID	NUMBER	The owner id for the vector index.
CHECKPOINT_ID	NUMBER	A monotonically increasing ID tracking various checkpoints for this HNSW index.
CHECKPOINT_SCN	NUMBER	The SCN as of which the HNSW checkpoint is taken.
CHECKPOINT_TYPE	NUMBER	Full Checkpoint is the only checkpoint type supported.
VERSION_NUMBER	NUMBER	The checkpoint format may change in between releases. Thus, you can use a version number to decide whether to reload the HNSW index using a particular checkpoint.
TABLESPACE_NUMBER	NUMBER	The tablespace number. You can view a tablespace number to decide whether to store full checkpoints of your HNSW graphs in the same tablespace or in a different one.

Related Topics

- Understand HNSW Index Population Mechanisms in Oracle RAC or Single Instance
  Learn how Hierarchical Navigable Small World (HNSW) indexes are populated during index creation, index repopulation, or instance startup in an Oracle Real Application Clusters (Oracle RAC) or a non-RAC environment.

<index name>$VECTORS

This dictionary table provides information about the vector index part of a hybrid vector index, showing the contents of the `$VR` table with row ids, chunks, and embeddings.

> **Note:**
>
> The *<index name>*`$VECTORS` view resides in a user's schema. Therefore, if a hybrid vector index is named `IDX`, then the view name is `IDX$VECTORS`.
> *<index name>*`$VECTORS` view is not supported for Local Hybrid Vector Indexes.

Column Name	Data Type	Description
DOC_ROWID	ROWID	Document table's row ID, excluding lazy deletes
DOC_CHUNK_ID	NUMBER	ID for each chunk text
DOC_CHUNK_COUNT	NUMBER	Number of chunks into which each document is split
DOC_CHUNK_OFFSET	NUMBER	Original position of each chunk in the source document, relative to the start of document (which has a position of 1)
DOC_CHUNK_LENGTH	NUMBER	Character length of each chunk text
DOC_CHUNK_TEXT	VARCHAR2(4000)	Human-readable content in each chunk
DOC_CHUNK_EMBEDDING	VECTOR(*, *)	Generated vector embedding for each chunk

Related Topics

- Manage Hybrid Vector Indexes
  Learn how to manage a hybrid vector index, which is a single index for searching by *similarity* and *keywords*, to enhance the accuracy of your search results.

Oracle AI Vector Search Statistics

These are a set of statistics related to Oracle AI Vector Search.

- Oracle AI Vector Search Dictionary Statistics
  A set of dictionary statistics related to Oracle AI Vector Search.
- Oracle Machine Learning Static Dictionary Views
  Lists data dictionary views related to Oracle Machine Learning models.

Oracle AI Vector Search Dictionary Statistics

A set of dictionary statistics related to Oracle AI Vector Search.

- **Vector simd dist single calls**: The number of vector distance function calls invoked by the user, where both the inputs are a single vector.
- **Vector simd dist point calls**: The number of vector distance function calls invoked by the user, where one input is a single vector, and the other input is an array of vectors.
- **Vector simd dist array calls**: The number of vector distance function calls invoked by the user, where both the inputs are an array of vectors.
- **Vector simd dist flex calls**: The number of vector distance function calls invoked by the user, where at least one input is with FLEX vector data type (no dimension or storage data type specified).
- **Vector simd dist total rows:** The number of rows processed by the vector distance function.
- **Vector simd dist flex rows**: The number of rows processed by the vector distance function with FLEX vector data type.

- **Vector simd topK single calls**: The number of `topK` distance function calls invoked by the user, where both the inputs are a single vector.
- **Vector simd topK point calls**: The number of `topK` distance function calls invoked by the user, where one input is a single vector and the other input is an array of vectors.
- **Vector simd topK array calls**: The number of `topK` distance function calls invoked by the user, where both the inputs are an array of vectors.
- **Vector simd topK flex calls**: The number of `topK` distance function calls invoked by the user, where at least one input is with FLEX vector data type (no dimension or storage data type specified).
- **Vector simd topK total rows**: The number of rows processed by the `topK` distance function.
- **Vector simd topK flex rows**: The number of rows processed by the `topK` distance function with FLEX vector data type.
- **Vector simd topK selected total rows**: The number of distance results returned by the `topK` vector distance function.

> **Note:**
>
> This is typically `sum(K)` for all `topK` queries.

- **Vector simd construction num of total calls**: The number of vector construction function calls invoked by the user.
- **Vector simd construction total result rows**: The number of vector rows returned by the vector construction function.
- **Vector simd construction num of ASCII calls**: The number of vector construction function calls invoked by the user, where the input encoding is ASCII.
- **Vector simd construction num of flex calls**: The number of vector construction function calls invoked by the user, where the output vector is of FLEX vector data type.
- **Vector simd construction result rows for flex**: The number of vector rows returned by the vector construction function, where the output vector is of FLEX vector data type.
- **Vector simd vector conversion num of total calls**: The number of vector conversion function calls invoked by the user.
- **VECTOR NEIGHBOR GRAPH HNSW build HT Element Allocated:** The number of hash table elements allocated for vector indexes having organization `Inmemory Neighbor Graph` and type `HNSW` that were created by a user.
- **VECTOR NEIGHBOR GRAPH HNSW build HT Element Freed:** The number of hash table elements freed for vector indexes having organization `Inmemory Neighbor Graph` and type `HNSW` that were created by a user.
- **VECTOR NEIGHBOR GRAPH HNSW reload attempted:** The number of reload operations attempted in the background for vector indexes having organization `Inmemory Neighbor Graph` and type `HNSW`.
- **VECTOR NEIGHBOR GRAPH HNSW reload successful:** The number of reload operations completed in the background for vector indexes having organization `Inmemory Neighbor Graph` and type `HNSW`.
- **VECTOR NEIGHBOR GRAPH HNSW build computed layers:** The total number of layers created in an HNSW index.

- **VECTOR NEIGHBOR GRAPH HNSW build indexed vectors:** The total number of vector indexes in the HNSW index.
- **VECTOR NEIGHBOR GRAPH HNSW build computed distances:** The total number of distance computations executed during the HNSW index build phase.
- **VECTOR NEIGHBOR GRAPH HNSW build sparse layers computed distances:** The total number of distance computations executed during the HNSW index build phase, in all the layers, except the bottom one.
- **VECTOR NEIGHBOR GRAPH HNSW build dense layer computed distances:** The total number of distance computations executed during the HNSW index phase in the bottom layer.
- **VECTOR NEIGHBOR GRAPH HNSW build prune operation computed distances:** The total number of distance computations that were executed during the pruning operations required to find the closest neighbors for a vector in the HNSW index build phase.
- **VECTOR NEIGHBOR GRAPH HNSW build pruned neighbor lists:** The total number of neighbors pruned during the HNSW index build phase.
- **VECTOR NEIGHBOR GRAPH HNSW build pruned neighbors:** The total number of neighbors pruned during the HNSW index build phase.
- **VECTOR NEIGHBOR GRAPH HNSW build created edges:** The total number of edges created in an HNSW index.
- **VECTOR NEIGHBOR GRAPH HNSW search executed approximate:** The total number of query searches executed using approximate search in an HNSW index.
- **VECTOR NEIGHBOR GRAPH HNSW search executed exhaustive:** The total number of query searches executed using exact search in an HNSW index.
- **VECTOR NEIGHBOR GRAPH HNSW search computed distances dense layer:** The total number of distance computations executed in the bottom layer of an HNSW index during query searches.
- **VECTOR NEIGHBOR GRAPH HNSW search computed distances sparse layer:** The total number of distance computations executed in all the layers except the bottom layer of an HNSW index during query searches.
- **VECTOR NEIGHBOR PARTITIONS IVF build HT Element Allocated:** The number of hash table elements allocated for vector indexes having organization `Neighbor Partitions` and type `IVF` that were created by a user.
- **VECTOR NEIGHBOR PARTITIONS IVF build HT Element Freed:** The number of hash table elements freed for vector indexes having organization `Neighbor Partitions` and type `IVF` that were created by a user.
- **VECTOR NEIGHBOR PARTITIONS IVF background Population Started:** The number of in-memory centroids background population operations started for vector indexes having organization `Neighbor Partitions` and type `IVF`.
- **VECTOR NEIGHBOR PARTITIONS IVF background Population Succeeded:** The number of in-memory centroids background population operations completed for vector indexes having organization `Neighbor Partitions` and type `IVF`.
- **VECTOR NEIGHBOR PARTITIONS IVF background Cleanup Started:** The number of in-memory centroids background cleanup operations started for vector indexes having organization `Neighbor Partitions` and type `IVF`.
- **VECTOR NEIGHBOR PARTITIONS IVF XIC Population Started:** The number of in-memory centroids cross-instance population operations started for vector indexes having organization `Neighbor Partitions` and type `IVF`.

- **VECTOR NEIGHBOR PARTITIONS IVF XIC Population Succeeded:** The number of in-memory centroids cross-instance population operations completed for vector indexes having organization `Neighbor Partitions` and type `IVF`.
- **VECTOR NEIGHBOR PARTITIONS IVF XIC Cleanup Started:** The number of in-memory centroids cross-instance cleanup operations started for vector indexes having organization `Neighbor Partitions` and type `IVF`.
- **VECTOR NEIGHBOR PARTITIONS IVF XIC Cleanup Succeeded:** The number of in-memory centroids cross-instance cleanup operations completed for vector indexes having organization `Neighbor Partitions` and type `IVF`.
- **VECTOR NEIGHBOR PARTITIONS IVF im centroids in scan:** The number of times that in-memory centroids are used in scan for vector indexes having organization `Neighbor Partitions` and type `IVF`.
- **VECTOR NEIGHBOR PARTITIONS IVF im centroids in dmls:** The number of times that in-memory centroids are used in the DML that happens on the base table with vector indexes having organization `Neighbor Partitions` and type `IVF`.

Oracle Machine Learning Static Dictionary Views

Lists data dictionary views related to Oracle Machine Learning models.

Query the following views to learn more about the machine learning models:

- ALL_MINING_MODEL_ATTRIBUTES
- ALL_MINING_MODELS

Oracle AI Vector Search Parameters

This is a set of parameters related to Oracle AI Vector Search.

- **VECTOR_MEMORY_SIZE**

  Syntax: `VECTOR_MEMORY_SIZE = integer[K | M | G]`

  The initialization parameter `VECTOR_MEMORY_SIZE` specifies either the current size of the Vector Pool (at CDB level) or the maximum Vector Pool usage allowed by a PDB (at PDB level).

  For more information about this parameter, see Size the Vector Pool.

- **VECTOR_INDEX_NEIGHBOR_GRAPH_RELOAD**

  Syntax: `VECTOR_INDEX_NEIGHBOR_GRAPH_RELOAD = [RESTART | OFF]` (default `RESTART`)

  The initialization parameter `VECTOR_INDEX_NEIGHBOR_GRAPH_RELOAD` is used to manage automatic recreation of HNSW indexes. You can enable or disable both the HNSW duplication and reload mechanisms in Oracle RAC and non-RAC environments.

  For more information about this parameter and the duplication and reload mechanisms, see Understand HNSW Index Population Mechanisms in Oracle RAC or Single Instance.

- **VECTOR_QUERY_CAPTURE**

  Syntax: `VECTOR_QUERY_CAPTURE = [ON | OFF]` (default `ON`)

  The initialization parameter `VECTOR_QUERY_CAPTURE` is used to enable or disable capture of query vectors.

For more information about this parameter and about capturing query vectors, see Index Accuracy Report.

- **COMPATIBLE**

  Syntax: `COMPATIBLE = release_number`

  The initialization parameter enables you to use a new release of Oracle while ensuring the ability to downgrade the database to an earlier release. To use the `VECTOR` data type and its related features, `COMPATIBLE` must be set to 23.4.0 or higher.

  Certain vector features require the `COMPATIBLE` parameter to be manually updated to a higher value in order to be available for use. To find requirements for a particular feature, see the applicable Release Update page at Oracle Database 23ai Release Updates.

  For more information about this parameter, see *Oracle Database Reference*.

12

Vector Search PL/SQL Packages

The DBMS_VECTOR, DBMS_VECTOR_CHAIN, and DBMS_HYBRID_VECTOR PL/SQL APIs are available to support Oracle AI Vector Search capabilities.

- DBMS_VECTOR
  The DBMS_VECTOR package simplifies common operations with Oracle AI Vector Search, such as extracting chunks or embeddings from user data, generating text for a given prompt or an image, creating a vector index, or reporting on index accuracy.
- DBMS_VECTOR_CHAIN
  The DBMS_VECTOR_CHAIN package enables advanced operations with Oracle AI Vector Search, such as chunking and embedding data along with text generation and summarization capabilities. It is more suitable for text processing with similarity search and hybrid search, using functionality that can be pipelined together for an end-to-end search.
- DBMS_HYBRID_VECTOR
  The DBMS_HYBRID_VECTOR package contains a JSON-based query API SEARCH, which lets you query against hybrid vector indexes.

DBMS_VECTOR

The DBMS_VECTOR package simplifies common operations with Oracle AI Vector Search, such as extracting chunks or embeddings from user data, generating text for a given prompt or an image, creating a vector index, or reporting on index accuracy.

This table lists the DBMS_VECTOR subprograms and briefly describes them.

Table 12-1 DBMS_VECTOR Package Subprograms

Subprogram	Description
ONNX Model Related Procedures:	
These procedures enable you to load an ONNX model into Oracle Database and drop the ONNX model.	
LOAD_ONNX_MODEL	Loads an ONNX model into the database
LOAD_ONNX_MODEL_CLOUD	Loads an ONNX model from object storage into the database
DROP_ONNX_MODEL Procedure	Drops the ONNX model
Chainable Utility (UTL) Functions:	
These functions are a set of modular and flexible functions within vector utility PL/SQL packages. You can chain these together to automate end-to-end data transformation and similarity search operations.	
UTL_TO_CHUNKS	Splits data into smaller pieces or chunks
UTL_TO_EMBEDDING and UTL_TO_EMBEDDINGS	Converts text or an image to one or more vector embeddings
UTL_TO_GENERATE_TEXT	Generates text for a prompt (input string) or an image
Credential Helper Procedures:	
These procedures enable you to securely manage authentication credentials in the database. You require these credentials to enable access to third-party service providers for making REST calls.	
CREATE_CREDENTIAL	Creates a credential name

Table 12-1 (Cont.) DBMS_VECTOR Package Subprograms

Subprogram	Description
DROP_CREDENTIAL	Drops an existing credential name
Data Access Functions: These functions enable you to retrieve data, create index, and perform simple similarity search operations.	
CREATE_INDEX	Creates a vector index
REBUILD_INDEX	Rebuilds a vector index
GET_INDEX_STATUS	Describes the status of a vector index creation
ENABLE_CHECKPOINT	Enables the Checkpoint feature for a vector index user and index name
DISABLE_CHECKPOINT	Disables the Checkpoint feature for a vector index user and index name
INDEX_VECTOR_MEMORY_ADVISOR	Determines the vector memory size that is needed for a vector index
QUERY	Performs a similarity search query
RERANK	Reorders search results for a more relevant output
Accuracy Reporting Function: These functions enable you to determine the accuracy of existing search indexes and to capture accuracy values achieved by approximate searches performed by past workloads.	
INDEX_ACCURACY_QUERY	Verifies the accuracy of a vector index
INDEX_ACCURACY_REPORT	Captures accuracy values achieved by approximate searches

> **Note:**
>
> `DBMS_VECTOR` is a lightweight package that does not support text processing or summarization operations. Therefore, the `UTL_TO_TEXT` and `UTL_TO_SUMMARY` chainable utility functions and all the chunker helper procedures are available only in the advanced `DBMS_VECTOR_CHAIN` package.

- CREATE_CREDENTIAL
  Use the `DBMS_VECTOR.CREATE_CREDENTIAL` credential helper procedure to create a credential name for storing user authentication details in Oracle Database.
- CREATE_INDEX
  Use the `DBMS_VECTOR.CREATE_INDEX` procedure to create a vector index.
- DISABLE_CHECKPOINT
  Use the `DISABLE_CHECKPOINT` procedure to disable the Checkpoint feature for a given Hierarchical Navigable Small World (HNSW) index user and HNSW index name. This operation purges all older checkpoints for the HNSW index. It also disables the creation of future checkpoints as part of the HNSW graph refresh.
- DROP_CREDENTIAL
  Use the `DBMS_VECTOR.DROP_CREDENTIAL` credential helper procedure to drop an existing credential name from the data dictionary.

- DROP_ONNX_MODEL Procedure
  This procedure deletes the specified ONNX model.
- ENABLE_CHECKPOINT
  Use the `ENABLE_CHECKPOINT` procedure to enable the Checkpoint feature for a given Hierarchical Navigable Small World (HNSW) index user and HNSW index name.
- GET_INDEX_STATUS
  Use the `GET_INDEX_STATUS` procedure to query the status of a vector index creation.
- INDEX_ACCURACY_QUERY
  Use the `DBMS_VECTOR.INDEX_ACCURACY_QUERY` function to verify the accuracy of a vector index for a given query vector, top-K, and target accuracy.
- INDEX_ACCURACY_REPORT
  Use the `DBMS_VECTOR.INDEX_ACCURACY_REPORT` function to capture from your past workloads, accuracy values achieved by approximate searches using a particular vector index for a certain period of time.
- INDEX_VECTOR_MEMORY_ADVISOR
  Use the `INDEX_VECTOR_MEMORY_ADVISOR` procedure to determine the vector memory size needed for a particular vector index. This helps you evaluate the number of indexes that can fit for each simulated vector memory size.
- LOAD_ONNX_MODEL
  This procedure enables you to load an ONNX model into the Database.
- LOAD_ONNX_MODEL_CLOUD
  This procedure enables you to load an ONNX model from object storage into the Database.
- QUERY
  Use the `DBMS_VECTOR.QUERY` function to perform a similarity search operation which returns the top-k results as a JSON array.
- REBUILD_INDEX
  Use the `DBMS_VECTOR.REBUILD_INDEX` function to rebuild a vector index.
- RERANK
  Use the `DBMS_VECTOR.RERANK` function to reassess and reorder an initial set of results to retrieve more relevant search output.
- UTL_TO_CHUNKS
  Use the `DBMS_VECTOR.UTL_TO_CHUNKS` chainable utility function to split a large plain text document into smaller chunks of text.
- UTL_TO_EMBEDDING and UTL_TO_EMBEDDINGS
  Use the `DBMS_VECTOR.UTL_TO_EMBEDDING` and `DBMS_VECTOR.UTL_TO_EMBEDDINGS` chainable utility functions to generate one or more vector embeddings from textual documents and images.
- UTL_TO_GENERATE_TEXT
  Use the `DBMS_VECTOR.UTL_TO_GENERATE_TEXT` chainable utility function to generate a text response for a given prompt or an image, by accessing third-party text generation models.

CREATE_CREDENTIAL

Use the `DBMS_VECTOR.CREATE_CREDENTIAL` credential helper procedure to create a credential name for storing user authentication details in Oracle Database.

Purpose

To securely manage authentication credentials in the database. You require these credentials to enable access during REST API calls to your chosen third-party service provider, such as Cohere, Google AI, Hugging Face, Oracle Cloud Infrastructure (OCI) Generative AI, OpenAI, or Vertex AI.

A credential name holds authentication parameters, such as user name, password, access token, private key, or fingerprint.

Note that if you are using Oracle Database as the service provider, then you do not need to create a credential.

> **WARNING:**
>
> Certain features of the database may allow you to access services offered separately by third-parties, for example, through the use of JSON specifications that facilitate your access to REST APIs.
>
> Your use of these features is solely at your own risk, and you are solely responsible for complying with any terms and conditions related to use of any such third-party services. Notwithstanding any other terms and conditions related to the third-party services, your use of such database features constitutes your acceptance of that risk and express exclusion of Oracle's responsibility or liability for any damages resulting from such access.

Syntax

```
DBMS_VECTOR.CREATE_CREDENTIAL (
    CREDENTIAL_NAME     IN VARCHAR2,
    PARAMS              IN JSON DEFAULT NULL
);
```

CREDENTIAL_NAME

Specify a name of the credential that you want to create for holding authentication parameters.

PARAMS

Specify authentication parameters in JSON format, based on your chosen service provider.

Generative AI requires the following authentication parameters:

```
{
"user_ocid"       : "<user ocid>",
"tenancy_ocid"    : "<tenancy ocid>",
"compartment_ocid": "<compartment ocid>",
"private_key"     : "<private key>",
```

```
"fingerprint"     : "<fingerprint>"
}
```

Cohere, Google AI, Hugging Face, OpenAI, and Vertex AI require the following authentication parameter:

```
{ "access_token": "<access token>" }
```

Table 12-2 Parameter Details

Parameter	Description
`user_ocid`	Oracle Cloud Identifier (OCID) of the user, as listed on the User Details page in the OCI console.
`tenancy_ocid`	OCID of your tenancy, as listed on the Tenancy Details page in the OCI console.
`compartment_ocid`	OCID of your compartment, as listed on the Compartments information page in the OCI console.

`private_key`	OCI private key.
	Note: The generated private key may appear as:
	`-----BEGIN RSA PRIVATE KEY-----`
	`<private key string>`
	`-----END RSA PRIVATE KEY-----`
	You pass the `<private key string>` value (excluding the `BEGIN` and `END` lines), either as a single line or as multiple lines.

`fingerprint`	Fingerprint of the OCI profile key, as listed on the User Details page under API Keys in the OCI console.
`access_token`	Access token obtained from your third-party service provider.

Required Privilege

You need the `CREATE CREDENTIAL` privilege to call this API.

Examples

- For Generative AI:

```
declare
  jo json_object_t;
begin
  jo := json_object_t();

jo.put('user_ocid','ocid1.user.oc1..aabbalbbaa1112233aabbaabb1111222aa1111b
b');

jo.put('tenancy_ocid','ocid1.tenancy.oc1..aaaaalbbbb1112233aaaabbaa1111222a
aa111a');

jo.put('compartment_ocid','ocid1.compartment.oc1..ababalabab1112233abababab
1111222aba11ab');
  jo.put('private_key','AAAaaaBBB11112222333...AAA111AAABBB222aaa1a/+');
  jo.put('fingerprint','01:1a:a1:aa:12:a1:12:1a:ab:12:01:ab:a1:12:ab:1a');
```

```
  dbms_vector.create_credential(
    credential_name   => 'OCI_CRED',
    params            => json(jo.to_string));
end;
/
```

- For Cohere:

```
declare
  jo json_object_t;
begin
  jo := json_object_t();
  jo.put('access_token', 'A1Aa0abA1AB1a1Abc123ab1A123ab123AbcA12a');
  dbms_vector.create_credential(
    credential_name   => 'COHERE_CRED',
    params            => json(jo.to_string));
end;
/
```

End-to-end examples:

To run end-to-end example scenarios using this procedure, see Use LLM-Powered APIs to Generate Summary and Text.

CREATE_INDEX

Use the `DBMS_VECTOR.CREATE_INDEX` procedure to create a vector index.

Purpose

To create a vector index such as Hierarchical Navigable Small World (HNSW) vector index or Inverted File Flat (IVF) vector index.

Syntax

```
DBMS_VECTOR.CREATE_INDEX (
    idx_name                    IN VARCHAR2,
    table_name                  IN VARCHAR2,
    idx_vector_col              IN VARCHAR2,
    idx_include_cols            IN VARCHAR2 DEFAULT NULL,
    idx_partitioning_scheme     IN VARCHAR2 default 'LOCAL',
    idx_organization            IN VARCHAR2,
    idx_distance_metric         IN VARCHAR2 DEFAULT COSINE,
    idx_accuracy                IN NUMBER DEFAULT 90,
    idx_parameters              IN CLOB,
    idx_parallel_creation       IN NUMBER DEFAULT 1
);
```

Parameters

Parameter	Description
idx_name	Name of the index to create.
table_name	Table on which to create the index.

Parameter	Description
idx_vector_col	Vector column on which to create the index.
idx_include_cols	A comma-separated list of column names to be covered by the index.
idx_partitioning_scheme	Partitioning scheme for IVF indexes: • GLOBAL • LOCAL IVF indexes support both global and local indexes on partitioned tables. By default, these indexes are globally partitioned by centroid. You can choose to create a local IVF index, which provides a one-to-one relationship between the base table partitions or subpartitions and the index partitions. For detailed information on these partitioning schemes, see Inverted File Flat Vector Indexes Partitioning Schemes.
idx_organization	Index organization: • NEIGHBOR PARTITIONS • INMEMORY NEIGHBOR GRAPH For detailed information on these organization types, see Manage the Different Categories of Vector Indexes.
idx_distance_metric	Distance metric or mathematical function used to compute the distance between vectors: • COSINE (default) • MANHATTAN • HAMMING • JACCARD • DOT • EUCLIDEAN • L2_SQUARED • EUCLIDEAN_SQUARED For detailed information on each of these metrics, see Vector Distance Functions and Operators.

Parameter	Description
idx_accuracy	Target accuracy at which the approximate search should be performed when running an approximate search query. As explained in Understand Approximate Similarity Search Using Vector Indexes, you can specify non-default target accuracy values either by specifying a percentage value or by specifying internal parameters values, depending on the index type you are using. • For an HNSW approximate search: In the case of an HNSW approximate search, you can specify a target accuracy percentage value to influence the number of candidates considered to probe the search. This is automatically calculated by the algorithm. A value of 100 will tend to impose a similar result as an exact search, although the system may still use the index and will not perform an exact search. The optimizer may choose to still use an index as it may be faster to do so given the predicates in the query. Instead of specifying a target accuracy percentage value, you can specify the EFSEARCH parameter to impose a certain maximum number of candidates to be considered while probing the index. The higher that number, the higher the accuracy. For detailed information, see Understand Hierarchical Navigable Small World Indexes. • For an IVF approximate search: In the case of an IVF approximate search, you can specify a target accuracy percentage value to influence the number of partitions used to probe the search. This is automatically calculated by the algorithm. A value of 100 will tend to impose an exact search, although the system may still use the index and will not perform an exact search. The optimizer may choose to still use an index as it may be faster to do so given the predicates in the query. Instead of specifying a target accuracy percentage value, you can specify the NEIGHBOR PARTITION PROBES parameter to impose a certain maximum number of partitions to be probed by the search. The higher that number, the higher the accuracy. For detailed information, see Understand Inverted File Flat Vector Indexes.

Parameter	Description
idx_parameters	Type of vector index and associated parameters. Specify the indexing parameters in JSON format: • For HNSW indexes: – type: Type of vector index to create, that is, HNSW – neighbors: Maximum number of connections permitted per vector in the HNSW graph – efConstruction: Maximum number of closest vector candidates considered at each step of the search during insertion For example: { "type" : "HNSW", "neighbors" : 3, "efConstruction" : 4 } For detailed information on these parameters, see Hierarchical Navigable Small World Index Syntax and Parameters. • For IVF indexes: – type: Type of vector index to create, that is, IVF – partitions: Neighbor partition or cluster in which you want to divide your vector space For example: { "type" : "IVF", "partitions" : 5 } For detailed information on these parameters, see Inverted File Flat Index Syntax and Parameters.
idx_parallel_creation	Number of parallel threads used for index construction.

Examples

- Specify neighbors and efConstruction for HNSW indexes:

```
dbms_vector.create_index(
    'v_hnsw_01',
    'vpt01',
    'EMBEDDING',
     NULL,
     NULL,
    'INMEMORY NEIGHBOR GRAPH',
    'EUCLIDEAN',
     95,
    '{"type" : "HNSW", "neighbors" : 3, "efConstruction" : 4}');
```

- Specify the number of partitions for IVF indexes:

```
dbms_vector.create_index(
    'V_IVF_01',
    'vpt01',
    'EMBEDDING',
     NULL,
     NULL,
    'NEIGHBOR PARTITIONS',
    'EUCLIDEAN',
     95,
    '{"type" : "IVF", "partitions" : 5}');
```

DISABLE_CHECKPOINT

Use the `DISABLE_CHECKPOINT` procedure to disable the Checkpoint feature for a given Hierarchical Navigable Small World (HNSW) index user and HNSW index name. This operation purges all older checkpoints for the HNSW index. It also disables the creation of future checkpoints as part of the HNSW graph refresh.

Syntax

```
DBMS_VECTOR.DISABLE_CHECKPOINT('INDEX_USER',['INDEX_NAME']);
```

INDEX_USER

Specify the user name of the HNSW vector index owner.

INDEX_NAME

Specify the name of the HNSW vector index for which you want to disable the Checkpoint feature.

The `INDEX_NAME` clause is optional. If you do not specify the index name, then this procedure disables the Checkpoint feature for all HNSW vector indexes under the given user.

Examples

- Using both the index name and index user:

```
DBMS_VECTOR.DISABLE_CHECKPOINT('VECTOR_USER','VIDX1');
```

- Using only the index user:

```
DBMS_VECTOR.DISABLE_CHECKPOINT('VECTOR_USER');
```

Related Topics

- *Oracle Database AI Vector Search User's Guide*
- ENABLE_CHECKPOINT
  Use the `ENABLE_CHECKPOINT` procedure to enable the Checkpoint feature for a given Hierarchical Navigable Small World (HNSW) index user and HNSW index name.

DROP_CREDENTIAL

Use the `DBMS_VECTOR.DROP_CREDENTIAL` credential helper procedure to drop an existing credential name from the data dictionary.

Syntax

```
DBMS_VECTOR.DROP_CREDENTIAL (
    CREDENTIAL_NAME      IN VARCHAR2
);
```

CREDENTIAL_NAME

Specify the credential name that you want to drop.

Examples

- For Generative AI:

  ```
  exec dbms_vector.drop_credential('OCI_CRED');
  ```

- For Cohere:

  ```
  exec dbms_vector.drop_credential('COHERE_CRED');
  ```

DROP_ONNX_MODEL Procedure

This procedure deletes the specified ONNX model.

Syntax

```
DBMS_VECTOR.DROP_ONNX_MODEL (model_name IN VARCHAR2,
                             force      IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 12-3 DROP_ONNX_MODEL Procedure Parameters

Parameter	Description
`model_name`	Name of the machine learning ONNX model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`force`	Forces the machine learning ONNX model to be dropped even if it is invalid. An ONNX model may be invalid if a serious system error interrupted the model build process.

Usage Note

To drop an ONNX model, you must be the owner or you must have the `DB_DEVELOPER_ROLE`.

Example

You can use the following command to delete a valid ONNX model named `doc_model` that exists in your schema.

```
BEGIN
  DBMS_VECTOR.DROP_ONNX_MODEL(model_name => 'doc_model');
END;
/
```

ENABLE_CHECKPOINT

Use the `ENABLE_CHECKPOINT` procedure to enable the Checkpoint feature for a given Hierarchical Navigable Small World (HNSW) index user and HNSW index name.

> **Note:**
>
> - This procedure only allows the index to create checkpoints. The checkpoint is created as part of the next HNSW graph refresh.
> - By default, HNSW checkpointing is enabled. If required, you can disable it using the `DBMS_VECTOR.DISABLE_CHECKPOINT` procedure.

Syntax

```
DBMS_VECTOR.ENABLE_CHECKPOINT('INDEX_USER',['INDEX_NAME']);
```

INDEX_USER

Specify the user name of the HNSW vector index owner.

INDEX_NAME

Specify the name of the HNSW vector index for which you want to enable the Checkpoint feature.

The `INDEX_NAME` clause is optional. If you do not specify the index name, then this procedure enables the Checkpoint feature for all HNSW vector indexes under the given user.

Examples

- Using both the index name and index user:

  ```
  DBMS_VECTOR.ENABLE_CHECKPOINT('VECTOR_USER','VIDX1');
  ```

- Using only the index user:

  ```
  DBMS_VECTOR.ENABLE_CHECKPOINT('VECTOR_USER');
  ```

Related Topics

- *Oracle Database AI Vector Search User's Guide*
- DISABLE_CHECKPOINT
  Use the `DISABLE_CHECKPOINT` procedure to disable the Checkpoint feature for a given Hierarchical Navigable Small World (HNSW) index user and HNSW index name. This operation purges all older checkpoints for the HNSW index. It also disables the creation of future checkpoints as part of the HNSW graph refresh.

GET_INDEX_STATUS

Use the `GET_INDEX_STATUS` procedure to query the status of a vector index creation.

Syntax

```
DBMS_VECTOR.GET_INDEX_STATUS ('USER_NAME','INDEX_NAME');
```

USER_NAME

Specify the user name of the vector index owner.

INDEX_NAME

Specify the name of the vector index. You can query the index creation status for both Hierarchical Navigable Small World (HNSW) indexes and Inverted File Flat (IVF) indexes.

Usage Notes

- You can use the `GET_INDEX_STATUS` procedure only during a vector index creation.
- The `Percentage` value is shown in the output only for Hierarchical Navigable Small World (HNSW) indexes (and not for Inverted File Flat (IVF) indexes).
- Along with the `DB_DEVELOPER_ROLE` privilege, you must have read access to the `VECSYS.VECTOR$INDEX$BUILD$` table.
- You can use the following query to view all auxiliary tables:

  ```
  select IDX_AUXILIARY_TABLES from vecsys.vector$index;
  ```

  - For HNSW indexes:

    `rowid_vid_map` stores the mapping between a row ID and vector ID. `shared_journal_change_log` stores the DML changes that are yet to be incorporated into an HNSW graph.

  - For IVF indexes:

    `centroids` stores the location for each centroid. `centroid_partitions` stores the best centroid for each vector.

- The possible values of `Stage` for HNSW vector indexes are:

Value	Description
`HNSW Index Initialization`	Initialization phase for the HNSW vector index creation
`HNSW Index Auxiliary Tables Creation`	Creation of the internal auxiliary tables for the HNSW Neighbor Graph vector index
`HNSW Index Graph Allocation`	Allocation of memory from the vector memory pool for the HNSW graph
`HNSW Index Loading Vectors`	Loading of the base table vectors into the vector pool memory
`HNSW Index Graph Construction`	Creation of the multi-layered HNSW graph with the previously loaded vectors
`HNSW Index Creation Completed`	HNSW vector index creation finished

- The possible values of `Stage` for IVF vector indexes are:

Value	Description
`IVF Index Initialization`	Initialization phase for the IVF vector index creation
`IVF Index Centroids Creation`	The K-means clustering phase that computes the cluster centroids on a sample of base table vectors
`IVF Index Centroid Partitions Creation`	Centroids assignment phase for the base table vectors
`IVF Index Creation Completed`	IVF vector index creation completed

Example

```
exec DBMS_VECTOR.GET_INDEX_STATUS('VECTOR_USER','VIDX_HNSW');

Index objn: 74745
Stage: HNSW Index Loading Vectors
Percentage: 80%
```

INDEX_ACCURACY_QUERY

Use the `DBMS_VECTOR.INDEX_ACCURACY_QUERY` function to verify the accuracy of a vector index for a given query vector, top-K, and target accuracy.

Syntax

```
DBMS_VECTOR.INDEX_ACCURACY_QUERY (
    OWNER_NAME          IN VARCHAR2,
    INDEX_NAME          IN VARCHAR2,
    QV                  IN VECTOR,
    TOP_K               IN NUMBER,
    TARGET_ACCURACY     IN NUMBER
) return VARCHAR2;

DBMS_VECTOR.INDEX_ACCURACY_QUERY (
    OWNER_NAME          IN VARCHAR2,
    INDEX_NAME          IN VARCHAR2,
    QV                  IN VECTOR,
    TOP_K               IN NUMBER,
    QUERY_PARAM         IN JSON
) return VARCHAR2;
```

Parameters

Table 12-4 INDEX_ACCURACY_QUERY (IN) Parameters of DBMS_VECTOR

Parameter	Description
`owner_name`	The name of the vector index owner.
`index_name`	The name of the vector index.
`qv`	Specifies the query vector.
`top_k`	The `top_k` value for accuracy computation.

Table 12-4 (Cont.) INDEX_ACCURACY_QUERY (IN) Parameters of DBMS_VECTOR

Parameter	Description
target_accuracy	The target accuracy value for the vector index.

For information about determining the accuracy of your vector indexes, see Index Accuracy Report in *Oracle Database AI Vector Search User's Guide*.

INDEX_ACCURACY_REPORT

Use the DBMS_VECTOR.INDEX_ACCURACY_REPORT function to capture from your past workloads, accuracy values achieved by approximate searches using a particular vector index for a certain period of time.

Syntax

```
DBMS_VECTOR.INDEX_ACCURACY_REPORT (
    OWNER_NAME          IN VARCHAR2,
    INDEX_NAME          IN VARCHAR2,
    START_TIME          IN TIMESTAMP WITH TIME ZONE,
    END_TIME            IN TIMESTAMP WITH TIME ZONE
) return NUMBER;
```

Parameters

Table 12-5 INDEX_ACCURACY_REPORT (IN) Parameters of DBMS_VECTOR

Parameter	Description
owner_name	The name of the vector index owner.
index_name	The name of the vector index.
start_time	Specifies from what time to capture query vectors to consider for the accuracy computation. A NULL start_time uses query vectors captured in the last 24 hours.
end_time	Specifies an end point up until which query vectors are considered for accuracy computation. A NULL end_time uses query vectors captured from start_time until the current time.

For information about determining the accuracy of your vector indexes, see Index Accuracy Report in *Oracle Database AI Vector Search User's Guide*.

INDEX_VECTOR_MEMORY_ADVISOR

Use the `INDEX_VECTOR_MEMORY_ADVISOR` procedure to determine the vector memory size needed for a particular vector index. This helps you evaluate the number of indexes that can fit for each simulated vector memory size.

Syntax

- Using the number and type of vector dimensions that you want to store in your vector index.

```
DBMS_VECTOR.INDEX_VECTOR_MEMORY_ADVISOR(
    INDEX_TYPE         IN    VARCHAR2,
    NUM_VECTORS        IN    NUMBER,
    DIM_COUNT          IN    NUMBER,
    DIM_TYPE           IN    VARCHAR2,
    PARAMETER_JSON     IN    CLOB,
    RESPONSE_JSON      OUT   CLOB);
```

- Using the table and vector column on which you want to create your vector index:

```
DBMS_VECTOR.INDEX_VECTOR_MEMORY_ADVISOR(
    TABLE_OWNER         IN    VARCHAR2,
    TABLE_NAME          IN    VARCHAR2,
    COLUMN_NAME         IN    VARCHAR2,
    INDEX_TYPE          IN    VARCHAR2,
    PARAMETER_JSON      IN    CLOB,
    RESPONSE_JSON       OUT   CLOB);
```

Table 12-6 Syntax Details: INDEX_VECTOR_MEMORY_ADVISOR

Parameter	Description
`INDEX_TYPE`	Type of vector index: • `IVF` for Inverted File Flat (IVF) vector indexes • `HNSW` for Hierarchical Navigable Small World (HNSW) vector indexes
`NUM_VECTORS`	Number of vectors that you plan to create the vector index with.
`DIM_COUNT`	Number of dimensions of a vector as a `NUMBER`.
`DIM_TYPE`	Type of dimensions of a vector. Possible values are: • `FLOAT16` • `FLOAT32` • `FLOAT64` • `INT8`
`TABLE_OWNER`	Owner name of the table on which to create the vector index.
`TABLE_NAME`	Table name on which to create the vector index.
`COLUMN_NAME`	Name of the vector column on which to create the vector index.

Table 12-6 (Cont.) Syntax Details: INDEX_VECTOR_MEMORY_ADVISOR

Parameter	Description
PARAMETER_JSON	Input parameter in JSON format. You can specify only one of the following form: • PARAMTER_JSON=>{"accuracy":value} • INDEX_TYPE=>IVF, parameter_json=>{"neighbor_partitions":value} • INDEX_TYPE=>HNSW, parameter_json=>{"neighbors":value} **Note**: You cannot specify values for accuracy along with neighbor_partitions or neighbors.
RESPONSE_JSON	JSON-formatted response string.

Examples

- Using neighbors in the parameters list:

```
SET SERVEROUTPUT ON;
DECLARE
    response_json CLOB;
BEGIN
    DBMS_VECTOR.INDEX_VECTOR_MEMORY_ADVISOR(
        INDEX_TYPE=>'HNSW',
        NUM_VECTORS=>10000,
        DIM_COUNT=>768,
        DIM_TYPE=>'FLOAT32',
        PARAMETER_JSON=>'{"neighbors":32}',
        RESPONSE_JSON=>response_json);
END;
/
```

Result:

```
Suggested vector memory pool size: 59918628 Bytes
```

- Using accuracy in the parameters list:

```
SET SERVEROUTPUT ON;
DECLARE
    response_json CLOB;
BEGIN
    DBMS_VECTOR.INDEX_VECTOR_MEMORY_ADVISOR(
        INDEX_TYPE=>'HNSW',
        NUM_VECTORS=>10000,
        DIM_COUNT=>768,
        DIM_TYPE=>'FLOAT32',
        PARAMETER_JSON=>'{"accuracy":90}',
        RESPONSE_JSON=>response_json);
END;
/
```

Result:

```
Suggested vector memory pool size: 53926765 Bytes
```

- Using the table and vector column on which you want to create the vector index:

```
SET SERVEROUTPUT ON;
DECLARE
    response_json CLOB;
BEGIN
    DBMS_VECTOR.INDEX_VECTOR_MEMORY_ADVISOR(
        'VECTOR_USER',
        'VECTAB',
        'DATA_VECTOR',
        'HNSW',
        RESPONSE_JSON=>response_json);
END;
/
```

Example result:

```
Using default accuracy: 90%
Suggested vector memory pool size: 76396251 Bytes
```

Related Topics

- *Oracle Database AI Vector Search User's Guide*

LOAD_ONNX_MODEL

This procedure enables you to load an ONNX model into the Database.

Syntax

```
DBMS_VECTOR.LOAD_ONNX_MODEL (
     directory  VARCHAR2,
     file_name   VARCHAR2,
     model_name  VARCHAR2,
     metadata   JSON);


DBMS_VECTOR.LOAD_ONNX_MODEL(
model_name  IN  VARCHAR2,
model_data  IN  BLOB,
metadata    IN  JSON);
```

Parameters

Table 12-7 LOAD_ONNX_MODEL Procedure Parameters

Parameter	Description
`directory`	The directory name of the data dump. For example, `DM_DUMP`.
`file_name`	A `VARCHAR2` type parameter that specifies the name of the ONNX model.

Table 12-7 (Cont.) LOAD_ONNX_MODEL Procedure Parameters

Parameter	Description
model_name	Name of the model in the form [schema_name.]model_name. If you do not specify a schema, then your own schema is used.
model_data	It is a BLOB holding the ONNX representation of the model. The BLOB contains the identical byte sequence as the one stored in an ONNX file.
metadata	A JSON description of the metadata describing the model. The metadata at minimum must describe the machine learning function supported by the model. The model's metadata parameters are described in JSON Metadata Parameters for ONNX Models.

Examples

The following examples illustrates a code snippet of using the DBMS_VECTOR.LOAD_ONNX_MODEL procedure. The complete step-by-step example is illustrated in Import ONNX Models and Generate Embeddings.

```
EXECUTE DBMS_VECTOR.LOAD_ONNX_MODEL(
    'DM_DUMP',
    'my_embedding_model.onnx',
    'doc_model',
    JSON('{"function" : "embedding", "embeddingOutput" : "embedding", "input": {"input":
["DATA"]}}'));
```

```
DBMS_VECTOR.LOAD_ONNX_MODEL('my_embedding_model.onnx',
                                                :blob_bind_variable,
                                                 JSON('{"function" :
"embedding",
                                                        "embeddingOutput" :
"embedding" ,
                                                         "input":{"input":
["DATA"]}}'));
```

For a complete example to illustrate how you can define a BLOB variable and use it in the LOAD_ONNX_MODEL procedure, you can have the following:

```
CREATE OR REPLACE MY_LOAD_EMBEDDING_MODEL(embedding_model_name VARCHAR2,
onnx_blob BLOB) IS
BEGIN
DBMS_VECTOR.LOAD_ONNX_MODEL(embedding_model_name,
                            onnx_blob,
                            JSON('{"function" : "embedding",
                                   "embeddingOutput" : "embedding" ,
                                   "input":{"input": ["DATA"]}}'));
END;
/
```

Usage Notes

The name of the model follows the same restrictions as those used for other machine learning models, namely:

- The schema name, if provided, is limited to 128 characters.
- The model name is limited to 123 characters and must follow the rules of unquoted identifiers: they contain only alphanumeric characters, the underscore (_), dollar sign ($), and pound sign (#). The initial character must be alphabetic.
- The model size is limited to 1 gigabyte.
- The model must not depend on external initializers. To know more about initializers and other ONNX concepts, see https://onnx.ai/onnx/intro/concepts.html.
- There are default input and output names for input and output attributes for models that are prepared by the Python utility. You can load those models without the JSON parameters. For example:

```
EXECUTE DBMS_VECTOR.LOAD_ONNX_MODEL('DM_DUMP', 'my_embedding_model.onnx',
'doc_model'));
```

- JSON Metadata Parameters for ONNX Models
  When importing models using the `IMPORT_ONNX_MODEL` (`DBMS_DATA_MINING`), `LOAD_ONNX_MODEL` (`DBMS_VECTOR`), or `LOAD_ONNX_MODEL_CLOUD` (`DBMS_VECTOR`) procedures, you supply metadata as JSON parameters.

> **See Also:**
>
> *Oracle Machine Learning for SQL User's Guide* for examples of using ONNX models for machine learning tasks

JSON Metadata Parameters for ONNX Models

When importing models using the `IMPORT_ONNX_MODEL` (`DBMS_DATA_MINING`), `LOAD_ONNX_MODEL` (`DBMS_VECTOR`), or `LOAD_ONNX_MODEL_CLOUD` (`DBMS_VECTOR`) procedures, you supply metadata as JSON parameters.

Parameters

Field	Value Type	Description
`function`	String	Specify regression, classification, clustering, or embedding. This is a mandatory setting. **NOTE**: The only JSON parameter required when importing the model is the machine learning function.
`input`	NA	Describes the model input mapping. See "Input" in Usage Notes.
`regressionOutput`	String	The name of the regression model output that stores the regression results. The output is expected to be a tensor of supported shape of any supported regression output type. See "Output" in Usage Notes.
`classificationProbOutput`	String	The name of the classification model output storing probabilities. The output is expected to be a tensor value of type float (width 32/64) of supported shape. See "Automatic normalization of output probabilities" in Usage Notes.

Field	Value Type	Description
clusteringDistanceOutput	String	The name of the clustering model output storing distances. The output is of type float (width 16/32/64) of supported shape.
clusteringProbOutput	String	The name of the clustering model output storing probabilities. The output is of type float (width 16/32/64) of supported shape.
classificationLabelOutput	String	The name of the model output holding label information. You have the following metadata parameters to specify the labels for classification: • labels: specify the labels directly in the JSON metadata • classificationLabelOutput: specify the model output that provides labels If you do not specify any value for this parameter or the function of the model is not classification, you will receive an error. The user can specify to use labels from the model directly by setting classificationLabelOutput to the model output holding the label information. The tensor output holding the label information must be the same size as the number of classes and must be of integer or string type. If the tensor that holds the labels is of string type, the returned type of the PREDICTION operator is VARCHAR2. If the tensor that holds the labels is of integer type, the returned type of the PREDICTION operator is NUMBER.
normalizeProb	String	Describes automatic normalization of output probabilities. See "Automatic normalization of output probabilities" in Usage Notes.
labels	NA	The labels used for classification. If you want to use custom labels, specify the labels using the labels field in the JSON metadata. The field can be set to an array of length equal to the number of classes. The labels for the class i must be stored at index i of the label array. If an array of strings is used, the returned type of the PREDICTION operator is VARCHAR2. The size of the string labels specified by the user cannot exceed *4000* bytes. If an array of numbers is used, the returned type of the PREDICTION operator is NUMBER. If you do not specify labels or classificationLabelOutput, classes are identified by integers in the range 1 to N where N is the number of classes. In this case, the returned type of the PREDICTION operator is NUMBER.
embeddingOutput	String	The model output that holds the generated embeddings.

Field	Value Type	Description
`suitableDistanceMetrics`	String	An array of names of suitable distance metrics for the model. The names must be the names of the distance metrics used for the Oracle `VECTOR_DISTANCE` operator. To know the supported distance metrics, see Vector Distance Metrics. This parameter is for informational purposes only.
`normalization`	Boolean	A boolean value indicates if normalization is applied to the output vector. The value 1 means normalization is applied. Normalization is process of converting an embedding vector so that it's norm or length equals 1. A normalized vector maintains its direction but its length becomes 1. The resulting vector is often called a unit vector.
`maxSequenceLength`	Number	The maximum length of the token (input) sequence that is meaningful for the model. This parameter sets a limit on the number of tokens, words, or elements in each input sequence that the model will process. This ensures uniform input size for the model. For example, the value could be 128, or 512 to 4096 depending on the task for which the parameter is used. A machine translation model might have a `maxSequenceLength` of 512, accommodating sentences or paragraphs up to 512 tokens for translation tasks. This parameter is for informational purposes only.
`pooling`	String	Indicates the pooling function performed on the output vector. This parameter is for informational purposes only.
`modelDescription`	Object	A JSON object that allows users to add additional descriptions to the models complementing the existing ONNX metadata for model description. This parameter is for informational purposes only.
`languages`	String	A comma-separated list of language name or abbreviation, as described in *"*A.1 Languages*"* of *Oracle Database Globalization Support Guide*. If you import multi-lingual embedding model, specify the language or the language abbreviation as the metadata. This parameter is for informational purposes only.
`tokenizer`	String	Tokenizers help in transforming text into words. There are several tokenizers available, including: bert, gpt2, bpe, wordpiece, sentencepiece, and clip. This parameter is for informational purposes only.

Field	Value Type	Description
embeddingLayer	String	An identifier for the embedding layer. An embedding layer, serving as a hidden layer in neural networks, transforms input data from high to lower dimensions, enhancing the network's understanding of input relationships and data processing efficiency. Embedding layer helps in processing and analyzing categorical or discrete data. It achieves this by transforming categories into continuous embeddings, capturing the essential semantic relationships and similarities between them. For example the last hidden state in some transformer, or a layer in a resnet network. This parameter is for informational purposes only.
defaultOnNull	NA	Specify the replacement of missing values in the JSON using the defaultOnNull field. If defaultOnNull is not specified, the replacement of missing values is not performed. The defaultOnNull sets the missing values to NULL by default. You can override the default value of NULL by providing meaningful default values to substitute for NULL. The field must be a JSON object literal, whose fields are the input attribute names and whose values are the default values for the input. Note that the default value is of type string and must be a valid Oracle PL/SQL NVL value for the given datatype.

Note: The parameters are case-sensitive. A number of default conventions for output parameter names and default values allows to minimize the information that you may have to provide. The parameters such as suitableDistanceMetrics are informational only and you are not expected to provide this information while importing the model. The JSON descriptor may specify only one input attribute. If more are specified, you will receive an error. You will receive an error if the normalizeProb field is specified as the JSON metadata parameter.

Usage Notes

The name of the model follows the same restrictions as those used for other machine learning models, namely:

- **Input**

  When importing a model from an ONNX representation, you must specify the name of the attribute used for scoring and how it maps to actual ONNX inputs. A scoring operator uses these attribute names to identify the columns to be used. (For example, PREDICTION). Follow these conventions to specify the attribute names using the input field:

  not specified: When the field input is not specified, attribute names are mapped directly to model inputs by name. That is, if the attribute name is not specified in the JSON metadata, then the name of the input tensor is used as an attribute name. Each model input must have dimension [batch_size, value]. If you do not specify input in the JSON metatdata, the value must be 1. You don't have to specify extra metadata if the input of the model already conforms to the format. For an embedding model, a single input is provided that may be used in batches. Here, if the input parameter is not specified in the JSON metadata, the valid model will have [batch_size, 1].

You must ensure that all attribute names, whether implied by the model or explicitly set by you through the input field, are valid Oracle Database identifiers for column names. Each attribute name within a model must be unique, ensuring no duplicates exist.

You can explicitly specify attribute name for model that use input tensors that have a dimension larger than 1 (for example, (batch_size, 2)). In this case, you must specify a name for each of these values for them to be interpreted as independent attribute name. This can be done for regression, classification, clustering which are models whose scoring operation can take multiple input attributes.

- **Output**

  As models might have multiple outputs, you can specify which output is of interest for a specific machine learning technique. You have the following ways to specify model outputs:

  – Specify the output name of interest in the JSON during model import. If the specified name is not a valid model output (see the table with valid outputs for a given machine learning function), you will receive an error.

  – If the model produces an output that matches the expected output name for the given machine learning technique (for example, `classificationProbOutput`) and you didn't explicitly specify it, the output is automatically assumed.

  – If you do not specify any output name and the model has a single output, the system assumes that the single output corresponds to a default specific to the machine learning technique. For an embedding machine learning function, the default value is `embeddingOutput`.
  The system reports an error if you do not specify model outputs or if you supply outputs that the specified machine learning function does not support. The following table displays supported outputs for a specific machine learning function:

Machine learning function	Output
regression	`regressionOutput`
classification	`classificationProbOutput`
clustering	`clusteringDistanceOutput`
embedding	`embeddingOutput`

  If none of the mentioned model outputs are specified, or if you supply outputs that are not supported by the specified machine learning function, you will receive an error.

- **Automatic Normalization of Output Probabilities**

  Many users widely employ the softmax function to normalize the output of multi-class classification models, as it enables to easily interpret the results of these models. The **softmax function** is a mathematical function that converts a vector of real numbers into a probability distribution. It is also known as the softargmax, or normalized exponential function. This function is available to you to specify at the model import-time that a softmax normalization must be applied to the tensor holding output probabilities such as

`classificationProbOutput` and `clusteringProbOutput`. Specify `normalizeProb` to define the normalization that must be applied for softmax normalization. The default setting is `none`, indicating that no normalization is applied. You can choose `softmax` to apply a softmax function to the probability output. Specifying any other value for this field will result in an error during import. Additionally, specifying this field for models other than classification and clustering will also lead to an error.

Example: Specifying JSON Metadata Parameters for Embedding Models

The following example illustrates a simple case of how you can specify JSON metadata parameters while importing an ONNX embedding model into the Database using the `DBMS_VECTOR.IMPORT_ONNX_MODEL` procedure.

```
DBMS_VECTOR.IMPORT_ONNX_MODEL('my_embedding_model.onnx', 'doc_model',
                JSON('{"function" : "embedding",
                      "embeddingOutput" : "embedding" ,
                       "input":{"input": ["DATA"]}}'));
```

Example: Specifying Complete JSON Metadata Parameters for Embedding Models

The following example illustrates how to provide a complete JSON metadata parameters, with an exception of `embeddingLayer`, for importing embedding models.

```
DECLARE
  metadata JSON;
  mdtxt varchar2(4000);
BEGIN
  metadata := JSON(q'#
            {
              "function"                : "embedding",
              "embeddingOutput"         : "embedding",
              "input"                   : { "input" : ["txt"]},
              "maxSequenceLength"       : 512,
              "tokenizer"               : "bert",
              "suitableDistanceMetrics" : [ "DOT", "COSINE", "EUCLIDEAN"],
              "pooling"                 : "Mean Pooling",
              "normalization"           : true,
              "languages"               : ["US"],
              "modelDescription"        : {
                  "description" : "This model was tuned for semantic search:
Given a query/question, if can find relevant passages. It was trained on a
large and diverse set of (question, a
nswer) pairs.",
                  "url" : "https://example.co/sentence-transformers/
my_embedding_model"}
            }
            #');
  -- load the onnx model
    DBMS_VECTOR.IMPORT_ONNX_MODEL('my_embedding_model.onnx', 'doc_model',
metadata);
END;
/
```

> **See Also:**
>
> *Oracle Machine Learning for SQL User's Guide* for examples of using ONNX models for machine learning tasks

LOAD_ONNX_MODEL_CLOUD

This procedure enables you to load an ONNX model from object storage into the Database.

Syntax

```
DBMS_VECTOR.LOAD_ONNX_MODEL_CLOUD (
     model_name  IN  VARCHAR2,
     credential  IN  VARCHAR2,
     uri         IN  VARCHAR2,
     metadata    IN  JSON DEFAULT JSON('{"function" : "embedding", '||
                          '"embeddingOutput" : "embedding", "input": {"input":
["DATA"]}}')
);
```

Parameters

Table 12-8 LOAD_ONNX_MODEL_CLOUD Procedure Parameters

Parameter	Description
`model_name`	The name of the model in the form `[schema_name.]model_name`. If you do not specify a schema, then your own schema is used.
`credential`	The name of the credential to be used to access Object Store.
`uri`	The URI of the ONNX model.
`metadata`	A JSON description of the metadata describing the model. The metadata at minimum must describe the machine learning function supported by the model. The model's metadata parameters are described in JSON Metadata Parameters for ONNX Models.

Examples

The following example includes a code snippet of using the `DBMS_VECTOR.LOAD_ONNX_MODEL_CLOUD` procedure.

```
EXECUTE DBMS_VECTOR.LOAD_ONNX_MODEL_CLOUD(
    model_name => 'database',
    credential => 'MYCRED',
    uri => 'https://objectstorage.us-phoenix-1.oraclecloud.com/n/namespace-string/b/
bucketname/o/all-MiniLM-L6-v2.onnx',
    metadata => JSON('{"function" : "embedding", "embeddingOutput" : "embedding" , "input":
{"input": ["DATA"]}}')
);
```

Usage Notes

The name of the model follows the same restrictions as those used for other machine learning models, namely:

- The schema name, if provided, is limited to 128 characters.

- The model name is limited to 123 characters and must follow the rules of unquoted identifiers: they contain only alphanumeric characters, the underscore (_), dollar sign ($), and pound sign (#). The initial character must be alphabetic.
- The model size is limited to 1 gigabyte.
- The model must not depend on external initializers. To know more about initializers and other ONNX concepts, see https://onnx.ai/onnx/intro/concepts.html.
- There are default input and output names for input and output attributes for models that are prepared by the Python utility. You can load those models without the JSON parameters. For example:

```
EXECUTE DBMS_VECTOR.LOAD_ONNX_MODEL_CLOUD(
    'database',
    'MYCRED',
    'https://objectstorage.us-phoenix-1.oraclecloud.com/n/namespace-
string/b/bucketname/o/all-MiniLM-L6-v2.onnx'
);
```

> **See Also:**
>
> *Oracle Machine Learning for SQL User's Guide* for examples of using ONNX models for machine learning tasks

QUERY

Use the `DBMS_VECTOR.QUERY` function to perform a similarity search operation which returns the top-k results as a JSON array.

Syntax

Query is overloaded and supports a version with `query_vector` passed in as a `VECTOR` type in addition to `CLOB`.

```
DBMS_VECTOR.QUERY (
    TAB_NAME              IN VARCHAR2,
    VEC_COL_NAME          IN VARCHAR2,
    QUERY_VECTOR          IN CLOB,
    TOP_K                 IN NUMBER,
    VEC_PROJ_COLS         IN JSON_ARRAY_T DEFAULT NULL,
    IDX_NAME              IN VARCHAR2 DEFAULT NULL,
    DISTANCE_METRIC       IN VARCHAR2 DEFAULT 'COSINE',
    USE_INDEX             IN BOOLEAN DEFAULT TRUE,
    ACCURACY              IN NUMBER DEFAULT '90',
    IDX_PARAMETERS        IN CLOB DEFAULT NULL
) return JSON_ARRAY_T;

DBMS_VECTOR.QUERY (
    TAB_NAME              IN VARCHAR2,
    VEC_COL_NAME          IN VARCHAR2,
    QUERY_VECTOR          IN VECTOR,
    TOP_K                 IN NUMBER,
    VEC_PROJ_COLS         IN JSON_ARRAY_T DEFAULT NULL,
```

```
    IDX_NAME              IN VARCHAR2 DEFAULT NULL,
    DISTANCE_METRIC       IN VARCHAR2 DEFAULT 'COSINE',
    USE_INDEX             IN BOOLEAN DEFAULT TRUE,
    ACCURACY              IN NUMBER DEFAULT '90',
    IDX_PARAMETERS        IN CLOB DEFAULT NULL
) return JSON_ARRAY_T;
```

Parameters

Specify the input parameters in JSON format.

Table 12-9 DBMS_VECTOR.QUERY Parameters

Parameter	Description
tab_name	Table name to query
vec_col_name	Vector column name
query_vector	Query vector passed in as CLOB or VECTOR.
top_k	Number of results to be returned.
vec_proj_cols	Columns to be projected as part of the result.
idx_name	Name of the index queried.
distance_metric	Distance computation metric. Defaults to COSINE. Can also be MANHATTAN, HAMMING, DOT, EUCLIDEAN, L2_SQUARED, EUCLIDEAN_SQUARED. .
use_index	Specifies whether the search is an approximate search or exact search. Defaults to TRUE (that is, approximate).
accuracy	Specifies the minimum desired query accuracy.
idx_parameters	Specifies values of efsearch and neighbor partition probes passed in, formatted as JSON

DATA

This function accepts the input data type as VARCHAR2, NUMBER, JSON, BOOLEAN or CLOB.

REBUILD_INDEX

Use the DBMS_VECTOR.REBUILD_INDEX function to rebuild a vector index.

Purpose

To rebuild a vector index such as Hierarchical Navigable Small World (HNSW) vector index or Inverted File Flat (IVF) vector index. In case only the idx_name is provided, it rebuilds the index using get_ddl. When all the parameters are provided, it performs a drop index followed by a call to dbms_vector.create_index().

Syntax

```
DBMS_VECTOR.REBUILD_INDEX (
    idx_name                    IN VARCHAR2,
    table_name                  IN VARCHAR2 DEFAULT NULL,
    idx_vector_col              IN VARCHAR2 DEFAULT NULL,
```

```
    idx_include_cols            IN VARCHAR2 DEFAULT NULL,
    idx_partitioning_scheme     IN VARCHAR2 DEFAULT NULL,
    idx_organization            IN VARCHAR2 DEFAULT NULL,
    idx_distance_metric         IN VARCHAR2 DEFAULT 'COSINE',
    idx_accuracy                IN NUMBER DEFAULT 90,
    idx_parameters              IN CLOB DEFAULT NULL,
    idx_parallel_creation       IN NUMBER DEFAULT 1,
);
```

Parameters

Parameter	Description
idx_name	Name of the index to rebuild.
table_name	Table on which to create the index.
idx_vector_col	Vector column on which to create the index.
idx_include_cols	A comma-separated list of column names to be covered by the index.
idx_partitioning_scheme	Partitioning scheme for IVF indexes: • GLOBAL • LOCAL IVF indexes support both global and local indexes on partitioned tables. By default, these indexes are globally partitioned by centroid. You can choose to create a local IVF index, which provides a one-to-one relationship between the base table partitions or subpartitions and the index partitions. For detailed information on these partitioning schemes, see Inverted File Flat Vector Indexes Partitioning Schemes.
idx_organization	Index organization: • NEIGHBOR PARTITIONS • INMEMORY NEIGHBOR GRAPH For detailed information on these organization types, see Manage the Different Categories of Vector Indexes.
idx_distance_metric	Distance metric or mathematical function used to compute the distance between vectors: • COSINE (default) • MANHATTAN • HAMMING • JACCARD • DOT • EUCLIDEAN • L2_SQUARED • EUCLIDEAN_SQUARED For detailed information on each of these metrics, see Vector Distance Functions and Operators.

Parameter	Description
idx_accuracy	Target accuracy at which the approximate search should be performed when running an approximate search query. As explained in Understand Approximate Similarity Search Using Vector Indexes, you can specify non-default target accuracy values either by specifying a percentage value or by specifying internal parameters values, depending on the index type you are using. • For an HNSW approximate search: In the case of an HNSW approximate search, you can specify a target accuracy percentage value to influence the number of candidates considered to probe the search. This is automatically calculated by the algorithm. A value of 100 will tend to impose a similar result as an exact search, although the system may still use the index and will not perform an exact search. The optimizer may choose to still use an index as it may be faster to do so given the predicates in the query. Instead of specifying a target accuracy percentage value, you can specify the EFSEARCH parameter to impose a certain maximum number of candidates to be considered while probing the index. The higher that number, the higher the accuracy. For detailed information, see Understand Hierarchical Navigable Small World Indexes. • For an IVF approximate search: In the case of an IVF approximate search, you can specify a target accuracy percentage value to influence the number of partitions used to probe the search. This is automatically calculated by the algorithm. A value of 100 will tend to impose an exact search, although the system may still use the index and will not perform an exact search. The optimizer may choose to still use an index as it may be faster to do so given the predicates in the query. Instead of specifying a target accuracy percentage value, you can specify the NEIGHBOR PARTITION PROBES parameter to impose a certain maximum number of partitions to be probed by the search. The higher that number, the higher the accuracy. For detailed information, see Understand Inverted File Flat Vector Indexes.

Parameter	Description
idx_parameters	Type of vector index and associated parameters. Specify the indexing parameters in JSON format: • For HNSW indexes: – type: Type of vector index to create, that is, HNSW – neighbors: Maximum number of connections permitted per vector in the HNSW graph – efConstruction: Maximum number of closest vector candidates considered at each step of the search during insertion For example: { "type" : "HNSW", "neighbors" : 3, "efConstruction" : 4 } For detailed information on these parameters, see Hierarchical Navigable Small World Index Syntax and Parameters. • For IVF indexes: – type: Type of vector index to create, that is, IVF – partitions: Neighbor partition or cluster in which you want to divide your vector space For example: { "type" : "IVF", "partitions" : 5 } For detailed information on these parameters, see Inverted File Flat Index Syntax and Parameters.
idx_parallel_creation	Number of parallel threads used for index construction.

Examples

- Specify neighbors and efConstruction for HNSW indexes:

```
dbms_vector.rebuild_index(
    'v_hnsw_01',
    'vpt01',
    'EMBEDDING',
     NULL,
     NULL,
    'INMEMORY NEIGHBOR GRAPH',
    'EUCLIDEAN',
     95,
    '{"type" : "HNSW", "neighbors" : 3, "efConstruction" : 4}');
```

- Specify the number of partitions for IVF indexes:

```
dbms_vector.rebuild_index(
    'V_IVF_01',
    'vpt01',
    'EMBEDDING',
     NULL,
     NULL,
    'NEIGHBOR PARTITIONS',
    'EUCLIDEAN',
     95,
    '{"type" : "IVF", "partitions" : 5}');
```

RERANK

Use the `DBMS_VECTOR.RERANK` function to reassess and reorder an initial set of results to retrieve more relevant search output.

Purpose

To improve the relevance and quality of search results in both similarity search and Retrieval Augmented Generation (RAG) scenarios.

Reranking improves the quality of information ingested into an LLM by ensuring that the most relevant documents or chunks are prioritized. This helps to reduce hallucinations and improves the accuracy of generated outputs.

For this operation, Oracle AI Vector Search supports reranking models provided by Cohere and Vertex AI.

> **WARNING:**
>
> Certain features of the database may allow you to access services offered separately by third-parties, for example, through the use of JSON specifications that facilitate your access to REST APIs.
>
> Your use of these features is solely at your own risk, and you are solely responsible for complying with any terms and conditions related to use of any such third-party services. Notwithstanding any other terms and conditions related to the third-party services, your use of such database features constitutes your acceptance of that risk and express exclusion of Oracle's responsibility or liability for any damages resulting from such access.

Syntax

```
DBMS_VECTOR.RERANK(
                QUERY      IN CLOB,
                DOCUMENTS  IN JSON,
                PARAMS     IN JSON default NULL
) return JSON;
```

This function accepts the input containing a query as `CLOB` and a list of documents in `JSON` format. It then processes this information to generate a `JSON` object containing a reranked list of documents, sorted by score.

For example, a reranked output includes:

```
{
    "index"   : "1",
    "score"   : "0.99",
    "content" : "Jupiter boasts an impressive system of 95 known moons."
}
```

Where,

- `index` specifies the position of the document in the list of input text.
- `score` specifies the relevance score.
- `content` specifies the input text corresponding to the index.

QUERY

Specify the search query (typically from an initial search) as `CLOB`.

DOCUMENTS

Specify a JSON array of strings (list of potentially relevant documents to rerank) in the following format:

```
{
  "documents": [
  "string1",
  "string2",
    ...
  ]
}
```

PARAMS

Specify the following list of parameters in JSON format. All these parameters are mandatory.

```
{
  "provider"         : "<service provider>",
  "credential_name"  : "<credential name>",
  "url"              : "<REST endpoint URL for reranking>",
  "model"            : "<reranking model name>",
  ...
}
```

Table 12-10 RERANK Parameter Details

Parameter	Description
`provider`	Supported REST provider to access for reranking: • `cohere` • `vertexai`

Table 12-10 (Cont.) RERANK Parameter Details

Parameter	Description
`credential_name`	Name of the credential in the form: *schema.credential_name* A credential name holds authentication credentials to enable access to your provider for making REST API calls. You need to first set up your credential by calling the `DBMS_VECTOR.CREATE_CREDENTIAL` helper function to create and store a credential, and then refer to the credential name here. See CREATE_CREDENTIAL.
`url`	URL of the third-party provider endpoint for each REST call, as listed in Supported Third-Party Provider Operations and Endpoints.
`model`	Name of the reranking model in the form: *schema.model_name* If the model name is not schema-qualified, then the schema of the procedure invoker is used.

Additional REST provider parameters:

Optionally, specify additional provider-specific parameters for reranking.

> **Important:**
>
> - The following examples are for illustration purposes. For accurate and up-to-date information on additional parameters to use, refer to your third-party provider's documentation.
> - For a list of all supported REST endpoints, see Supported Third-Party Provider Operations and Endpoints.

Cohere example:

```
{
  "provider"        : "cohere",
  "credential_name" : "COHERE_CRED",
  "url"             : "https://api.cohere.example.com/rerank",
  "model"           : "rerank-english-v3.0",
  "return_documents": false,
  "top_n"           : 3
}
```

Vertex AI example:

```
{
  "provider"         : "vertexai",
  "credential_name"  : "VERTEXAI_CRED",
  "url"              : "https://googleapis.example.com/
default_ranking_config:rank",
  "model"            : "semantic-ranker-512@latest",
```

```
"ignoreRecordDetailsInResponse" : true,
"topN"              : 3
}
```

Table 12-11 Additional REST Provider Parameter Details

Parameter	Description
`return_documents`	Whether to return search results with original documents or input text (`content`): • `false` (default, also recommended) to not return any input text (return only index and score) • `true` to return input text along with index and score **Note**: With Cohere as the provider, Oracle recommends that you keep this option disabled for better performance. You may choose to enable it for debugging purposes when you need to view the original text.
`ignoreRecordDetailsInResponse`	Whether to return search results with original record details or input text (`content`): • `false` (default) to return input text along with index and score • `true` (recommended) to not return any input text (return only index and score) **Note**: With Vertex AI as the provider, Oracle recommends that you keep this option enabled for better performance. You may choose to disable it for debugging purposes when you need to view the original text.
`top_n` or `topN`	The number of most relevant documents to return.

Examples

- Using Cohere:

```
declare
  params clob;
  reranked_output json;
begin
  params := '
{
  "provider": "cohere",
  "credential_name": "COHERE_CRED",
  "url": "https://api.cohere.com/v1/rerank",
  "model": "rerank-english-v3.0",
  "return_documents": true,
  "top_n": 3
}';

  reranked_output := dbms_vector.rerank(:query,
json(:initial_retrieval_docs), json(params));
  dbms_output.put_line(json_serialize(reranked_output));
end;
/
```

- Using Vertex AI:

```
declare
  params clob;
```

```
  reranked_output json;
begin
  params := '
{
  "provider": "vertexai",
  "credential_name": "VERTEXAI_CRED",
  "url": "https://discoveryengine.googleapis.com/v1/projects/1085581009881/
locations/global/rankingConfigs/default_ranking_config:rank",
  "model": "semantic-ranker-512@latest",
  "ignoreRecordDetailsInResponse": false,
  "topN": 3
}';

  reranked_output := dbms_vector.rerank(:query,
json(:initial_retrieval_docs), json(params));
  dbms_output.put_line(json_serialize(reranked_output));
end;
/
```

End-to-end example:

To run an end-to-end example scenario using this function, see Use Reranking for Better RAG Results.

UTL_TO_CHUNKS

Use the `DBMS_VECTOR.UTL_TO_CHUNKS` chainable utility function to split a large plain text document into smaller chunks of text.

Purpose

To perform a text-to-chunks transformation. This chainable utility function internally calls the `VECTOR_CHUNKS` SQL function for the operation.

To embed a large document, you may first need to split it into multiple appropriate-sized segments or chunks through a splitting process known as chunking (as explained in Understand the Stages of Data Transformations). A chunk can be words (to capture specific words or word pieces), sentences (to capture a specific meaning), or paragraphs (to capture broader themes). A single document may be split into multiple chunks, each transformed into a vector.

Syntax

```
DBMS_VECTOR.UTL_TO_CHUNKS (
    DATA         IN  CLOB | VARCHAR2,
    PARAMS       IN  JSON  default  NULL
) return VECTOR_ARRAY_T;
```

DATA

This function accepts the input data type as `CLOB` or `VARCHAR2`.

It returns an array of CLOBs, where each CLOB contains a chunk along with its metadata in JSON format, as follows:

```
{
    "chunk_id"     : NUMBER,
    "chunk_offset" : NUMBER,
    "chunk_length" : NUMBER,
    "chunk_data"   : "VARCHAR2(4000)"
}
```

For example:

```
{"chunk_id":1,"chunk_offset":1,"chunk_length":6,"chunk_data":"sample"}
```

Where,

- chunk_id specifies the chunk ID for each chunk.
- chunk_offset specifies the original position of each chunk in the source document, relative to the start of document which has a position of 1.
- chunk_length specifies the character length of each chunk.
- chunk_data displays text pieces from each chunk.

PARAMS

Specify input parameters in JSON format:

```
{
    "by"           :     mode,
    "max"          :     max,
    "overlap"      :     overlap,
    "split"        :     split_condition,
    "custom_list"  :     [ split_chars1, ... ],
    "vocabulary"   :     vocabulary_name,
    "language"     :     nls_language,
    "normalize"    :     normalize_mode,
    "norm_options" :     [ normalize_option1, ... ],
    "extended"     :     boolean
}
```

For example:

```
JSON('
  { "by"           :    "vocabulary",
    "vocabulary"   :    "myvocab",
    "max"          :    "100",
    "overlap"      :    "0",
    "split"        :    "custom",
    "custom_list"  :    [ "<p>" , "<s>" ],
    "language"     :    "american",
    "normalize"    :    "options",
    "norm_options" :    [ "WHITESPACE" ]
  }')
```

Here is a complete description of these parameters:

Parameter	Description and Acceptable Values
`by`	Specify a mode for splitting your data, that is, to split by counting the number of characters, words, or vocabulary tokens. **Valid values**: • `characters` (or `chars`): Splits by counting the number of characters. • `words`: Splits by counting the number of words. Words are defined as sequences of alphabetic characters, sequences of digits, individual punctuation marks, or symbols. For segmented languages without white space word boundaries (such as Chinese, Japanese, or Thai), each native character is considered a word (that is, unigram). • `vocabulary`: Splits by counting the number of vocabulary tokens. Vocabulary tokens are words or word pieces, recognized by the vocabulary of the tokenizer that your embedding model uses. You can load your vocabulary file using the chunker helper API `DBMS_VECTOR_CHAIN.CREATE_VOCABULARY`. **Note**: For accurate results, ensure that the chosen model matches the vocabulary file used for chunking. If you are not using a vocabulary file, then ensure that the input length is defined within the token limits of your model. **Default value**: `words`
`max`	Specify a limit on the maximum size of each chunk. This setting splits the input text at a fixed point where the maximum limit occurs in the larger text. The units of `max` correspond to the `by` mode, that is, to split data when it reaches the maximum size limit of a certain number of characters, words, numbers, punctuation marks, or vocabulary tokens. **Valid values**: • `by characters`: `50` to `4000` characters • `by words`: `10` to `1000` words • `by vocabulary`: `10` to `1000` tokens **Default value**: `100`

Parameter	Description and Acceptable Values

split [by]	Specify where to split the input text when it reaches the maximum size limit. This helps to keep related data together by defining appropriate boundaries for chunks.
	Valid values:
	• none:
	Splits at the max limit of characters, words, or vocabulary tokens.
	• newline, blankline, and space:
	These are single-split character conditions that split at the last split character before the max value.
	Use newline to split at the end of a line of text. Use blankline to split at the end of a blank line (sequence of characters, such as two newlines). Use space to split at the end of a blank space.
	• recursively:
	This is a multiple-split character condition that breaks the input text using an ordered list of characters (or sequences).
	recursively is predefined as BLANKLINE, newline, space, none in this order:
	1. If the input text is more than the max value, then split by the first split character.
	2. If that fails, then split by the second split character.
	3. And so on.
	4. If no split characters exist, then split by max wherever it appears in the text.
	• sentence:
	This is an end-of-sentence split condition that breaks the input text at a sentence boundary.
	This condition automatically determines sentence boundaries by using knowledge of the input language's sentence punctuation and contextual rules. This language-specific condition relies mostly on end-of-sentence (EOS) punctuations and common abbreviations.
	Contextual rules are based on word information, so this condition is only valid when splitting the text by words or vocabulary (not by characters).
	Note: This condition obeys the by word and max settings, and thus may not determine accurate sentence boundaries in some cases. For example, when a sentence is larger than the max value, it splits the sentence at max. Similarly, it includes multiple sentences in the text only when they fit within the max limit.
	• custom:
	Splits based on a custom split characters list. You can provide custom sequences up to a limit of 16 split character strings, with a maximum length of 10 each.
	Specify an array of valid text literals using the custom_list parameter.
	`{ "split" : "custom", "custom_list" : [ "split_chars1", ... ] }`
	For example:
	`{ "split" : "custom", "custom_list" : [ "<p>" , "<s>" ] }`
	Note: You can omit sequences only for tab (\t), newline (\n), and linefeed (\r).
	Default value: recursively

Parameter	Description and Acceptable Values
`overlap`	Specify the amount (as a positive integer literal or zero) of the preceding text that the chunk should contain, if any. This helps in logically splitting up related text (such as a sentence) by including some amount of the preceding chunk text. The amount of overlap depends on how the maximum size of the chunk is measured (in characters, words, or vocabulary tokens). The overlap begins at the specified `split` condition (for example, at `newline`). **Valid value**: `5%` to `20%` of `max` **Default value**: `0`

`language`	Specify the language of your input data.
	This clause is important, especially when your text contains certain characters (for example, punctuations or abbreviations) that may be interpreted differently in another language.
	Valid values:
	• NLS-supported language name or its abbreviation, as listed in *Oracle Database Globalization Support Guide*.
	• Custom language name or its abbreviation, as listed in Supported Languages and Data File Locations. You use the `DBMS_VECTOR_CHAIN.CREATE_LANG_DATA` chunker helper API to load language-specific data (abbreviation tokens) into the database, for your specified language.
	Note: You must use escape characters with any language abbreviation that is also a SQL reserved word (for example, language abbreviations such as `IN`, `AS`, `OR`, `IS`).
	For example:
	`SELECT dbms_vector_chain.utl_to_chunks('this is an example', JSON('{ "language" : "\"in\"" }')) from dual;`
	`SELECT dbms_vector_chain.utl_to_chunks('this is an example', JSON_OBJECT('language' value '"in"' RETURNING JSON)) from dual;`
	Default value: `NLS_LANGUAGE` from session

Parameter	Description and Acceptable Values

normalize	Automatically pre-processes or post-processes issues (such as multiple consecutive spaces and smart quotes) that may arise when documents are converted into text. Oracle recommends you to use a normalization mode to extract high-quality chunks.
	Valid values:
	• none:
	Applies no normalization.
	• all:
	Normalizes common multi-byte (unicode) punctuation to standard single-byte.
	• options:
	Specify an array of normalization options using the norm_options parameter.
	`{ "normalize" : "options", "norm_options" : [ "normalize_option1", ... ] }`
	– punctuation:
	Converts quotes, dashes, and other punctuation characters supported in the character set of the text to their common ASCII form. For example:
	* U+2013 (En Dash) maps to U+002D (Hyphen-Minus)
	* U+2018 (Left Single Quotation Mark) maps to U+0027 (Apostrophe)
	* U+2019 (Right Single Quotation Mark) maps to U+0027 (Apostrophe)
	* U+201B (Single High-Reversed-9 Quotation Mark) maps to U+0027 (Apostrophe)
	– whitespace:
	Minimizes whitespace by eliminating unnecessary characters.
	For example, retain blank lines, but remove any extra newlines and interspersed spaces or tabs: `" \n \n " => "\n\n"`
	– widechar:
	Normalizes wide, multi-byte digits and (a-z) letters to single-byte.
	These are multi-byte equivalents for 0-9 and a-z A-Z, which can show up in Chinese, Japanese, or Korean text.
	For example:
	`{ "normalize" : "options", "norm_options" : [ "whitespace" ] }`
	Default value: none

extended	Increases the output limit of a VARCHAR2 string to 32767 bytes, without requiring you to set the max_string_size parameter to extended. **Default value**: 4000 or 32767 (when max_string_size=extended)

Example

```
SELECT D.id doc,
    JSON_VALUE(C.column_value, '$.chunk_id' RETURNING NUMBER) AS id,
    JSON_VALUE(C.column_value, '$.chunk_offset' RETURNING NUMBER) AS pos,
    JSON_VALUE(C.column_value, '$.chunk_length' RETURNING NUMBER) AS siz,
    JSON_VALUE(C.column_value, '$.chunk_data') AS txt
```

```
FROM docs D,
   dbms_vector.utl_to_chunks(D.text,
   JSON('{ "by"       : "words",
           "max"      : "100",
           "overlap"  : "0",
           "split"    : "recursively",
           "language" : "american",
           "normalize": "all" }')) C;
```

End-to-end examples:

To run end-to-end example scenarios using this function, see Perform Chunking With Embedding and Configure Chunking Parameters.

Related Topics

- VECTOR_CHUNKS

UTL_TO_EMBEDDING and UTL_TO_EMBEDDINGS

Use the `DBMS_VECTOR.UTL_TO_EMBEDDING` and `DBMS_VECTOR.UTL_TO_EMBEDDINGS` chainable utility functions to generate one or more vector embeddings from textual documents and images.

Purpose

To automatically generate one or more vector embeddings from textual documents and images.

- Text to Vector:

  You can perform a text-to-embedding transformation by accessing either Oracle Database or a third-party service provider:

  - Oracle Database as the service provider (default setting):

    This API calls an ONNX format embedding model that you load into the database.

  - Third-party embedding model:

    This API makes a REST API call to your chosen remote service provider (Cohere, Generative AI, Google AI, Hugging Face, OpenAI, or Vertex AI) or local service provider (Ollama).

- Image to Vector:

  You can also perform an image-to-embedding transformation. This API makes a REST call to your chosen image embedding model or multimodal embedding model by Vertex AI. Note that currently Vertex AI is the only supported service provider for this operation.

> **WARNING:**
>
> Certain features of the database may allow you to access services offered separately by third-parties, for example, through the use of JSON specifications that facilitate your access to REST APIs.
>
> Your use of these features is solely at your own risk, and you are solely responsible for complying with any terms and conditions related to use of any such third-party services. Notwithstanding any other terms and conditions related to the third-party services, your use of such database features constitutes your acceptance of that risk and express exclusion of Oracle's responsibility or liability for any damages resulting from such access.

Syntax

- Text to Vector:

```
DBMS_VECTOR.UTL_TO_EMBEDDING (
    DATA           IN CLOB,
    PARAMS         IN JSON default NULL
) return VECTOR;


DBMS_VECTOR.UTL_TO_EMBEDDINGS (
    DATA           IN VECTOR_ARRAY_T,
    PARAMS         IN JSON default NULL
) return VECTOR_ARRAY_T;
```

- Image to Vector:

```
DBMS_VECTOR.UTL_TO_EMBEDDING (
    DATA           IN BLOB,
    MODALITY       IN VARCHAR2,
    PARAMS         IN JSON default NULL
) return VECTOR;
```

DATA

- Text to Vector:

  `UTL_TO_EMBEDDING` accepts the input as `CLOB` containing textual data (text strings or small documents). It then converts the text to a single embedding (`VECTOR`).

  `UTL_TO_EMBEDDINGS` converts an array of chunks (`VECTOR_ARRAY_T`) to an array of embeddings (`VECTOR_ARRAY_T`).

  > **Note:**
  >
  > Although data is a `CLOB` or a `VECTOR_ARRAY_T` of `CLOB`, the maximum input is 4000 characters. If you have input that is greater, you can use `UTL_TO_CHUNKS` to split the data into smaller chunks before passing in.

- Image to Vector:

`UTL_TO_EMBEDDING` accepts the input as `BLOB` containing media data for media files such as images. It then converts the image input to a single embedding (`VECTOR`).

A generated embedding output includes:

```
{
    "embed_id"    :  NUMBER,
    "embed_data"  : "VARCHAR2(4000)",
    "embed_vector": "CLOB"
}
```

Where,

- `embed_id` displays the ID number of each embedding.
- `embed_data` displays the input text that is transformed into embeddings.
- `embed_vector` displays the generated vector representations.

MODALITY

For `BLOB` inputs, specify the type of content to vectorize. The only supported value is `image`.

PARAMS

Specify input parameters in JSON format, depending on the service provider that you want to use.

If using Oracle Database as the provider:

```
{
  "provider" : "database",
  "model"    : "<in-database ONNX embedding model filename>"
}
```

Table 12-12 Database Provider Parameter Details

Parameter	Description
`provider`	Specify `database` (default setting) to use Oracle Database as the provider. With this setting, you must load an ONNX format embedding model into the database.
`model`	User-specified name under which the imported ONNX embedding model is stored in Oracle Database. If you do not have an embedding model in ONNX format, then perform the steps listed in Convert Pretrained Models to ONNX Format.

If using a third-party provider:

Set the following parameters along with additional embedding parameters specific to your provider:

- For `UTL_TO_EMBEDDING`:

```
{
  "provider"        : "<AI service provider>",
  "credential_name" : "<credential name>",
  "url"             : "<REST endpoint URL for embedding service>",
```

```
  "model"           : "<REST provider embedding model name>",
  "transfer_timeout": <maximum wait time for the request to complete>,
  "max_count": "<maximum calls to the AI service provider>",
  "<additional REST provider parameter>": "<REST provider parameter
value>"
}
```

- For `UTL_TO_EMBEDDINGS`:

```
{
  "provider"        : "<AI service provider>",
  "credential_name" : "<credential name>",
  "url"             : "<REST endpoint URL for embedding service>",
  "model"           : "<REST provider embedding model name>",
  "transfer_timeout": <maximum wait time for the request to complete>,
  "batch_size"      : "<number of vectors to request at a time>",
  "max_count": "<maximum calls to the AI service provider>",
  "<additional REST provider parameter>": "<REST provider parameter
value>"
}
```

Table 12-13 Third-Party Provider Parameter Details

Parameter	Description
`provider`	Third-party service provider that you want to access for this operation. A REST call is made to the specified provider to access its embedding model. For image input, specify `vertexai`. For text input, specify one of the following values: • `cohere` • `googleai` • `huggingface` • `ocigenai` • `openai` • `vertexai`
`credential_name`	Name of the credential in the form: *`schema.credential_name`* A credential name holds authentication credentials to enable access to your provider for making REST API calls. You need to first set up your credential by calling the `DBMS_VECTOR.CREATE_CREDENTIAL` helper function to create and store a credential, and then refer to the credential name here. See CREATE_CREDENTIAL.
`url`	URL of the third-party provider endpoint for each REST call, as listed in Supported Third-Party Provider Operations and Endpoints.

Table 12-13 (Cont.) Third-Party Provider Parameter Details

Parameter	Description
`model`	Name of the third-party embedding model in the form: *`schema.model_name`* If you do not specify a schema, then the schema of the procedure invoker is used. **Note**: • For Generative AI, all the supported third-party models are listed in Supported Third-Party Provider Operations and Endpoints. • For accurate results, ensure that the chosen text embedding model matches the vocabulary file used for chunking. If you are not using a vocabulary file, then ensure that the input length is defined within the token limits of your model. • To get image embeddings, you can use any image embedding model or multimodal embedding model supported by Vertex AI. Multimodal embedding is a technique that vectorizes data from different modalities such as text and images. When using a multimodal embedding model to generate embeddings, ensure that you use the same model to vectorize both types of content (text and images). By doing so, the resulting embeddings are compatible and situated in the same vector space, which allows for effective comparison between the two modalities during similarity searches.
`transfer_timeout`	Maximum time to wait for the request to complete. The default value is `60` seconds. You can increase this value for busy web servers.
`batch_size`	Maximum number of vectors to request at a time. For example, for a batch size of `50`, if `100` chunks are passed, then this API sends two requests with an array of `50` strings each. If `30` chunks are passed (which is lesser than the defined batch size), then the API sends those in a single request. For REST calls, it is more efficient to send a batch of inputs at a time rather than requesting a single input per call. Increasing the batch size can provide better performance, whereas reducing the batch size may reduce memory and data usage, especially if your provider has a rate limit. The default or maximum allowed value depends on the third-party provider settings.
`max_count`	Maximum number of times the API can be called for a given third-party provider. When set to an integer *n*, `max_count` stops the execution of the API for the given provider beyond *n* times. This prevents accidental calling of a third-party over some limit, for example to avoid surpassing the service amount that you have purchased.

Additional third-party provider parameters:

Optionally, specify additional provider-specific parameters.

Table 12-14 Additional REST Provider Parameter Details

Parameter	Description
`input_type`	Type of input to vectorize.

Let us look at some example configurations for all third-party providers:

Important:

- The following examples are for illustration purposes. For accurate and up-to-date information on the parameters to use, refer to your third-party provider's documentation.
- For a list of all supported REST endpoint URLs, see Supported Third-Party Provider Operations and Endpoints.
- The generated embedding results may be different between requests for the same input and configuration, depending on your embedding model or floating point precision. However, this does not affect your queries (and provides semantically correct results) because the vector distance will be similar.

Cohere example:

```
{
  "provider"       : "cohere",
  "credential_name": "COHERE_CRED",
  "url"            : "https://api.cohere.example.com/embed",
  "model"          : "embed-english-light-v2.0",
  "input_type"     : "search_query"
}
```

Generative AI example:

```
{
  "provider"       : "ocigenai",
  "credential_name": "OCI_CRED",
  "url"            : "https://generativeai.oci.example.com/embedText",
  "model"          : "cohere.embed-english-v3.0",
  "batch_size"     : 10
}
```

Google AI example:

```
{
  "provider"       : "googleai",
  "credential_name": "GOOGLEAI_CRED",
  "url"            : "https://googleapis.example.com/models/",
  "model"          : "embedding-001",
  "max_count"      : 500
}
```

Hugging Face example:

```
{
  "provider"       : "huggingface",
  "credential_name": "HF_CRED",
  "url"            : "https://api.huggingface.example.com/",
  "model"          : "sentence-transformers/all-MiniLM-L6-v2"
}
```

Ollama example:

```
{
  "provider"       : "ollama",
  "host"           : "local",
  "url"            : "http://localhost:11434/api/embeddings",
  "model"          : "phi3:mini"
}
```

OpenAI example:

```
{
  "provider"       : "openai",
  "credential_name": "OPENAI_CRED",
  "url"            : "https://api.openai.example.com/embeddings",
  "model"          : "text-embedding-3-small"
}
```

Vertex AI example:

```
{
  "provider"       : "vertexai",
  "credential_name": "VERTEXAI_CRED",
  "url"            : "https://googleapis.example.com/models/",
  "model"          : "textembedding-gecko:predict"
}
```

Examples

You can use `UTL_TO_EMBEDDING` in a `SELECT` clause and `UTL_TO_EMBEDDINGS` in a `FROM` clause, as follows:

UTL_TO_EMBEDDING:

- **Text to vector using Generative AI**:

  The following examples use `UTL_TO_EMBEDDING` to generate an embedding with `Hello world` as the input.

  Here, the cohere.embed-english-v3.0 model is used by accessing Generative AI as the provider. You can replace the `model` value with any other supported model that you want to use with Generative AI, as listed in Supported Third-Party Provider Operations and Endpoints.

  ```
  -- declare embedding parameters

  var params clob;

  begin
   :params := '
  {
    "provider": "ocigenai",
    "credential_name": "OCI_CRED",
    "url": "https://inference.generativeai.us-chicago-1.oci.oraclecloud.com/
  20231130/actions/embedText",
    "model": "cohere.embed-english-v3.0",
  ```

```
  "batch_size": 10
}';
end;
/

-- get text embedding: PL/SQL example

declare
  input clob;
  v vector;
begin
  input := 'Hello world';

  v := dbms_vector.utl_to_embedding(input, json(params));
  dbms_output.put_line(vector_serialize(v));
exception
  when OTHERS THEN
    DBMS_OUTPUT.PUT_LINE (SQLERRM);
    DBMS_OUTPUT.PUT_LINE (SQLCODE);
end;
/

-- get text embedding: select example

select dbms_vector.utl_to_embedding('Hello world', json(:params)) from
dual;
```

- **Image to vector using Vertex AI**:

  The following examples use `UTL_TO_EMBEDDING` to generate an embedding by accessing the Vertex AI's multimodal embedding model.

  Here, the input is `parrots.jpg`, `VEC_DUMP` is a local directory that stores the `parrots.jpg` file, and the modality is specified as `image`.

```
-- declare embedding parameters

var params clob;

begin
  :params := '
{
  "provider": "vertexai",
  "credential_name": "VERTEXAI_CRED",
  "url": "https://LOCATION-aiplatform.googleapis.com/v1/projects/PROJECT/
locations/LOCATION/publishers/google/models/",
  "model": "multimodalembedding:predict"
}';
end;
/

-- get image embedding: PL/SQL example

declare
  v vector;
  output clob;
begin
```

```
  v := dbms_vector.utl_to_embedding(
    to_blob(bfilename('VEC_DUMP', 'parrots.jpg')), 'image', json(:params));
  output := vector_serialize(v);
  dbms_output.put_line('vector data=' || dbms_lob.substr(output, 100) ||
'...');
end;
/

-- get image embedding: select example

select dbms_vector.utl_to_embedding(
  to_blob(bfilename('VEC_DUMP', 'parrots.jpg')), 'image', json(:params));
```

- **Text to vector using in-database embedding model**:

  The following example uses `UTL_TO_EMBEDDING` to generate a vector embedding by calling an ONNX format embedding model (`doc_model`) loaded into Oracle Database.

  Here, the provider is `database`, and the input is `hello`.

  ```
  var params clob;
  exec :params := '{"provider":"database", "model":"doc_model"}';

  select dbms_vector.utl_to_embedding('hello', json(:params)) from dual;
  ```

  For complete example, see Convert Text String to Embedding Within Oracle Database.

- **End-to-end examples**:

  To run various end-to-end example scenarios using `UTL_TO_EMBEDDING`, see Generate Embedding.

UTL_TO_EMBEDDINGS:

- **Text to vector using in-database embedding model**:

  The following example uses `UTL_TO_EMBEDDINGS` to generate an array of embeddings by calling an ONNX format embedding model (`doc_model`) loaded into Oracle Database.

  Here, the provider is `database`, and the input is a PDF document stored in the `documentation_tab` table. As you can see, you first use `UTL_TO_CHUNKS` to split the data into smaller chunks before passing in to `UTL_TO_EMBEDDINGS`.

  ```
  CREATE TABLE doc_chunks as
  (select dt.id doc_id, et.embed_id, et.embed_data,
  to_vector(et.embed_vector) embed_vector
   from
     documentation_tab dt,
     dbms_vector.utl_to_embeddings(
         dbms_vector.utl_to_chunks(dbms_vector.utl_to_text(dt.data),
  json('{"normalize":"all"}')),
         json('{"provider":"database", "model":"doc_model"}')) t,
     JSON_TABLE(t.column_value, '$[*]' COLUMNS (embed_id NUMBER PATH
  '$.embed_id', embed_data VARCHAR2(4000) PATH '$.embed_data', embed_vector
  CLOB PATH '$.embed_vector')) et
  );
  ```

For complete example, see SQL Quick Start Using a Vector Embedding Model Uploaded into the Database.

- **End-to-end examples**:

  To run various end-to-end example scenarios using `UTL_TO_EMBEDDINGS`, see Perform Chunking With Embedding.

UTL_TO_GENERATE_TEXT

Use the `DBMS_VECTOR.UTL_TO_GENERATE_TEXT` chainable utility function to generate a text response for a given prompt or an image, by accessing third-party text generation models.

Purpose

To communicate with Large Language Models (LLMs) through natural language conversations. You can generate a textual answer, description, or summary for prompts and images, given as input to LLM-powered chat interfaces.

- Prompt to Text:

  A prompt can be an input text string, such as a question that you ask an LLM. For example, "`What is Oracle Text?`". A prompt can also be a command, such as "`Summarize the following ...`", "`Draft an email asking for ...`", or "`Rewrite the following ...`", and can include results from a search. The LLM responds with a textual answer or description based on the specified task in the prompt.

  For this operation, this API makes a REST call to your chosen remote third-party provider (Cohere, Generative AI, Google AI, Hugging Face, OpenAI, or Vertex AI) or local third-party provider (Ollama).

- Image to Text:

  You can also prompt with a media file, such as an image, to extract text from pictures or photos. You supply a text question as the prompt (such as "`What is this image about?`" or "`How many birds are there in this painting?`") along with the image. The LLM responds with a textual analysis or description of the contents of the image.

  For this operation, this API makes a REST call to your chosen remote third-party provider (Google AI, Hugging Face, OpenAI, or Vertex AI) or local third-party provider (Ollama).

> **WARNING:**
>
> Certain features of the database may allow you to access services offered separately by third-parties, for example, through the use of JSON specifications that facilitate your access to REST APIs.
>
> Your use of these features is solely at your own risk, and you are solely responsible for complying with any terms and conditions related to use of any such third-party services. Notwithstanding any other terms and conditions related to the third-party services, your use of such database features constitutes your acceptance of that risk and express exclusion of Oracle's responsibility or liability for any damages resulting from such access.

Syntax

This function accepts the input as `CLOB` containing text data (for textual prompts) or as `BLOB` containing media data (for media files such as images). It then processes this information to generate a new `CLOB` containing the generated text.

- Prompt to Text:

```
DBMS_VECTOR.UTL_TO_GENERATE_TEXT (
        DATA          IN CLOB,
        PARAMS        IN JSON default NULL
) return CLOB;
```

- Image to Text:

```
DBMS_VECTOR.UTL_TO_GENERATE_TEXT(
          TEXT_DATA      IN CLOB,
          MEDIA_DATA     IN BLOB,
          MEDIA_TYPE     IN VARCHAR2 default 'image/jpeg',
          PARAMS         IN JSON default NULL
) return CLOB;
```

DATA and TEXT_DATA

Specify the textual prompt as `CLOB` for the `DATA` or `TEXT_DATA` clause.

> **Note:**
>
> Hugging Face uses an image captioning model that does not require a prompt, when giving an image as input. If you input a prompt along with an image, then the prompt will be ignored.

MEDIA_DATA

Specify the `BLOB` file, such as an image or a visual PDF file.

MEDIA_TYPE

Specify the image format for the given image or visual PDF file (`BLOB` file) in one of the supported image data MIME types. For example:

- For PNG: `image/png`
- For JPEG: `image/jpeg`
- For PDF: `application/pdf`

> **Note:**
>
> For a complete list of the supported image formats, refer to your third-party provider's documentation.

PARAMS

Specify the following input parameters in JSON format, depending on the service provider that you want to access for text generation:

```
{
  "provider"         : "<AI service provider>",
  "credential_name"  : "<credential name>",
  "url"              : "<REST endpoint URL for text generation service>",
  "model"            : "<text generation model name>",
  "transfer_timeout" : <maximum wait time for the request to complete>,
  "max_count": "<maximum calls to the AI service provider>",
  "<additional REST provider parameter>": "<REST provider parameter value>"
}
```

Table 12-15 UTL_TO_GENERATE_TEXT Parameter Details

Parameter	Description
`provider`	Supported REST provider that you want to access to generate text. Specify one of the following values: For `CLOB` input: • `cohere` • `googleai` • `huggingface` • `ocigenai` • `openai` • `vertexai` For `BLOB` input: • `googleai` • `huggingface` • `openai` • `vertexai`
`credential_name`	Name of the credential in the form: *`schema.credential_name`* A credential name holds authentication credentials to enable access to your provider for making REST API calls. You need to first set up your credential by calling the `DBMS_VECTOR.CREATE_CREDENTIAL` helper function to create and store a credential, and then refer to the credential name here. See CREATE_CREDENTIAL.
`url`	URL of the third-party provider endpoint for each REST call, as listed in Supported Third-Party Provider Operations and Endpoints.
`model`	Name of the third-party text generation model in the form: *`schema.model_name`* If the model name is not schema-qualified, then the schema of the procedure invoker is used. **Note**: For Generative AI, all the supported third-party models are listed in Supported Third-Party Provider Operations and Endpoints.
`transfer_timeout`	Maximum time to wait for the request to complete. The default value is `60` seconds. You can increase this value for busy web servers.

Table 12-15 (Cont.) UTL_TO_GENERATE_TEXT Parameter Details

Parameter	Description
`max_count`	Maximum number of times the API can be called for a given third-party provider. When set to an integer *n*, `max_count` stops the execution of the API for the given provider beyond *n* times. This prevents accidental calling of a third-party over some limit, for example to avoid surpassing the service amount that you have purchased.

Additional third-party provider parameters:

Optionally, specify additional provider-specific parameters.

Table 12-16 Additional REST Provider Parameter Details

Parameter	Description
`max_tokens`	Maximum number of tokens in the output text.
`temperature`	Degree of randomness used when generating the output text, in the range of `0.0-5.0`. To generate the same output for a prompt, use `0`. To generate a random new text for that prompt, increase the temperature. **Note**: Start with the temperature set to `0`. If you do not require random results, a recommended temperature value is between `0` and `1`. A higher value is not recommended because a high temperature may produce creative text, which might also include hallucinations.
`topP`	Probability of tokens in the output, in the range of `0.0-1.0`. A lower value provides less random responses and a higher value provides more random responses.
`candidateCount`	Number of response variations to return, in the range of `1-4`.
`maxOutputTokens`	Maximum number of tokens to generate for each response.

Let us look at some example configurations for all third-party providers:

Important:

- The following examples are for illustration purposes. For accurate and up-to-date information on additional parameters to use, refer to your third-party provider's documentation.
- For a list of all supported REST endpoint URLs, see Supported Third-Party Provider Operations and Endpoints.

Cohere example:

```
{
  "provider"       : "cohere",
  "credential_name": "COHERE_CRED",
  "url"            : "https://api.cohere.example.com/chat",
```

```
  "model"           : "command"
}
```

Generative AI example:

> **Note:**
>
> For Generative AI, if you want to pass any additional REST provider-specific parameters, then you must enclose those in `chatRequest`.

```
{
  "provider"       : "ocigenai",
  "credential_name": "OCI_CRED",
  "url"            : "https://inference.generativeai.us-example.com/chat",
  "model"          : "cohere.command-r-16k",
  "chatRequest"    : {
                      "maxTokens"    : 256
                     }
}
```

Google AI example:

```
{
  "provider"        : "googleai",
  "credential_name" : "GOOGLEAI_CRED",
  "url"             : "https://googleapis.example.com/models/",
  "model"           : "gemini-pro:generateContent"
}
```

Hugging Face example:

```
{
  "provider"        : "huggingface",
  "credential_name" : "HF_CRED",
  "url"             : "https://api.huggingface.example.com/models/",
  "model"           : "gpt2"
}
```

Ollama example:

```
{
  "provider"       : "ollama",
  "host"           : "local",
  "url"            : "http://localhost:11434/api/generate",
  "model"          : "phi3:mini"
}
```

OpenAI example:

```
{
  "provider"        : "openai",
  "credential_name" : "OPENAI_CRED",
  "url"             : "https://api.openai.example.com",
  "model"           : "gpt-4o-mini",
  "max_tokens"      : 60,
  "temperature"     : 1.0
}
```

Vertex AI example:

```
{
  "provider"         : "vertexai",
  "credential_name"  : "VERTEXAI_CRED",
  "url"              : "https://googleapis.example.com/models/",
  "model"            : "gemini-1.0-pro:generateContent",
  "generation_config": {
                        "temperature"    : 0.9,
                        "topP"           : 1,
                        "candidateCount" : 1,
                        "maxOutputTokens": 256
                       }
}
```

Examples

- Prompt to Text:

  The following statements generate a text response by making a REST call to Generative AI. The prompt given here is "`What is Oracle Text?`".

  Here, the cohere.command-r-16k and meta.llama-3.1-70b-instruct models are used. You can replace the `model` value with any other supported model that you want to use with Generative AI, as listed in Supported Third-Party Provider Operations and Endpoints.

  Using the cohere.command-r-16k model:

  ```
  -- select example

  var params clob;
  exec :params := '
  {
    "provider"       : "ocigenai",
    "credential_name": "OCI_CRED",
    "url"            : "https://inference.generativeai.us-
  chicago-1.oci.oraclecloud.com/20231130/actions/chat",
    "model"          : "cohere.command-r-16k",
    "chatRequest"    : {
                         "maxTokens": 256
                       }
  }';

  select dbms_vector.utl_to_generate_text(
   'What is Oracle Text?',
  ```

```
 json(:params)) from dual;

-- PL/SQL example

declare
  input clob;
  params clob;
  output clob;
begin
  input := 'What is Oracle Text?';

  params := '
{
  "provider"       : "ocigenai",
  "credential_name": "OCI_CRED",
  "url"            : "https://inference.generativeai.us-
chicago-1.oci.oraclecloud.com/20231130/actions/chat",
  "model"          : "cohere.command-r-16k",
  "chatRequest"    : {
                      "maxTokens": 256
                     }
}';

  output := dbms_vector.utl_to_generate_text(input, json(params));
  dbms_output.put_line(output);
  if output is not null then
    dbms_lob.freetemporary(output);
  end if;
exception
  when OTHERS THEN
    DBMS_OUTPUT.PUT_LINE (SQLERRM);
    DBMS_OUTPUT.PUT_LINE (SQLCODE);
end;
/
```

Using the meta.llama-3.1-70b-instruct model:

```
-- select example

var params clob;
exec :params := '
{
   "provider"       : "ocigenai",
   "credential_name": "OCI_CRED",
   "url"            : "https://inference.generativeai.us-
chicago-1.oci.oraclecloud.com/20231130/actions/chat",
   "model"          : "meta.llama-3.1-70b-instruct",
   "chatRequest"    : {
                       "topK" : 1
                      }
}';

select dbms_vector.utl_to_generate_text(
 'What is Oracle Text?',
 json(:params)) from dual;
```

```
-- PL/SQL example

declare
  input clob;
  params clob;
  output clob;
begin
  input := 'What is Oracle Text?';

  params := '
{
   "provider"       : "ocigenai",
   "credential_name": "OCI_CRED",
   "url"            : "https://inference.generativeai.us-
chicago-1.oci.oraclecloud.com/20231130/actions/chat",
   "model"          : "meta.llama-3.1-70b-instruct",
   "chatRequest"    : {
                       "topK" : 1
                      }
}';

  output := dbms_vector.utl_to_generate_text(input, json(params));
  dbms_output.put_line(output);
  if output is not null then
    dbms_lob.freetemporary(output);
  end if;
exception
  when OTHERS THEN
    DBMS_OUTPUT.PUT_LINE (SQLERRM);
    DBMS_OUTPUT.PUT_LINE (SQLCODE);
end;
/
```

End-to-end examples:

To run end-to-end example scenarios, see Generate Text Response.

- Image to Text:

  The following statements generate a text response by making a REST call to OpenAI. Here, the input is an image (`sample_image.jpeg`) along with the prompt "`Describe this image?`".

```
-- select example

var input clob;
var media_data blob;
var media_type clob;
var params clob;

begin
  :input := 'Describe this image';
  :media_data := load_blob_from_file('DEMO_DIR', 'sample_image.jpeg');
  :media_type := 'image/jpeg';
  :params := '
{
```

```
  "provider"       : "openai",
  "credential_name": "OPENAI_CRED",
  "url"            : "https://api.openai.com/v1/chat/completions",
  "model"          : "gpt-4o-mini",
  "max_tokens"     : 60
}';
end;
/

select dbms_vector.utl_to_generate_text(:input, :media_data, :media_type,
json(:params));

-- PL/SQL example

declare
  input clob;
  media_data blob;
  media_type varchar2(32);
  params clob;
  output clob;

begin
  input := 'Describe this image';
  media_data := load_blob_from_file('DEMO_DIR', 'image_file');
  media_type := 'image/jpeg';
  params := '
{
  "provider"       : "openai",
  "credential_name": "OPENAI_CRED",
  "url"            : "https://api.openai.com/v1/chat/completions",
  "model"          : "gpt-4o-mini",
  "max_tokens"     : 60
}';

  output := dbms_vector.utl_to_generate_text(
    input, media_data, media_type, json(params));
  dbms_output.put_line(output);

  if output is not null then
    dbms_lob.freetemporary(output);
  end if;
  if media_data is not null then
    dbms_lob.freetemporary(media_data);
  end if;
exception
  when OTHERS THEN
    DBMS_OUTPUT.PUT_LINE (SQLERRM);
    DBMS_OUTPUT.PUT_LINE (SQLCODE);
end;
/
```

End-to-end examples:

To run end-to-end example scenarios, see Describe Image Content.

DBMS_VECTOR_CHAIN

The DBMS_VECTOR_CHAIN package enables advanced operations with Oracle AI Vector Search, such as chunking and embedding data along with text generation and summarization capabilities. It is more suitable for text processing with similarity search and hybrid search, using functionality that can be pipelined together for an end-to-end search.

This table lists the DBMS_VECTOR_CHAIN subprograms and briefly describes them.

Table 12-17 DBMS_VECTOR_CHAIN Package Subprograms

Subprogram	Description
Chainable Utility (UTL) Functions:	
These functions are a set of modular and flexible functions within vector utility PL/SQL packages. You can chain these together to automate end-to-end data transformation and similarity search operations.	
UTL_TO_TEXT	Extracts plain text data from documents
UTL_TO_CHUNKS	Splits data into smaller pieces or chunks
UTL_TO_EMBEDDING and UTL_TO_EMBEDDINGS	Converts text or an image to one or more vector embeddings
UTL_TO_SUMMARY	Extracts a summary from documents
UTL_TO_GENERATE_TEXT	Generates text for a prompt (input string) or an image
Credential Helper Procedures:	
These procedures enable you to securely manage authentication credentials in the database. You require these credentials to enable access to third-party service providers for making REST calls.	
CREATE_CREDENTIAL	Creates a credential name
DROP_CREDENTIAL	Drops an existing credential name
Preference Helper Procedures:	
These procedures enable you to manage vectorizer preferences, to be used with the CREATE_HYBRID_VECTOR_INDEX and ALTER_INDEX SQL statements when creating or managing hybrid vector indexes.	
CREATE_PREFERENCE	Creates a vectorizer preference
DROP_PREFERENCE	Drops an existing vectorizer preference
Chunker Helper Procedures:	
These procedures enable you to configure vocabulary and language data (abbreviations), to be used with the VECTOR_CHUNKS SQL function or UTL_TO_CHUNKS PL/SQL function.	
CREATE_VOCABULARY	Loads your token vocabulary file into the database
DROP_VOCABULARY	Removes existing vocabulary data
CREATE_LANG_DATA	Loads your language data file into the database
DROP_LANG_DATA	Removes existing abbreviation data
Data Access Function:	
This function enables you to enhance search operations.	
RERANK	Reorders search results for a more relevant output

> **Note:**
>
> The `DBMS_VECTOR_CHAIN` package requires you to install the `CONTEXT` component of Oracle Text, an Oracle Database technology that provides indexing, term extraction, text analysis, text summarization, word and theme searching, and other utilities.
>
> Due to underlying dependance on the text processing capabilities of Oracle Text, note that both the `UTL_TO_TEXT` and `UTL_TO_SUMMARY` chainable utility functions and all the chunker helper procedures are available only in this package through Oracle Text.

- CREATE_CREDENTIAL
  Use the `DBMS_VECTOR_CHAIN.CREATE_CREDENTIAL` credential helper procedure to create a credential name for storing user authentication details in Oracle Database.
- CREATE_LANG_DATA
  Use the `DBMS_VECTOR_CHAIN.CREATE_LANG_DATA` chunker helper procedure to load your own language data file into the database.
- CREATE_PREFERENCE
  Use the `DBMS_VECTOR_CHAIN.CREATE_PREFERENCE` helper procedure to create a vectorizer preference, to be used when creating or updating hybrid vector indexes.
- CREATE_VOCABULARY
  Use the `DBMS_VECTOR_CHAIN.CREATE_VOCABULARY` chunker helper procedure to load your own token vocabulary file into the database.
- DROP_CREDENTIAL
  Use the `DBMS_VECTOR_CHAIN.DROP_CREDENTIAL` credential helper procedure to drop an existing credential name from the data dictionary.
- DROP_LANG_DATA
  Use the `DBMS_VECTOR_CHAIN.DROP_LANG_DATA` chunker helper procedure to remove abbreviation data from the data dictionary.
- DROP_PREFERENCE
  Use the `DBMS_VECTOR_CHAIN.DROP_PREFERENCE` preference helper procedure to remove an existing Vectorizer preference.
- DROP_VOCABULARY
  Use the `DBMS_VECTOR_CHAIN.DROP_VOCABULARY` chunker helper procedure to remove vocabulary data from the data dictionary.
- RERANK
  Use the `DBMS_VECTOR_CHAIN.RERANK` function to reassess and reorder an initial set of results to retrieve more relevant search output.
- UTL_TO_CHUNKS
  Use the `DBMS_VECTOR_CHAIN.UTL_TO_CHUNKS` chainable utility function to split a large plain text document into smaller chunks of text.
- UTL_TO_EMBEDDING and UTL_TO_EMBEDDINGS
  Use the `DBMS_VECTOR_CHAIN.UTL_TO_EMBEDDING` and `DBMS_VECTOR_CHAIN.UTL_TO_EMBEDDINGS` chainable utility functions to generate one or more vector embeddings from textual documents and images.

- UTL_TO_GENERATE_TEXT
  Use the `DBMS_VECTOR_CHAIN.UTL_TO_GENERATE_TEXT` chainable utility function to generate a text response for a given prompt or an image, by accessing third-party text generation models.
- UTL_TO_SUMMARY
  Use the `DBMS_VECTOR_CHAIN.UTL_TO_SUMMARY` chainable utility function to generate a summary for textual documents.
- UTL_TO_TEXT
  Use the `DBMS_VECTOR_CHAIN.UTL_TO_TEXT` chainable utility function to convert an input document (for example, PDF, DOC, JSON, XML, or HTML) to plain text.
- Supported Languages and Data File Locations
  These are the supported languages for which language data files are distributed by default in the specified directories.

CREATE_CREDENTIAL

Use the `DBMS_VECTOR_CHAIN.CREATE_CREDENTIAL` credential helper procedure to create a credential name for storing user authentication details in Oracle Database.

Purpose

To securely manage authentication credentials in the database. You require these credentials to enable access during REST API calls to your chosen third-party service provider, such as Cohere, Google AI, Hugging Face, Oracle Cloud Infrastructure (OCI) Generative AI, OpenAI, or Vertex AI.

A credential name holds authentication parameters, such as user name, password, access token, private key, or fingerprint.

Note that if you are using Oracle Database as the service provider, then you do not need to create a credential.

> **WARNING:**
>
> Certain features of the database may allow you to access services offered separately by third-parties, for example, through the use of JSON specifications that facilitate your access to REST APIs.
>
> Your use of these features is solely at your own risk, and you are solely responsible for complying with any terms and conditions related to use of any such third-party services. Notwithstanding any other terms and conditions related to the third-party services, your use of such database features constitutes your acceptance of that risk and express exclusion of Oracle's responsibility or liability for any damages resulting from such access.

Syntax

```
DBMS_VECTOR_CHAIN.CREATE_CREDENTIAL (
    CREDENTIAL_NAME     IN VARCHAR2,
    PARAMS              IN JSON DEFAULT NULL
);
```

CREDENTIAL_NAME

Specify a name of the credential that you want to create for holding authentication parameters.

PARAMS

Specify authentication parameters in JSON format, based on your chosen service provider.

Generative AI requires the following authentication parameters:

```
{
"user_ocid"       : "<user ocid>",
"tenancy_ocid"    : "<tenancy ocid>",
"compartment_ocid": "<compartment ocid>",
"private_key"     : "<private key>",
"fingerprint"     : "<fingerprint>"
}
```

Cohere, Google AI, Hugging Face, OpenAI, and Vertex AI require the following authentication parameter:

```
{ "access_token": "<access token>" }
```

Table 12-18 Parameter Details

Parameter	Description
`user_ocid`	Oracle Cloud Identifier (OCID) of the user, as listed on the User Details page in the OCI console.
`tenancy_ocid`	OCID of your tenancy, as listed on the Tenancy Details page in the OCI console.
`compartment_ocid`	OCID of your compartment, as listed on the Compartments information page in the OCI console.

`private_key`	OCI private key.
	Note: The generated private key may appear as:
	`-----BEGIN RSA PRIVATE KEY-----`
	`<private key string>`
	`-----END RSA PRIVATE KEY-----`
	You pass the `<private key string>` value (excluding the `BEGIN` and `END` lines), either as a single line or as multiple lines.

`fingerprint`	Fingerprint of the OCI profile key, as listed on the User Details page under API Keys in the OCI console.
`access_token`	Access token obtained from your third-party service provider.

Required Privilege

You need the `CREATE CREDENTIAL` privilege to call this API.

Examples

- For Generative AI:

```
declare
  jo json_object_t;
begin
  jo := json_object_t();

jo.put('user_ocid','ocid1.user.oc1..aabbalbbaa1112233aabbaabb1111222aa1111b
b');

jo.put('tenancy_ocid','ocid1.tenancy.oc1..aaaaalbbbb1112233aaaabbaa1111222a
aa111a');

jo.put('compartment_ocid','ocid1.compartment.oc1..ababalabab1112233abababab
1111222aba11ab');
  jo.put('private_key','AAAaaaBBB11112222333...AAA111AAABBB222aaa1a/+');
  jo.put('fingerprint','01:1a:a1:aa:12:a1:12:1a:ab:12:01:ab:a1:12:ab:1a');
  dbms_vector_chain.create_credential(
    credential_name   => 'OCI_CRED',
    params            => json(jo.to_string));
end;
/
```

- For Cohere:

```
declare
  jo json_object_t;
begin
  jo := json_object_t();
  jo.put('access_token', 'A1Aa0abA1AB1a1Abc123ab1A123ab123AbcA12a');
  dbms_vector_chain.create_credential(
    credential_name   => 'COHERE_CRED',
    params            => json(jo.to_string));
end;
/
```

End-to-end examples:

To run end-to-end example scenarios using this procedure, see Use LLM-Powered APIs to Generate Summary and Text.

CREATE_LANG_DATA

Use the `DBMS_VECTOR_CHAIN.CREATE_LANG_DATA` chunker helper procedure to load your own language data file into the database.

Purpose

To create custom language data for your chosen language (specified using the `language` chunking parameter).

A language data file contains language-specific abbreviation tokens. You can supply this data to the chunker to help in accurately determining sentence boundaries of chunks, by using

knowledge of the input language's end-of-sentence (EOS) punctuations, abbreviations, and contextual rules.

Usage Notes

- All supported languages are distributed with the default language-specific abbreviation dictionaries. You can create a language data based on the abbreviation tokens loaded in the `schema.table.column`, using a user-specified language data name (`PREFERENCE_NAME`).
- After loading your language data, you can use language-specific chunking by specifying the `language` chunking parameter with `VECTOR_CHUNKS` or `UTL_TO_CHUNKS`.
- You can query these data dictionary views to access existing language data:
  - `ALL_VECTOR_LANG` displays all available languages data.
  - `USER_VECTOR_LANG` displays languages data from the schema of the current user.
  - `ALL_VECTOR_ABBREV_TOKENS` displays abbreviation tokens from all available language data.
  - `USER_VECTOR_ABBREV_TOKENS` displays abbreviation tokens from the language data owned by the current user.

Syntax

```
DBMS_VECTOR_CHAIN.CREATE_LANG_DATA (
    PARAMS       IN JSON default NULL
);
```

PARAMS

Specify the input parameters in JSON format:

```
{
    table_name,
    column_name,
    language,
    preference_name
}
```

Table 12-19 Parameter Details

Parameter	Description	Required	Default Value
`table_name`	Name of the table (along with the optional table owner) in which you want to load the language data	Yes	No value
`column_name`	Column name in the language data table in which you want to load the language data	Yes	No value
`language`	Any supported language name, as listed in Supported Languages and Data File Locations	Yes	No value
`preference_name`	User-specified preference name for this language data	Yes	No value

Example

```
declare
    params CLOB := '{"table_name"      : "eos_data_1",
                     "column_name"     : "token",
                     "language"        : "indonesian",
                     "preference_name" : "my_lang_1"}';
begin
    DBMS_VECTOR_CHAIN.CREATE_LANG_DATA(
        JSON (params));
end;
/
```

End-to-end example:

To run an end-to-end example scenario using this procedure, see Create and Use Custom Language Data.

Related Topics

- VECTOR_CHUNKS
- UTL_TO_CHUNKS
- Text Processing Views

CREATE_PREFERENCE

Use the `DBMS_VECTOR_CHAIN.CREATE_PREFERENCE` helper procedure to create a vectorizer preference, to be used when creating or updating hybrid vector indexes.

Purpose

To create a vectorizer preference.

This allows you to customize vector search parameters of a hybrid vector indexing pipeline. The goal of a vectorizer preference is to provide you with a straightforward way to configure how to chunk or embed your documents, without requiring a deep understanding of various chunking or embedding strategies.

Usage Notes

A **vectorizer** preference is a JSON object that collectively holds user-specified values related to the following chunking, embedding, or vector index creation parameters:

- Chunking (`UTL_TO_CHUNKS` and `VECTOR_CHUNKS`)
- Embedding (`UTL_TO_EMBEDDING`, `UTL_TO_EMBEDDINGS`, and `VECTOR_EMBEDDING`)
- Vector index creation (`distance`, `accuracy`, and `vector_idxtype`)

All vector index preferences follow the same JSON syntax as defined for their corresponding `DBMS_VECTOR` and `DBMS_VECTOR_CHAIN` APIs.

After creating a vectorizer preference, you can use the `VECTORIZER` parameter to pass this preference name in the `paramstring` of the `PARAMETERS` clause for `CREATE_HYBRID_VECTOR_INDEX` and `ALTER_INDEX` SQL statements.

Creating a preference is optional. If you do not specify any optional preference, then the index is created with system defaults.

Syntax

```
DBMS_VECTOR_CHAIN.CREATE_PREFERENCE (
    PREF_NAME     IN VARCHAR2,
    PREF_TYPE     IN VARCHAR2,
    PARAMS        IN JSON default NULL
);
```

PREF_NAME

Specify the name of the vectorizer preference to create.

PREF_TYPE

Type of preference. The only supported preference type is:

```
DBMS_VECTOR_CHAIN.VECTORIZER
```

PARAMS

Specify vector search-specific parameters in JSON format:

- Embedding Parameter
- Chunking Parameters
- Vector Index Parameters
- Paths Parameter

Embedding Parameter:

```
{ "model" : <embedding_model_for_vector_generation> }
```

For example:

```
{ "model" : MY_INDB_MODEL }
```

`model` specifies the name under which your ONNX embedding model is stored in the database.

If you do not have an in-database embedding model in ONNX format, then perform the steps listed in *Oracle Database AI Vector Search User's Guide*.

Chunking Parameters:

```
{
    "by"           :  mode,
    "max"          :  max,
    "overlap"      :  overlap,
    "split"        :  split_condition,
    "vocabulary"   :  vocabulary_name,
    "language"     :  nls_language,
    "normalize"    :  normalize_mode,
    "extended"     :  boolean
}
```

For example:

```
JSON(
 '{ "by"           :    "vocabulary",
    "max"          :    "100",
    "overlap"      :    "0",
    "split"        :    "none",
    "vocabulary"   :    "myvocab",
    "language"     :    "american",
    "normalize"    :    "all"
  }')
```

If you specify `split` as `custom` and `normalize` as `options`, then you must additionally specify the `custom_list` and `norm_options` parameters, respectively:

```
JSON(
 '{ "by"           :    "vocabulary",
    "max"          :    "100",
    "overlap"      :    "0",
    "split"        :    "custom",
    "custom_list"  :    [ "<p>" , "<s>" ],
    "vocabulary"   :    "myvocab",
    "language"     :    "american",
    "normalize"    :    "options",
    "norm_options" :    [ "whitespace" ]
  }')
```

The following table describes all the chunking parameters:

Parameter	Description and Acceptable Values
`by`	Specify a mode for splitting your data, that is, to split by counting the number of characters, words, or vocabulary tokens. **Valid values**: • `characters` (or `chars`): Splits by counting the number of characters. • `words`: Splits by counting the number of words. Words are defined as sequences of alphabetic characters, sequences of digits, individual punctuation marks, or symbols. For segmented languages without white space word boundaries (such as Chinese, Japanese, or Thai), each native character is considered a word (that is, unigram). • `vocabulary`: Splits by counting the number of vocabulary tokens. Vocabulary tokens are words or word pieces, recognized by the vocabulary of the tokenizer that your embedding model uses. You can load your vocabulary file using the chunker helper API `DBMS_VECTOR_CHAIN.CREATE_VOCABULARY`. **Note**: For accurate results, ensure that the chosen model matches the vocabulary file used for chunking. If you are not using a vocabulary file, then ensure that the input length is defined within the token limits of your model. **Default value**: `words`

Parameter	Description and Acceptable Values
`max`	Specify a limit on the maximum size of each chunk. This setting splits the input text at a fixed point where the maximum limit occurs in the larger text. The units of `max` correspond to the `by` mode, that is, to split data when it reaches the maximum size limit of a certain number of characters, words, numbers, punctuation marks, or vocabulary tokens. **Valid values**: • `by characters`: `50` to `4000` characters • `by words`: `10` to `1000` words • `by vocabulary`: `10` to `1000` tokens **Default value**: `100`

Parameter	Description and Acceptable Values

split [by]	Specify where to split the input text when it reaches the maximum size limit. This helps to keep related data together by defining appropriate boundaries for chunks.
	Valid values:
	• none:
	Splits at the max limit of characters, words, or vocabulary tokens.
	• newline, blankline, and space:
	These are single-split character conditions that split at the last split character before the max value.
	Use newline to split at the end of a line of text. Use blankline to split at the end of a blank line (sequence of characters, such as two newlines). Use space to split at the end of a blank space.
	• recursively:
	This is a multiple-split character condition that breaks the input text using an ordered list of characters (or sequences).
	recursively is predefined as BLANKLINE, newline, space, none in this order:
	1. If the input text is more than the max value, then split by the first split character.
	2. If that fails, then split by the second split character.
	3. And so on.
	4. If no split characters exist, then split by max wherever it appears in the text.
	• sentence:
	This is an end-of-sentence split condition that breaks the input text at a sentence boundary.
	This condition automatically determines sentence boundaries by using knowledge of the input language's sentence punctuation and contextual rules. This language-specific condition relies mostly on end-of-sentence (EOS) punctuations and common abbreviations.
	Contextual rules are based on word information, so this condition is only valid when splitting the text by words or vocabulary (not by characters).
	Note: This condition obeys the by word and max settings, and thus may not determine accurate sentence boundaries in some cases. For example, when a sentence is larger than the max value, it splits the sentence at max. Similarly, it includes multiple sentences in the text only when they fit within the max limit.
	• custom:
	Splits based on a custom split characters list. You can provide custom sequences up to a limit of 16 split character strings, with a maximum length of 10 each.
	Specify an array of valid text literals using the custom_list parameter.
	`{ "split" : "custom", "custom_list" : [ "split_chars1", ... ] }`
	For example:
	`{ "split" : "custom", "custom_list" : [ "<p>" , "<s>" ] }`
	Note: You can omit sequences only for tab (\t), newline (\n), and linefeed (\r).
	Default value: recursively

Parameter	Description and Acceptable Values
`overlap`	Specify the amount (as a positive integer literal or zero) of the preceding text that the chunk should contain, if any. This helps in logically splitting up related text (such as a sentence) by including some amount of the preceding chunk text. The amount of overlap depends on how the maximum size of the chunk is measured (in characters, words, or vocabulary tokens). The overlap begins at the specified `split` condition (for example, at `newline`). **Valid value**: `5%` to `20%` of `max` **Default value**: `0`

`language`	Specify the language of your input data.
	This clause is important, especially when your text contains certain characters (for example, punctuations or abbreviations) that may be interpreted differently in another language.
	Valid values:
	• NLS-supported language name or its abbreviation, as listed in *Oracle Database Globalization Support Guide*.
	• Custom language name or its abbreviation, as listed in Supported Languages and Data File Locations. You use the `DBMS_VECTOR_CHAIN.CREATE_LANG_DATA` chunker helper API to load language-specific data (abbreviation tokens) into the database, for your specified language.
	Note: You must use escape characters with any language abbreviation that is also a SQL reserved word (for example, language abbreviations such as `IN`, `AS`, `OR`, `IS`).
	For example:
	`SELECT dbms_vector_chain.utl_to_chunks('this is an example',`
	`JSON('{ "language" : "\"in\"" }'))`
	`from dual;`
	`SELECT dbms_vector_chain.utl_to_chunks('this is an example',`
	`JSON_OBJECT('language' value '"in"' RETURNING JSON))`
	`from dual;`
	Default value: `NLS_LANGUAGE` from session

Parameter	Description and Acceptable Values

normalize	Automatically pre-processes or post-processes issues (such as multiple consecutive spaces and smart quotes) that may arise when documents are converted into text. Oracle recommends you to use a normalization mode to extract high-quality chunks.
	Valid values:
	• none:
	Applies no normalization.
	• all:
	Normalizes common multi-byte (unicode) punctuation to standard single-byte.
	• options:
	Specify an array of normalization options using the norm_options parameter.
	`{`
	`"normalize" : "options",`
	`"norm_options" : [ "normalize_option1", ... ]`
	`}`
	– punctuation:
	Converts quotes, dashes, and other punctuation characters supported in the character set of the text to their common ASCII form. For example:
	* U+2013 (En Dash) maps to U+002D (Hyphen-Minus)
	* U+2018 (Left Single Quotation Mark) maps to U+0027 (Apostrophe)
	* U+2019 (Right Single Quotation Mark) maps to U+0027 (Apostrophe)
	* U+201B (Single High-Reversed-9 Quotation Mark) maps to U+0027 (Apostrophe)
	– whitespace:
	Minimizes whitespace by eliminating unnecessary characters.
	For example, retain blank lines, but remove any extra newlines and interspersed spaces or tabs: `" \n \n " => "\n\n"`
	– widechar:
	Normalizes wide, multi-byte digits and (a-z) letters to single-byte.
	These are multi-byte equivalents for 0-9 and a-z A-Z, which can show up in Chinese, Japanese, or Korean text.
	For example:
	`{`
	`"normalize" : "options",`
	`"norm_options" : [ "whitespace" ]`
	`}`
	Default value: none

extended	Increases the output limit of a VARCHAR2 string to 32767 bytes, without requiring you to set the max_string_size parameter to extended. **Default value**: 4000 or 32767 (when max_string_size=extended)

Vector Index Parameters:

```
{
  "distance"        :  <vector_distance>,
  "accuracy"        :  <vector_accuracy>,
```

```
  "vector_idxtype"   :  <vector_idxtype>
}
```

For example:

```
{
  "distance"         :  COSINE,
  "accuracy"         :  95,
  "vector_idxtype"   :  HNSW
}
```

Parameter	Description
distance	Distance metric or mathematical function used to compute the distance between vectors: • COSINE • MANHATTAN • DOT • EUCLIDEAN • L2_SQUARED • EUCLIDEAN_SQUARED **Note**: Currently, the HAMMING and JACCARD vector distance metrics are not supported with hybrid vector indexes. For detailed information on each of these metrics, see Vector Distance Functions and Operators. **Default value**: COSINE

Parameter	Description
`accuracy`	Target accuracy at which the approximate search should be performed when running an approximate search query using vector indexes. As explained in Understand Approximate Similarity Search Using Vector Indexes, you can specify non-default target accuracy values either by specifying a percentage value or by specifying internal parameters values, depending on the index type you are using. • For a Hierarchical Navigable Small World (HNSW) approximate search: In the case of an HNSW approximate search, you can specify a target accuracy percentage value to influence the number of candidates considered to probe the search. This is automatically calculated by the algorithm. A value of 100 will tend to impose a similar result as an exact search, although the system may still use the index and will not perform an exact search. The optimizer may choose to still use an index as it may be faster to do so given the predicates in the query. Instead of specifying a target accuracy percentage value, you can specify the `EFSEARCH` parameter to impose a certain maximum number of candidates to be considered while probing the index. The higher that number, the higher the accuracy. For detailed information, see Understand Hierarchical Navigable Small World Indexes. • For an Inverted File Flat (IVF) approximate search: In the case of an IVF approximate search, you can specify a target accuracy percentage value to influence the number of partitions used to probe the search. This is automatically calculated by the algorithm. A value of 100 will tend to impose an exact search, although the system may still use the index and will not perform an exact search. The optimizer may choose to still use an index as it may be faster to do so given the predicates in the query. Instead of specifying a target accuracy percentage value, you can specify the `NEIGHBOR PARTITION PROBES` parameter to impose a certain maximum number of partitions to be probed by the search. The higher that number, the higher the accuracy. For detailed information, see Understand Inverted File Flat Vector Indexes. Valid range for both HNSW and IVF vector indexes is: `> 0 and <= 100` **Default value**: None
`vector_idxtype`	Type of vector index to create: • `HNSW` for an HNSW vector index • `IVF` for an IVF vector index For detailed information on each of these index types, see Manage the Different Categories of Vector Indexes. **Default value**: `IVF`

Paths Parameter:

This field allows specification of an array of path objects. There can be many path objects and each path object must specify a `type` and a `path_list`.

> **Note:**
>
> If the user does not specify the `paths` field, the whole document would be considered.

```
"paths":[
        {"type"      : "<path_type>",
         "path_list" : ["<path_list_array>"]
        }
        ]
```

Let us consider a sample JSON document:

```
 {
      "person":
       {
         "bio": "James is a data scientist who specializes in natural
language .. ",
         "profile":
          {
              "text" : "James is a data scientist with expertise in Python
programming...",
              "embedding" :
[1.60541728E-001,5.76677322E-002,4.0473938E-003,1.2037459E-001,-5.98970801E-00
4, ..]
          },
         "avatar": "https://example.com/images/James.jpg"
       },
      "product":
      {
        "description": "A social media analytics tool.", "It helps brands
track...",
        "image": "https://example.com/images/data_tool.jpg",
        "embedding" :
[1.60541728E-001,5.76677322E-002,4.0473938E-003,1.2037459E-001,-5.98970801E-00
4, ..]
      }
 }
```

And a `path_list` corresponding to the above JSON is provided here:

```
"paths": [
         {"type"       : "VECTOR",
          "path_list" : ["$.person.profile.embedding", "$.product.embedding"]
         },
         {"type"       : "STRING",
          "path_list" : ["$.person.bio", "$.product.description"]
         }
         ]
```

The following table describes the details of `paths` parameter:

Parameter	Accepted Values
type	The possible values for this field are: • STRING - for fields that need to be converted to vectors. • VECTOR - for fields that are already vectors.
path_list	Accepts an array of paths with at least one path in valid JSON format - ($.a.b.c.d). **Note:** For the VECTOR option, Oracle currently accepts one vector array per path.

Example

```
begin
  DBMS_VECTOR_CHAIN.CREATE_PREFERENCE(
    'my_vec_spec',
     DBMS_VECTOR_CHAIN.VECTORIZER,
     json('{ "vector_idxtype" :  "hnsw",
             "model"          :  "my_doc_model",
             "by"             :  "words",
             "max"            :  100,
             "overlap"        :  10,
             "split"          :  "recursively,
             "language"       :  "english",
             "paths":          : [
                                  {
                                   "type"      : "VECTOR",
                                   "path_list" : ["$.person.profile.embedding"]
                                   }
                                  ]
              }'));
end;
/

CREATE HYBRID VECTOR INDEX my_hybrid_idx on
    doc_table(text_column)
    parameters('VECTORIZER my_vec_spec');
```

Related Topics

- CREATE HYBRID VECTOR INDEX

CREATE_VOCABULARY

Use the DBMS_VECTOR_CHAIN.CREATE_VOCABULARY chunker helper procedure to load your own token vocabulary file into the database.

Purpose

To create custom token vocabulary that is recognized by the tokenizer used by your vector embedding model.

A vocabulary contains a set of tokens (words and word pieces) that are collected during a model's statistical training process. You can supply this data to the chunker to help in accurately selecting the text size that approximates the maximum input limit imposed by the tokenizer of your embedding model.

Usage Notes

- Usually, the supported vocabulary files (containing recognized tokens) are included as part of a model's distribution. Oracle recommends to use the vocabulary files associated with your model.

  If a vocabulary file is not available, then you may download one of the following files depending on the tokenizer type:

  - **WordPiece**:

    Vocabulary file (`vocab.txt`) for the "bert-base-uncased" (English) or "bert-base-multilingual-cased" model

  - **Byte-Pair Encoding (BPE)**:

    Vocabulary file (`vocab.json`) for the "GPT2" model

    Use the following python script to extract the file:

    ```
    import json
    import sys

    with open(sys.argv[1], encoding="utf-8") as f:
      d = json.load(f)
      for term in d:
        print(term)
    ```

  - **SentencePiece**:

    Vocabulary file (`tokenizer.json`) for the "xlm-roberta-base" model

    Use the following python script to extract the file:

    ```
    import json
    import sys

    with open(sys.argv[1], encoding="utf-8") as f:
      d = json.load(f)
      for entry in d["model"]["vocab"]:
        print(entry[0])
    ```

  Ensure to save your vocabulary files in `UTF-8` encoding.

- You can create a vocabulary based on the tokens loaded in the `schema.table.column`, using a user-specified vocabulary name (`vocabulary_name`).

  After loading your vocabulary data, you can use the `by vocabulary` chunking mode (with `VECTOR_CHUNKS` or `UTL_TO_CHUNKS`) to split input data by counting the number of tokens.

- You can query these data dictionary views to access existing vocabulary data:

  - `ALL_VECTOR_VOCAB` displays all available vocabularies.
  - `USER_VECTOR_VOCAB` displays vocabularies from the schema of the current user.
  - `ALL_VECTOR_VOCAB_TOKENS` displays a list of tokens from all available vocabularies.

- `USER_VECTOR_VOCAB_TOKENS` displays a list of tokens from the vocabularies owned by the current user.

Syntax

```
DBMS_VECTOR_CHAIN.CREATE_VOCABULARY(
    PARAMS      IN JSON default NULL
);
```

PARAMS

Specify the input parameters in JSON format:

```
{
    table_name,
    column_name,
    vocabulary_name,
    format,
    cased
}
```

Table 12-20 Parameter Details

Parameter	Description	Required	Default Value
`table_name`	Name of the table (along with the optional table owner) in which you want to load the vocabulary file	Yes	No value
`column_name`	Column name in the vocabulary table in which you want to load the vocabulary file	Yes	No value
`vocabulary_name`	User-specified name of the vocabulary, along with the optional owner name (if other than the current owner)	Yes	No value
`format`	• `xlm` for SentencePiece tokenization • `bert` for WordPiece tokenization • `gpt2` for BPE tokenization	Yes	No value
`cased`	Character-casing of the vocabulary, that is, vocabulary to be treated as cased or uncased	No	`false`

Example

```
DECLARE
  params clob := '{"table_name"       : "doc_vocabtab",
                   "column_name"      : "token",
                   "vocabulary_name"  : "doc_vocab",
                   "format"           : "bert",
                   "cased"            : false}';

BEGIN
  dbms_vector_chain.create_vocabulary(json(params));
END;
/
```

End-to-end example:

To run an end-to-end example scenario using this procedure, see Create and Use Custom Vocabulary.

Related Topics

- VECTOR_CHUNKS
- UTL_TO_CHUNKS
- Text Processing Views

DROP_CREDENTIAL

Use the `DBMS_VECTOR_CHAIN.DROP_CREDENTIAL` credential helper procedure to drop an existing credential name from the data dictionary.

Syntax

```
DBMS_VECTOR_CHAIN.DROP_CREDENTIAL (
    CREDENTIAL_NAME      IN VARCHAR2
);
```

CREDENTIAL_NAME

Specify the credential name that you want to drop.

Examples

- For Generative AI:

  ```
  exec dbms_vector_chain.drop_credential('OCI_CRED');
  ```

- For Cohere:

  ```
  exec dbms_vector_chain.drop_credential('COHERE_CRED');
  ```

DROP_LANG_DATA

Use the `DBMS_VECTOR_CHAIN.DROP_LANG_DATA` chunker helper procedure to remove abbreviation data from the data dictionary.

Syntax

```
DBMS_VECTOR_CHAIN.DROP_LANG_DATA(
    PREF_NAME     IN VARCHAR2
);
```

LANG

Specify the name of the language data that you want to drop for a given language.

Example

```
DBMS_VECTOR_CHAIN.DROP_LANG_DATA('indonesian');
```

DROP_PREFERENCE

Use the `DBMS_VECTOR_CHAIN.DROP_PREFERENCE` preference helper procedure to remove an existing Vectorizer preference.

Syntax

```
DBMS_VECTOR_CHAIN.DROP_PREFERENCE (PREF_NAME);
```

PREF_NAME

Name of the Vectorizer preference to drop.

Example

```
DBMS_VECTOR_CHAIN.DROP_PREFERENCE ('scott_vectorizer');
```

DROP_VOCABULARY

Use the `DBMS_VECTOR_CHAIN.DROP_VOCABULARY` chunker helper procedure to remove vocabulary data from the data dictionary.

Syntax

```
DBMS_VECTOR_CHAIN.DROP_VOCABULARY(
    VOCABULARY_NAME    IN VARCHAR2
);
```

VOCAB_NAME

Specify the name of the vocabulary that you want to drop, in the form:

vocabulary_name

or

owner.vocabulary_name

Example

```
DBMS_VECTOR_CHAIN.DROP_VOCABULARY('MY_VOCAB_1');
```

RERANK

Use the `DBMS_VECTOR_CHAIN.RERANK` function to reassess and reorder an initial set of results to retrieve more relevant search output.

Purpose

To improve the relevance and quality of search results in both similarity search and Retrieval Augmented Generation (RAG) scenarios.

Reranking improves the quality of information ingested into an LLM by ensuring that the most relevant documents or chunks are prioritized. This helps to reduce hallucinations and improves the accuracy of generated outputs.

For this operation, Oracle AI Vector Search supports reranking models provided by Cohere and Vertex AI.

> **WARNING:**
>
> Certain features of the database may allow you to access services offered separately by third-parties, for example, through the use of JSON specifications that facilitate your access to REST APIs.
>
> Your use of these features is solely at your own risk, and you are solely responsible for complying with any terms and conditions related to use of any such third-party services. Notwithstanding any other terms and conditions related to the third-party services, your use of such database features constitutes your acceptance of that risk and express exclusion of Oracle's responsibility or liability for any damages resulting from such access.

Syntax

```
DBMS_VECTOR_CHAIN.RERANK(
                QUERY      IN CLOB,
                DOCUMENTS  IN JSON,
                PARAMS     IN JSON default NULL
) return JSON;
```

This function accepts the input containing a query as `CLOB` and a list of documents in `JSON` format. It then processes this information to generate a `JSON` object containing a reranked list of documents, sorted by score.

For example, a reranked output includes:

```
{
    "index"   : "1",
    "score"   : "0.99",
    "content" : "Jupiter boasts an impressive system of 95 known moons."
}
```

Where,

- `index` specifies the position of the document in the list of input text.

- `score` specifies the relevance score.
- `content` specifies the input text corresponding to the index.

QUERY

Specify the search query (typically from an initial search) as `CLOB`.

DOCUMENTS

Specify a JSON array of strings (list of potentially relevant documents to rerank) in the following format:

```
{
  "documents": [
  "string1",
  "string2",
    ...
  ]
}
```

PARAMS

Specify the following list of parameters in JSON format. All these parameters are mandatory.

```
{
  "provider"         : "<service provider>",
  "credential_name"  : "<credential name>",
  "url"              : "<REST endpoint URL for reranking>",
  "model"            : "<reranking model name>",
  ...
}
```

Table 12-21 RERANK Parameter Details

Parameter	Description
`provider`	Supported REST provider to access for reranking: • `cohere` • `vertexai`
`credential_name`	Name of the credential in the form: *`schema.credential_name`* A credential name holds authentication credentials to enable access to your provider for making REST API calls. You need to first set up your credential by calling the `DBMS_VECTOR_CHAIN.CREATE_CREDENTIAL` helper function to create and store a credential, and then refer to the credential name here. See CREATE_CREDENTIAL.
`url`	URL of the third-party provider endpoint for each REST call, as listed in Supported Third-Party Provider Operations and Endpoints.
`model`	Name of the reranking model in the form: *`schema.model_name`* If the model name is not schema-qualified, then the schema of the procedure invoker is used.

Additional REST provider parameters:

Optionally, specify additional provider-specific parameters for reranking.

> **Important:**
>
> - The following examples are for illustration purposes. For accurate and up-to-date information on additional parameters to use, refer to your third-party provider's documentation.
> - For a list of all supported REST endpoints, see Supported Third-Party Provider Operations and Endpoints.

Cohere example:

```
{
  "provider"        : "cohere",
  "credential_name" : "COHERE_CRED",
  "url"             : "https://api.cohere.example.com/rerank",
  "model"           : "rerank-english-v3.0",
  "return_documents": false,
  "top_n"           : 3
}
```

Vertex AI example:

```
{
  "provider"         : "vertexai",
  "credential_name"  : "VERTEXAI_CRED",
  "url"              : "https://googleapis.example.com/
default_ranking_config:rank",
  "model"            : "semantic-ranker-512@latest",
  "ignoreRecordDetailsInResponse" : true,
  "topN"             : 3
  }
```

Table 12-22 Additional REST Provider Parameter Details

Parameter	Description
`return_documents`	Whether to return search results with original documents or input text (`content`): • `false` (default, also recommended) to not return any input text (return only index and score) • `true` to return input text along with index and score **Note**: With Cohere as the provider, Oracle recommends that you keep this option disabled for better performance. You may choose to enable it for debugging purposes when you need to view the original text.

Table 12-22 (Cont.) Additional REST Provider Parameter Details

Parameter	Description
`ignoreRecordDetailsInResponse`	Whether to return search results with original record details or input text (`content`): • `false` (default) to return input text along with index and score • `true` (recommended) to not return any input text (return only index and score) **Note**: With Vertex AI as the provider, Oracle recommends that you keep this option enabled for better performance. You may choose to disable it for debugging purposes when you need to view the original text.
`top_n` or `topN`	The number of most relevant documents to return.

Examples

- Using Cohere:

```
declare
  params clob;
  reranked_output json;
begin
  params := '
{
  "provider": "cohere",
  "credential_name": "COHERE_CRED",
  "url": "https://api.cohere.com/v1/rerank",
  "model": "rerank-english-v3.0",
  "return_documents": true,
  "top_n": 3
}';

  reranked_output := dbms_vector_chain.rerank(:query,
json(:initial_retrieval_docs), json(params));
  dbms_output.put_line(json_serialize(reranked_output));
end;
/
```

- Using Vertex AI:

```
declare
  params clob;
  reranked_output json;
begin
  params := '
{
  "provider": "vertexai",
  "credential_name": "VERTEXAI_CRED",
  "url": "https://discoveryengine.googleapis.com/v1/projects/1085581009881/
locations/global/rankingConfigs/default_ranking_config:rank",
  "model": "semantic-ranker-512@latest",
  "ignoreRecordDetailsInResponse": false,
  "topN": 3
}';
```

```
  reranked_output := dbms_vector_chain.rerank(:query,
json(:initial_retrieval_docs), json(params));
  dbms_output.put_line(json_serialize(reranked_output));
end;
/
```

End-to-end example:

To run an end-to-end example scenario using this function, see Use Reranking for Better RAG Results.

UTL_TO_CHUNKS

Use the `DBMS_VECTOR_CHAIN.UTL_TO_CHUNKS` chainable utility function to split a large plain text document into smaller chunks of text.

Purpose

To perform a text-to-chunks transformation. This chainable utility function internally calls the `VECTOR_CHUNKS` SQL function for the operation.

To embed a large document, you may first need to split it into multiple appropriate-sized segments or chunks through a splitting process known as chunking (as explained in Understand the Stages of Data Transformations). A chunk can be words (to capture specific words or word pieces), sentences (to capture a specific meaning), or paragraphs (to capture broader themes). A single document may be split into multiple chunks, each transformed into a vector.

Syntax

```
DBMS_VECTOR_CHAIN.UTL_TO_CHUNKS (
    DATA         IN  CLOB | VARCHAR2,
    PARAMS       IN  JSON  default  NULL
) return VECTOR_ARRAY_T;
```

DATA

This function accepts the input data type as `CLOB` or `VARCHAR2`.

It returns an array of `CLOB`s, where each `CLOB` contains a chunk along with its metadata in JSON format, as follows:

```
{
    "chunk_id"     : NUMBER,
    "chunk_offset" : NUMBER,
    "chunk_length" : NUMBER,
    "chunk_data"   : "VARCHAR2(4000)"
}
```

For example:

```
{"chunk_id":1,"chunk_offset":1,"chunk_length":6,"chunk_data":"sample"}
```

Where,

- `chunk_id` specifies the chunk ID for each chunk.
- `chunk_offset` specifies the original position of each chunk in the source document, relative to the start of document which has a position of `1`.
- `chunk_length` specifies the character length of each chunk.
- `chunk_data` displays text pieces from each chunk.

PARAMS

Specify input parameters in JSON format.

```
{
    "by"           :     mode,
    "max"          :     max,
    "overlap"      :     overlap,
    "split"        :     split_condition,
    "custom_list"  :     [ split_chars1, ... ],
    "vocabulary"   :     vocabulary_name,
    "language"     :     nls_language,
    "normalize"    :     normalize_mode,
    "norm_options" :     [ normalize_option1, ... ],
    "extended"     :     boolean
}
```

For example:

```
JSON('
  { "by"           :    "vocabulary",
    "vocabulary"   :    "myvocab",
    "max"          :    "100",
    "overlap"      :    "0",
    "split"        :    "custom",
    "custom_list"  :    [ "<p>" , "<s>" ],
    "language"     :    "american",
    "normalize"    :    "options",
    "norm_options" :    [ "whitespace" ]
  }')
```

Here is a complete description of these parameters:

Parameter	Description and Acceptable Values
by	Specify a mode for splitting your data, that is, to split by counting the number of characters, words, or vocabulary tokens. **Valid values**: • characters (or chars): Splits by counting the number of characters. • words: Splits by counting the number of words. Words are defined as sequences of alphabetic characters, sequences of digits, individual punctuation marks, or symbols. For segmented languages without white space word boundaries (such as Chinese, Japanese, or Thai), each native character is considered a word (that is, unigram). • vocabulary: Splits by counting the number of vocabulary tokens. Vocabulary tokens are words or word pieces, recognized by the vocabulary of the tokenizer that your embedding model uses. You can load your vocabulary file using the chunker helper API DBMS_VECTOR_CHAIN.CREATE_VOCABULARY. **Note**: For accurate results, ensure that the chosen model matches the vocabulary file used for chunking. If you are not using a vocabulary file, then ensure that the input length is defined within the token limits of your model. **Default value**: words
max	Specify a limit on the maximum size of each chunk. This setting splits the input text at a fixed point where the maximum limit occurs in the larger text. The units of max correspond to the by mode, that is, to split data when it reaches the maximum size limit of a certain number of characters, words, numbers, punctuation marks, or vocabulary tokens. **Valid values**: • by characters: 50 to 4000 characters • by words: 10 to 1000 words • by vocabulary: 10 to 1000 tokens **Default value**: 100

Parameter	Description and Acceptable Values

split [by]	Specify where to split the input text when it reaches the maximum size limit. This helps to keep related data together by defining appropriate boundaries for chunks.
	Valid values:
	• none:
	Splits at the max limit of characters, words, or vocabulary tokens.
	• newline, blankline, and space:
	These are single-split character conditions that split at the last split character before the max value.
	Use newline to split at the end of a line of text. Use blankline to split at the end of a blank line (sequence of characters, such as two newlines). Use space to split at the end of a blank space.
	• recursively:
	This is a multiple-split character condition that breaks the input text using an ordered list of characters (or sequences).
	recursively is predefined as BLANKLINE, newline, space, none in this order:
	1. If the input text is more than the max value, then split by the first split character.
	2. If that fails, then split by the second split character.
	3. And so on.
	4. If no split characters exist, then split by max wherever it appears in the text.
	• sentence:
	This is an end-of-sentence split condition that breaks the input text at a sentence boundary.
	This condition automatically determines sentence boundaries by using knowledge of the input language's sentence punctuation and contextual rules. This language-specific condition relies mostly on end-of-sentence (EOS) punctuations and common abbreviations.
	Contextual rules are based on word information, so this condition is only valid when splitting the text by words or vocabulary (not by characters).
	Note: This condition obeys the by word and max settings, and thus may not determine accurate sentence boundaries in some cases. For example, when a sentence is larger than the max value, it splits the sentence at max. Similarly, it includes multiple sentences in the text only when they fit within the max limit.
	• custom:
	Splits based on a custom split characters list. You can provide custom sequences up to a limit of 16 split character strings, with a maximum length of 10 each.
	Specify an array of valid text literals using the custom_list parameter.
	`{ "split" : "custom", "custom_list" : [ "split_chars1", ... ] }`
	For example:
	`{ "split" : "custom", "custom_list" : [ "<p>" , "<s>" ] }`
	Note: You can omit sequences only for tab (\t), newline (\n), and linefeed (\r).
	Default value: recursively

Parameter	Description and Acceptable Values
`overlap`	Specify the amount (as a positive integer literal or zero) of the preceding text that the chunk should contain, if any. This helps in logically splitting up related text (such as a sentence) by including some amount of the preceding chunk text. The amount of overlap depends on how the maximum size of the chunk is measured (in characters, words, or vocabulary tokens). The overlap begins at the specified `split` condition (for example, at `newline`). **Valid value**: `5%` to `20%` of `max` **Default value**: `0`

`language`	Specify the language of your input data.
	This clause is important, especially when your text contains certain characters (for example, punctuations or abbreviations) that may be interpreted differently in another language.
	Valid values:
	• NLS-supported language name or its abbreviation, as listed in *Oracle Database Globalization Support Guide*.
	• Custom language name or its abbreviation, as listed in Supported Languages and Data File Locations. You use the `DBMS_VECTOR_CHAIN.CREATE_LANG_DATA` chunker helper API to load language-specific data (abbreviation tokens) into the database, for your specified language.
	Note: You must use escape characters with any language abbreviation that is also a SQL reserved word (for example, language abbreviations such as `IN`, `AS`, `OR`, `IS`).
	For example:
	`SELECT dbms_vector_chain.utl_to_chunks('this is an example',`
	`JSON('{ "language" : "\"in\"" }'))`
	`from dual;`
	`SELECT dbms_vector_chain.utl_to_chunks('this is an example',`
	`JSON_OBJECT('language' value '"in"' RETURNING JSON))`
	`from dual;`
	Default value: `NLS_LANGUAGE` from session

Parameter	Description and Acceptable Values

normalize	Automatically pre-processes or post-processes issues (such as multiple consecutive spaces and smart quotes) that may arise when documents are converted into text. Oracle recommends you to use a normalization mode to extract high-quality chunks.
	Valid values:
	• none:
	Applies no normalization.
	• all:
	Normalizes common multi-byte (unicode) punctuation to standard single-byte.
	• options:
	Specify an array of normalization options using the norm_options parameter.
	`{ "normalize" : "options", "norm_options" : [ "normalize_option1", ... ] }`
	– punctuation:
	Converts quotes, dashes, and other punctuation characters supported in the character set of the text to their common ASCII form. For example:
	* U+2013 (En Dash) maps to U+002D (Hyphen-Minus)
	* U+2018 (Left Single Quotation Mark) maps to U+0027 (Apostrophe)
	* U+2019 (Right Single Quotation Mark) maps to U+0027 (Apostrophe)
	* U+201B (Single High-Reversed-9 Quotation Mark) maps to U+0027 (Apostrophe)
	– whitespace:
	Minimizes whitespace by eliminating unnecessary characters.
	For example, retain blank lines, but remove any extra newlines and interspersed spaces or tabs: `" \n \n " => "\n\n"`
	– widechar:
	Normalizes wide, multi-byte digits and (a-z) letters to single-byte.
	These are multi-byte equivalents for 0-9 and a-z A-Z, which can show up in Chinese, Japanese, or Korean text.
	For example:
	`{ "normalize" : "options", "norm_options" : [ "whitespace" ] }`
	Default value: none

extended	Increases the output limit of a VARCHAR2 string to 32767 bytes, without requiring you to set the max_string_size parameter to extended. **Default value**: 4000 or 32767 (when max_string_size=extended)

Example

```
SELECT D.id doc,
    JSON_VALUE(C.column_value, '$.chunk_id' RETURNING NUMBER) AS id,
    JSON_VALUE(C.column_value, '$.chunk_offset' RETURNING NUMBER) AS pos,
    JSON_VALUE(C.column_value, '$.chunk_length' RETURNING NUMBER) AS siz,
    JSON_VALUE(C.column_value, '$.chunk_data') AS txt
```

```
FROM docs D,
   dbms_vector_chain.utl_to_chunks(D.text,
   JSON('{ "by"       : "words",
           "max"      : "100",
           "overlap"  : "0",
           "split"    : "recursively",
           "language" : "american",
           "normalize": "all" }')) C;
```

End-to-end examples:

To run end-to-end example scenarios using this function, see Perform Chunking With Embedding and Configure Chunking Parameters.

Related Topics

- VECTOR_CHUNKS

UTL_TO_EMBEDDING and UTL_TO_EMBEDDINGS

Use the `DBMS_VECTOR_CHAIN.UTL_TO_EMBEDDING` and `DBMS_VECTOR_CHAIN.UTL_TO_EMBEDDINGS` chainable utility functions to generate one or more vector embeddings from textual documents and images.

Purpose

To automatically generate one or more vector embeddings from textual documents and images.

- Text to Vector:

  You can perform a text-to-embedding transformation by accessing either Oracle Database or a third-party service provider:

  - Oracle Database as the service provider (default setting):

    This API calls an ONNX format embedding model that you load into the database.

  - Third-party embedding model:

    This API makes a REST API call to your chosen remote service provider (Cohere, Generative AI, Google AI, Hugging Face, OpenAI, or Vertex AI) or local service provider (Ollama).

- Image to Vector:

  You can also perform an image-to-embedding transformation. This API makes a REST call to your chosen image embedding model or multimodal embedding model by Vertex AI. Note that currently Vertex AI is the only supported service provider for this operation.

> **WARNING:**
>
> Certain features of the database may allow you to access services offered separately by third-parties, for example, through the use of JSON specifications that facilitate your access to REST APIs.
>
> Your use of these features is solely at your own risk, and you are solely responsible for complying with any terms and conditions related to use of any such third-party services. Notwithstanding any other terms and conditions related to the third-party services, your use of such database features constitutes your acceptance of that risk and express exclusion of Oracle's responsibility or liability for any damages resulting from such access.

Syntax

- Text to Vector:

```
DBMS_VECTOR_CHAIN.UTL_TO_EMBEDDING (
    DATA            IN CLOB,
    PARAMS          IN JSON default NULL
) return VECTOR;


DBMS_VECTOR_CHAIN.UTL_TO_EMBEDDINGS (
    DATA            IN VECTOR_ARRAY_T,
    PARAMS          IN JSON default NULL
) return VECTOR_ARRAY_T;
```

- Image to Vector:

```
DBMS_VECTOR_CHAIN.UTL_TO_EMBEDDING (
    DATA            IN BLOB,
    MODALITY        IN VARCHAR2,
    PARAMS          IN JSON default NULL
) return VECTOR;
```

DATA

- Text to Vector:

  `UTL_TO_EMBEDDING` accepts the input as `CLOB` containing textual data (text strings or small documents). It then converts the text to a single embedding (`VECTOR`).

  `UTL_TO_EMBEDDINGS` converts an array of chunks (`VECTOR_ARRAY_T`) to an array of embeddings (`VECTOR_ARRAY_T`).

  > **Note:**
  >
  > Although data is a `CLOB` or a `VECTOR_ARRAY_T` of `CLOB`, the maximum input is 4000 characters. If you have input that is greater, you can use `UTL_TO_CHUNKS` to split the data into smaller chunks before passing in.

- Image to Vector:

UTL_TO_EMBEDDING accepts the input as BLOB containing media data for media files such as images. It then converts the image input to a single embedding (VECTOR).

A generated embedding output includes:

```
{
    "embed_id"    : NUMBER,
    "embed_data"  : "VARCHAR2(4000)",
    "embed_vector": "CLOB"
}
```

Where,

- embed_id displays the ID number of each embedding.
- embed_data displays the input text that is transformed into embeddings.
- embed_vector displays the generated vector representations.

MODALITY

For BLOB inputs, specify the type of content to vectorize. The only supported value is image.

PARAMS

Specify input parameters in JSON format, depending on the service provider that you want to use.

If using Oracle Database as the provider:

```
{
  "provider" : "database",
  "model"    : "<in-database ONNX embedding model filename>"
}
```

Table 12-23 Database Provider Parameter Details

Parameter	Description
provider	Specify database (default setting) to use Oracle Database as the provider. With this setting, you must load an ONNX format embedding model into the database.
model	User-specified name under which the imported ONNX embedding model is stored in Oracle Database. If you do not have an embedding model in ONNX format, then perform the steps listed in Convert Pretrained Models to ONNX Format.

If using a third-party provider:

Set the following parameters along with additional embedding parameters specific to your provider:

- For UTL_TO_EMBEDDING:

```
{
  "provider"        : "<AI service provider>",
  "credential_name" : "<credential name>",
  "url"             : "<REST endpoint URL for embedding service>",
```

```
  "model"           : "<REST provider embedding model name>",
  "transfer_timeout": <maximum wait time for the request to complete>,
  "max_count": "<maximum calls to the AI service provider>",
  "<additional REST provider parameter>": "<REST provider parameter
value>"
}
```

- For `UTL_TO_EMBEDDINGS`:

```
{
  "provider"        : "<AI service provider>",
  "credential_name" : "<credential name>",
  "url"             : "<REST endpoint URL for embedding service>",
  "model"           : "<REST provider embedding model name>",
  "transfer_timeout": <maximum wait time for the request to complete>,
  "batch_size"      : "<number of vectors to request at a time>",
  "max_count": "<maximum calls to the AI service provider>",
  "<additional REST provider parameter>": "<REST provider parameter
value>"
}
```

Table 12-24 Third-Party Provider Parameter Details

Parameter	Description
`provider`	Third-party service provider that you want to access for this operation. A REST call is made to the specified provider to access its embedding model. For image input, specify `vertexai`. For text input, specify one of the following values: • `cohere` • `googleai` • `huggingface` • `ocigenai` • `openai` • `vertexai`
`credential_name`	Name of the credential in the form: *`schema.credential_name`* A credential name holds authentication credentials to enable access to your provider for making REST API calls. You need to first set up your credential by calling the `DBMS_VECTOR_CHAIN.CREATE_CREDENTIAL` helper function to create and store a credential, and then refer to the credential name here. See CREATE_CREDENTIAL.
`url`	URL of the third-party provider endpoint for each REST call, as listed in Supported Third-Party Provider Operations and Endpoints.

Table 12-24 (Cont.) Third-Party Provider Parameter Details

Parameter	Description
`model`	Name of the third-party embedding model in the form: *`schema.model_name`* If you do not specify a schema, then the schema of the procedure invoker is used. **Note**: • For Generative AI, all the supported third-party models are listed in Supported Third-Party Provider Operations and Endpoints. • For accurate results, ensure that the chosen text embedding model matches the vocabulary file used for chunking. If you are not using a vocabulary file, then ensure that the input length is defined within the token limits of your model. • To get image embeddings, you can use any image embedding model or multimodal embedding model supported by Vertex AI. Multimodal embedding is a technique that vectorizes data from different modalities such as text and images. When using a multimodal embedding model to generate embeddings, ensure that you use the same model to vectorize both types of content (text and images). By doing so, the resulting embeddings are compatible and situated in the same vector space, which allows for effective comparison between the two modalities during similarity searches.
`transfer_timeout`	Maximum time to wait for the request to complete. The default value is `60` seconds. You can increase this value for busy web servers.
`batch_size`	Maximum number of vectors to request at a time. For example, for a batch size of `50`, if `100` chunks are passed, then this API sends two requests with an array of `50` strings each. If `30` chunks are passed (which is lesser than the defined batch size), then the API sends those in a single request. For REST calls, it is more efficient to send a batch of inputs at a time rather than requesting a single input per call. Increasing the batch size can provide better performance, whereas reducing the batch size may reduce memory and data usage, especially if your provider has a rate limit. The default or maximum allowed value depends on the third-party provider settings.
`max_count`	Maximum number of times the API can be called for a given third-party provider. When set to an integer *n*, `max_count` stops the execution of the API for the given provider beyond *n* times. This prevents accidental calling of a third-party over some limit, for example to avoid surpassing the service amount that you have purchased.

Additional third-party provider parameters:

Optionally, specify additional provider-specific parameters.

Table 12-25 Additional REST Provider Parameter Details

Parameter	Description
`input_type`	Type of input to vectorize.

Let us look at some example configurations for all third-party providers:

Important:

- The following examples are for illustration purposes. For accurate and up-to-date information on the parameters to use, refer to your third-party provider's documentation.
- For a list of all supported REST endpoint URLs, see Supported Third-Party Provider Operations and Endpoints.
- The generated embedding results may be different between requests for the same input and configuration, depending on your embedding model or floating point precision. However, this does not affect your queries (and provides semantically correct results) because the vector distance will be similar.

Cohere example:

```
{
  "provider"       : "cohere",
  "credential_name": "COHERE_CRED",
  "url"            : "https://api.cohere.example.com/embed",
  "model"          : "embed-english-light-v2.0",
  "input_type"     : "search_query"
}
```

Generative AI example:

```
{
  "provider"       : "ocigenai",
  "credential_name": "OCI_CRED",
  "url"            : "https://generativeai.oci.example.com/embedText",
  "model"          : "cohere.embed-english-v3.0",
  "batch_size"     : 10
}
```

Google AI example:

```
{
  "provider"       : "googleai",
  "credential_name": "GOOGLEAI_CRED",
  "url"            : "https://googleapis.example.com/models/",
  "model"          : "embedding-001"
}
```

Hugging Face example:

```
{
  "provider"       : "huggingface",
  "credential_name": "HF_CRED",
  "url"            : "https://api.huggingface.example.com/",
  "model"          : "sentence-transformers/all-MiniLM-L6-v2"
}
```

Ollama example:

```
{
  "provider"       : "ollama",
  "host"           : "local",
  "url"            : "http://localhost:11434/api/embeddings",
  "model"          : "phi3:mini"
}
```

OpenAI example:

```
{
  "provider"       : "openai",
  "credential_name": "OPENAI_CRED",
  "url"            : "https://api.openai.example.com/embeddings",
  "model"          : "text-embedding-3-small"
}
```

Vertex AI example:

```
{
  "provider"       : "vertexai",
  "credential_name": "VERTEXAI_CRED",
  "url"            : "https://googleapis.example.com/models/",
  "model"          : "textembedding-gecko:predict"
}
```

Examples

You can use `UTL_TO_EMBEDDING` in a `SELECT` clause and `UTL_TO_EMBEDDINGS` in a `FROM` clause, as follows:

UTL_TO_EMBEDDING:

- **Text to vector using Generative AI**:

  The following examples use `UTL_TO_EMBEDDING` to generate an embedding with `Hello world` as the input.

  Here, the cohere.embed-english-v3.0 model is used by accessing Generative AI as the provider. You can replace the `model` value with any other supported model that you want to use with Generative AI, as listed in Supported Third-Party Provider Operations and Endpoints.

```
-- declare embedding parameters

var params clob;

begin
 :params := '
{
  "provider": "ocigenai",
  "credential_name": "OCI_CRED",
  "url": "https://inference.generativeai.us-chicago-1.oci.oraclecloud.com/
20231130/actions/embedText",
  "model": "cohere.embed-english-v3.0",
```

```
  "batch_size": 10
}';
end;
/

-- get text embedding: PL/SQL example

declare
  input clob;
  v vector;
begin
  input := 'Hello world';

  v := dbms_vector_chain.utl_to_embedding(input, json(params));
  dbms_output.put_line(vector_serialize(v));
exception
  when OTHERS THEN
    DBMS_OUTPUT.PUT_LINE (SQLERRM);
    DBMS_OUTPUT.PUT_LINE (SQLCODE);
end;
/

-- get text embedding: select example

select dbms_vector_chain.utl_to_embedding('Hello world', json(:params))
from dual;
```

- **Image to vector using Vertex AI**:

  The following examples use `UTL_TO_EMBEDDING` to generate an embedding by accessing the Vertex AI's multimodal embedding model.

  Here, the input is `parrots.jpg`, `VEC_DUMP` is a local directory that stores the `parrots.jpg` file, and the modality is specified as `image`.

```
-- declare embedding parameters

var params clob;

begin
  :params := '
{
  "provider": "vertexai",
  "credential_name": "VERTEXAI_CRED",
  "url": "https://LOCATION-aiplatform.googleapis.com/v1/projects/PROJECT/
locations/LOCATION/publishers/google/models/",
  "model": "multimodalembedding:predict"
}';
end;
/

-- get image embedding: PL/SQL example

declare
  v vector;
  output clob;
begin
```

```
  v := dbms_vector_chain.utl_to_embedding(
    to_blob(bfilename('VEC_DUMP', 'parrots.jpg')), 'image', json(:params));
  output := vector_serialize(v);
  dbms_output.put_line('vector data=' || dbms_lob.substr(output, 100) ||
'...');
end;
/

-- get image embedding: select example

select dbms_vector_chain.utl_to_embedding(
  to_blob(bfilename('VEC_DUMP', 'parrots.jpg')), 'image', json(:params));
```

- **Text to vector using in-database embedding model**:

  The following example uses `UTL_TO_EMBEDDING` to generate a vector embedding by calling an ONNX format embedding model (`doc_model`) loaded into Oracle Database.

  Here, the provider is `database`, and the input is `hello`.

```
var params clob;
exec :params := '{"provider":"database", "model":"doc_model"}';

select dbms_vector_chain.utl_to_embedding('hello', json(:params)) from
dual;
```

  For complete example, see Convert Text String to Embedding Within Oracle Database.

- **End-to-end examples**:

  To run various end-to-end example scenarios using `UTL_TO_EMBEDDING`, see Generate Embedding.

UTL_TO_EMBEDDINGS:

- **Text to vector using in-database embedding model**:

  The following example uses `UTL_TO_EMBEDDINGS` to generate an array of embeddings by calling an ONNX format embedding model (`doc_model`) loaded into Oracle Database.

  Here, the provider is `database`, and the input is a PDF document stored in the `documentation_tab` table. As you can see, you first use `UTL_TO_CHUNKS` to split the data into smaller chunks before passing in to `UTL_TO_EMBEDDINGS`.

```
CREATE TABLE doc_chunks as
(select dt.id doc_id, et.embed_id, et.embed_data,
to_vector(et.embed_vector) embed_vector
 from
   documentation_tab dt,
   dbms_vector_chain.utl_to_embeddings(

dbms_vector_chain.utl_to_chunks(dbms_vector_chain.utl_to_text(dt.data),
json('{"normalize":"all"}')),
       json('{"provider":"database", "model":"doc_model"}')) t,
   JSON_TABLE(t.column_value, '$[*]' COLUMNS (embed_id NUMBER PATH
'$.embed_id', embed_data VARCHAR2(4000) PATH '$.embed_data', embed_vector
CLOB PATH '$.embed_vector')) et
);
```

For complete example, see SQL Quick Start Using a Vector Embedding Model Uploaded into the Database.

- **End-to-end examples**:

  To run various end-to-end example scenarios using `UTL_TO_EMBEDDINGS`, see Perform Chunking With Embedding.

UTL_TO_GENERATE_TEXT

Use the `DBMS_VECTOR_CHAIN.UTL_TO_GENERATE_TEXT` chainable utility function to generate a text response for a given prompt or an image, by accessing third-party text generation models.

Purpose

To communicate with Large Language Models (LLMs) through natural language conversations. You can generate a textual answer, description, or summary for prompts and images, given as input to LLM-powered chat interfaces.

- Prompt to Text:

  A prompt can be an input text string, such as a question that you ask an LLM. For example, "`What is Oracle Text?`". A prompt can also be a command, such as "`Summarize the following ...`", "`Draft an email asking for ...`", or "`Rewrite the following ...`", and can include results from a search. The LLM responds with a textual answer or description based on the specified task in the prompt.

  For this operation, this API makes a REST call to your chosen remote third-party provider (Cohere, Generative AI, Google AI, Hugging Face, OpenAI, or Vertex AI) or local third-party provider (Ollama).

- Image to Text:

  You can also prompt with a media file, such as an image, to extract text from pictures or photos. You supply a text question as the prompt (such as "`What is this image about?`" or "`How many birds are there in this painting?`") along with the image. The LLM responds with a textual analysis or description of the contents of the image.

  For this operation, this API makes a REST call to your chosen remote third-party provider (Google AI, Hugging Face, OpenAI, or Vertex AI) or local third-party provider (Ollama).

> **WARNING:**
>
> Certain features of the database may allow you to access services offered separately by third-parties, for example, through the use of JSON specifications that facilitate your access to REST APIs.
>
> Your use of these features is solely at your own risk, and you are solely responsible for complying with any terms and conditions related to use of any such third-party services. Notwithstanding any other terms and conditions related to the third-party services, your use of such database features constitutes your acceptance of that risk and express exclusion of Oracle's responsibility or liability for any damages resulting from such access.

Syntax

This function accepts the input as `CLOB` containing text data (for textual prompts) or as `BLOB` containing media data (for media files such as images). It then processes this information to generate a new `CLOB` containing the generated text.

- Prompt to Text:

```
DBMS_VECTOR_CHAIN.UTL_TO_GENERATE_TEXT (
        DATA          IN CLOB,
        PARAMS        IN JSON default NULL
) return CLOB;
```

- Image to Text:

```
DBMS_VECTOR_CHAIN.UTL_TO_GENERATE_TEXT(
          TEXT_DATA      IN CLOB,
          MEDIA_DATA     IN BLOB,
          MEDIA_TYPE     IN VARCHAR2 default 'image/jpeg',
          PARAMS         IN JSON default NULL
) return CLOB;
```

DATA and TEXT_DATA

Specify the textual prompt as `CLOB` for the `DATA` or `TEXT_DATA` clause.

> **Note:**
>
> Hugging Face uses an image captioning model that does not require a prompt, when giving an image as input. If you input a prompt along with an image, then the prompt will be ignored.

MEDIA_DATA

Specify the `BLOB` file, such as an image or a visual PDF file.

MEDIA_TYPE

Specify the image format for the given image or visual PDF file (`BLOB` file) in one of the supported image data MIME types. For example:

- For PNG: `image/png`
- For JPEG: `image/jpeg`
- For PDF: `application/pdf`

> **Note:**
>
> For a complete list of the supported image formats, refer to your third-party provider's documentation.

PARAMS

Specify the following input parameters in JSON format, depending on the service provider that you want to access for text generation:

```
{
  "provider"        : "<AI service provider>",
  "credential_name" : "<credential name>",
  "url"             : "<REST endpoint URL for text generation service>",
  "model"           : "<text generation model name>",
  "transfer_timeout": <maximum wait time for the request to complete>,
  "max_count": "<maximum calls to the AI service provider>",
  "<additional REST provider parameter>": "<REST provider parameter value>"
}
```

Table 12-26 UTL_TO_GENERATE_TEXT Parameter Details

Parameter	Description
provider	Supported REST provider that you want to access to generate text. Specify one of the following values: For CLOB input: • cohere • googleai • huggingface • ocigenai • openai • vertexai For BLOB input: • googleai • huggingface • openai • vertexai
credential_name	Name of the credential in the form: *schema.credential_name* A credential name holds authentication credentials to enable access to your provider for making REST API calls. You need to first set up your credential by calling the DBMS_VECTOR_CHAIN.CREATE_CREDENTIAL helper function to create and store a credential, and then refer to the credential name here. See CREATE_CREDENTIAL.
url	URL of the third-party provider endpoint for each REST call, as listed in Supported Third-Party Provider Operations and Endpoints.
model	Name of the third-party text generation model in the form: *schema.model_name* If the model name is not schema-qualified, then the schema of the procedure invoker is used. **Note**: For Generative AI, all the supported third-party models are listed in Supported Third-Party Provider Operations and Endpoints.
transfer_timeout	Maximum time to wait for the request to complete. The default value is 60 seconds. You can increase this value for busy web servers.

Table 12-26 (Cont.) UTL_TO_GENERATE_TEXT Parameter Details

Parameter	Description
`max_count`	Maximum number of times the API can be called for a given third-party provider. When set to an integer *n*, `max_count` stops the execution of the API for the given provider beyond *n* times. This prevents accidental calling of a third-party over some limit, for example to avoid surpassing the service amount that you have purchased.

Additional third-party provider parameters:

Optionally, specify additional provider-specific parameters.

Table 12-27 Additional REST Provider Parameter Details

Parameter	Description
`max_tokens`	Maximum number of tokens in the output text.
`temperature`	Degree of randomness used when generating the output text, in the range of `0.0-5.0`. To generate the same output for a prompt, use `0`. To generate a random new text for that prompt, increase the temperature. **Note**: Start with the temperature set to `0`. If you do not require random results, a recommended temperature value is between `0` and `1`. A higher value is not recommended because a high temperature may produce creative text, which might also include hallucinations.
`topP`	Probability of tokens in the output, in the range of `0.0-1.0`. A lower value provides less random responses and a higher value provides more random responses.
`candidateCount`	Number of response variations to return, in the range of `1-4`.
`maxOutputTokens`	Maximum number of tokens to generate for each response.

Let us look at some example configurations for all third-party providers:

Important:

- The following examples are for illustration purposes. For accurate and up-to-date information on additional parameters to use, refer to your third-party provider's documentation.
- For a list of all supported REST endpoint URLs, see Supported Third-Party Provider Operations and Endpoints.

Cohere example:

```
{
  "provider"       : "cohere",
  "credential_name": "COHERE_CRED",
  "url"            : "https://api.cohere.example.com/chat",
```

```
  "model"           : "command"
}
```

Generative AI example:

> **Note:**
>
> For Generative AI, if you want to pass any additional REST provider-specific parameters, then you must enclose those in `chatRequest`.

```
{
  "provider"       : "ocigenai",
  "credential_name": "OCI_CRED",
  "url"            : "https://inference.generativeai.us-example.com/chat",
  "model"          : "cohere.command-r-16k",
  "chatRequest"    : {
                      "maxTokens"    : 256
                     }
}
```

Google AI example:

```
{
  "provider"        : "googleai",
  "credential_name" : "GOOGLEAI_CRED",
  "url"             : "https://googleapis.example.com/models/",
  "model"           : "gemini-pro:generateContent"
}
```

Hugging Face example:

```
{
  "provider"        : "huggingface",
  "credential_name" : "HF_CRED",
  "url"             : "https://api.huggingface.example.com/models/",
  "model"           : "gpt2"
}
```

Ollama example:

```
{
  "provider"       : "ollama",
  "host"           : "local",
  "url"            : "http://localhost:11434/api/generate",
  "model"          : "phi3:mini"
}
```

OpenAI example:

```
{
  "provider"        : "openai",
  "credential_name" : "OPENAI_CRED",
  "url"             : "https://api.openai.example.com",
  "model"           : "gpt-4o-mini",
  "max_tokens"      : 60,
  "temperature"     : 1.0
}
```

Vertex AI example:

```
{
  "provider"         : "vertexai",
  "credential_name"  : "VERTEXAI_CRED",
  "url"              : "https://googleapis.example.com/models/",
  "model"            : "gemini-1.0-pro:generateContent",
  "generation_config": {
                        "temperature"    : 0.9,
                        "topP"           : 1,
                        "candidateCount" : 1,
                        "maxOutputTokens": 256
                       }
}
```

Examples

- Prompt to Text:

  The following statements generate a text response by making a REST call to Generative AI. The prompt given here is "`What is Oracle Text?`".

  Here, the cohere.command-r-16k and meta.llama-3.1-70b-instruct models are used. You can replace the `model` value with any other supported model that you want to use with Generative AI, as listed in Supported Third-Party Provider Operations and Endpoints.

  Using the cohere.command-r-16k model:

  ```
  -- select example

  var params clob;
  exec :params := '
  {
    "provider"       : "ocigenai",
    "credential_name": "OCI_CRED",
    "url"            : "https://inference.generativeai.us-
  chicago-1.oci.oraclecloud.com/20231130/actions/chat",
    "model"          : "cohere.command-r-16k",
    "chatRequest"    : {
                         "maxTokens": 256
                       }
  }';

  select dbms_vector_chain.utl_to_generate_text(
   'What is Oracle Text?',
  ```

```
 json(:params)) from dual;

-- PL/SQL example

declare
  input clob;
  params clob;
  output clob;
begin
  input := 'What is Oracle Text?';

  params := '
{
  "provider"       : "ocigenai",
  "credential_name": "OCI_CRED",
  "url"            : "https://inference.generativeai.us-
chicago-1.oci.oraclecloud.com/20231130/actions/chat",
  "model"          : "cohere.command-r-16k",
  "chatRequest"    : {
                      "maxTokens": 256
                     }
}';

  output := dbms_vector_chain.utl_to_generate_text(input, json(params));
  dbms_output.put_line(output);
  if output is not null then
    dbms_lob.freetemporary(output);
  end if;
exception
  when OTHERS THEN
    DBMS_OUTPUT.PUT_LINE (SQLERRM);
    DBMS_OUTPUT.PUT_LINE (SQLCODE);
end;
/
```

Using the meta.llama-3.1-70b-instruct model:

```
-- select example

var params clob;
exec :params := '
{
   "provider"       : "ocigenai",
   "credential_name": "OCI_CRED",
   "url"            : "https://inference.generativeai.us-
chicago-1.oci.oraclecloud.com/20231130/actions/chat",
   "model"          : "meta.llama-3.1-70b-instruct",
   "chatRequest"    : {
                       "topK" : 1
                      }
}';

select dbms_vector_chain.utl_to_generate_text(
 'What is Oracle Text?',
 json(:params)) from dual;
```

```
-- PL/SQL example

declare
  input clob;
  params clob;
  output clob;
begin
  input := 'What is Oracle Text?';

  params := '
{
   "provider"       : "ocigenai",
   "credential_name": "OCI_CRED",
   "url"            : "https://inference.generativeai.us-
chicago-1.oci.oraclecloud.com/20231130/actions/chat",
   "model"          : "meta.llama-3.1-70b-instruct",
   "chatRequest"    : {
                       "topK" : 1
                      }
}';

  output := dbms_vector_chain.utl_to_generate_text(input, json(params));
  dbms_output.put_line(output);
  if output is not null then
    dbms_lob.freetemporary(output);
  end if;
exception
  when OTHERS THEN
    DBMS_OUTPUT.PUT_LINE (SQLERRM);
    DBMS_OUTPUT.PUT_LINE (SQLCODE);
end;
/
```

End-to-end examples:

To run end-to-end example scenarios, see Generate Text Response.

- Image to Text:

  The following statements generate a text response by making a REST call to OpenAI. Here, the input is an image (`sample_image.jpeg`) along with the prompt "`Describe this image?`".

```
-- select example

var input clob;
var media_data blob;
var media_type clob;
var params clob;

begin
  :input := 'Describe this image';
  :media_data := load_blob_from_file('DEMO_DIR', 'sample_image.jpeg');
  :media_type := 'image/jpeg';
  :params := '
{
```

```
  "provider"       : "openai",
  "credential_name": "OPENAI_CRED",
  "url"            : "https://api.openai.com/v1/chat/completions",
  "model"          : "gpt-4o-mini",
  "max_tokens"     : 60
}';
end;
/

select
dbms_vector_chain.utl_to_generate_text(:input, :media_data, :media_type,
json(:params));

-- PL/SQL example

declare
  input clob;
  media_data blob;
  media_type varchar2(32);
  params clob;
  output clob;

begin
  input := 'Describe this image';
  media_data := load_blob_from_file('DEMO_DIR', 'image_file');
  media_type := 'image/jpeg';
  params := '
{
  "provider"       : "openai",
  "credential_name": "OPENAI_CRED",
  "url"            : "https://api.openai.com/v1/chat/completions",
  "model"          : "gpt-4o-mini",
  "max_tokens"     : 60
}';

  output := dbms_vector_chain.utl_to_generate_text(
    input, media_data, media_type, json(params));
  dbms_output.put_line(output);

  if output is not null then
    dbms_lob.freetemporary(output);
  end if;
  if media_data is not null then
    dbms_lob.freetemporary(media_data);
  end if;
exception
  when OTHERS THEN
    DBMS_OUTPUT.PUT_LINE (SQLERRM);
    DBMS_OUTPUT.PUT_LINE (SQLCODE);
end;
/
```

End-to-end examples:

To run end-to-end example scenarios, see Describe Image Content.

UTL_TO_SUMMARY

Use the `DBMS_VECTOR_CHAIN.UTL_TO_SUMMARY` chainable utility function to generate a summary for textual documents.

A summary is a short and concise extract with key features of a document that best represents what the document is about as a whole. A summary can be free-form paragraphs or bullet points based on the format that you specify.

Purpose

To perform a text-to-summary transformation by accessing either Oracle Database or a third-party service provider:

- Oracle Database as the service provider (default setting):

  Uses the in-house implementation with Oracle Database, where Oracle Text is internally used to extract a summary (gist) from your document using the Oracle Text PL/SQL procedure `CTX_DOC.GIST`.

- Third-party summarization model:

  Makes a REST API call to your chosen remote service provider (Cohere, Generative AI, Google AI, Hugging Face, OpenAI, or Vertex AI) or local service provider (Ollama).

> **Note:**
>
> Currently, `UTL_TO_SUMMARY` does not work for Generative AI because the model and summary endpoint supported for Generative AI have been retired. It will be available in a subsequent release.

> **WARNING:**
>
> Certain features of the database may allow you to access services offered separately by third-parties, for example, through the use of JSON specifications that facilitate your access to REST APIs.
>
> Your use of these features is solely at your own risk, and you are solely responsible for complying with any terms and conditions related to use of any such third-party services. Notwithstanding any other terms and conditions related to the third-party services, your use of such database features constitutes your acceptance of that risk and express exclusion of Oracle's responsibility or liability for any damages resulting from such access.

Syntax

```
DBMS_VECTOR_CHAIN.UTL_TO_SUMMARY (
    DATA           IN CLOB,
    PARAMS         IN JSON default NULL
) return CLOB;
```

DATA

This function accepts the input data type in plain text as `CLOB`.

It returns a summary of the input document also as `CLOB`.

PARAMS

Specify summary parameters in JSON format, depending on the service provider that you want to use for document summarization.

If using Oracle Database as the provider:

```
{
  "provider"     : "database",
  "glevel"       : "<summary format>",
  "numParagraphs": <number in the range 1-16>,
  "maxPercent"   : <number in the range 1-100>,
  "num_themes"   : <number in the range 1-50>,
  "language"     : "<name of the language>"
}
```

Table 12-28 Database Provider Parameter Details

Parameter	Description
`provider`	Specify `database` (default setting) to access Oracle Database as the provider. Leverages the document `gist` or summary generated by Oracle Text.
`glevel`	Format to display the summary: • `SENTENCE` | `S`: As a list of sentences • `PARAGRAPH` | `P`: In a free-form paragraph
`numParagraphs`	Maximum number of document paragraphs (or sentences) selected for the summary. The default value is `16`. The `numParagraphs` parameter is used only when this parameter yields a smaller summary size than the summary size yielded by the `maxPercent` parameter, because the function always returns the smallest size summary.
`maxPercent`	Maximum number of document paragraphs (or sentences) selected for the summary, as a percentage of the total paragraphs (or sentences) in the document. The default value is `10`. The `maxPercent` parameter is used only when this parameter yields a smaller summary size than the summary size yielded by the `numParagraphs` parameter, because the function always returns the smallest size summary.
`num_themes`	Number of theme summaries to produce. For example, if you specify `10`, then this function returns the top 10 theme summaries. If you specify `0` or `NULL`, then this function returns all themes in a document. The default value is `50`. If the document contains more than 50 themes, only the top 50 themes show conceptual hierarchy.
`language`	Language name of your summary text, as listed in Supported Languages and Data File Locations.

For example:

```
{
  "provider"          : "database",
  "glevel"            : "sentence",
  "numParagraphs"     : 1
}
```

If using a third-party provider:

Set the following parameters along with additional summarization parameters specific to your provider:

```
{
  "provider"          : "<AI service provider>",
  "credential_name"   : "<credential name>",
  "url"               : "<REST endpoint URL for summarization service>",
  "model"             : "<REST provider summarization model name>",
  "transfer_timeout"  : <maximum wait time for the request to complete>,
  "max_count": "<maximum calls to the AI service provider>",
  "<additional REST provider parameter>": "<REST provider parameter value>"
}
```

Table 12-29 Third-Party Provider Parameter Details

Parameter	Description
`provider`	Third-party service provider that you want to access to get the summary. A REST call is made to the specified provider to access its text summarization model. Specify one of the following values: • `cohere` • `googleai` • `huggingface` • `ocigenai` • `openai` • `vertexai`
`credential_name`	Name of the credential in the form: *`schema.credential_name`* A credential name holds authentication credentials to enable access to your provider for making REST API calls. You need to first set up your credential by calling the `DBMS_VECTOR_CHAIN.CREATE_CREDENTIAL` helper function to create and store a credential, and then refer to the credential name here. See CREATE_CREDENTIAL.
`url`	URL of the third-party provider endpoint for each REST call, as listed in Supported Third-Party Provider Operations and Endpoints.
`model`	Name of the third-party text summarization model in the form: *`schema.model_name`* If the model name is not schema-qualified, then the schema of the procedure invoker is used. **Note**: For Generative AI, you must specify *`schema.model_name`*. All the third-party models supported for Generative AI are listed in Supported Third-Party Provider Operations and Endpoints.

Table 12-29 (Cont.) Third-Party Provider Parameter Details

Parameter	Description
`transfer_timeout`	Maximum time to wait for the request to complete. The default value is `60` seconds. You can increase this value for busy web servers.
`max_count`	Maximum number of times the API can be called for a given third-party provider. When set to an integer *n*, `max_count` stops the execution of the API for the given provider beyond *n* times. This prevents accidental calling of a third-party over some limit, for example to avoid surpassing the service amount that you have purchased.

Additional third-party provider parameters:

Optionally, specify additional provider-specific parameters.

Table 12-30 Additional REST Provider Parameter Details

Parameter	Description
`length`	Approximate length of the summary text: • `SHORT`: Roughly up to 2 sentences • `MEDIUM`: Between 3 and 5 sentences • `LONG`: 6 or more sentences • `AUTO`: The model chooses a length based on the input size **Note**: For Generative AI, you must enter this value in uppercase.
`format`	Format to display the summary: • `PARAGRAPH`: In a free-form paragraph • `BULLETS`: In bullet points **Note**: For Generative AI, you must enter this value in uppercase.
`temperature`	Degree of randomness used when generating output text, in the range of `0.0-5.0`. To generate the same output for a prompt, use `0`. To generate a random new text for that prompt, increase the temperature. Default temperature is `1` and the maximum temperature is `5`. **Note**: To summarize a text, start with the temperature set to `0`. If you do not require random results, a recommended temperature value is `0.2` for Generative AI and between `0` and `1` for Cohere. Use a higher value if for example you plan to perform a selection of the various summaries afterward. Do not use a high temperature for summarization because a high temperature encourages the model to produce creative text, which might also include hallucinations.
`extractiveness`	How much to reuse the input in the summary: • `LOW`: Summaries with low extractiveness tend to paraphrase. • `HIGH`: Summaries with high extractiveness lean toward reusing sentences verbatim. **Note**: For Generative AI, you must enter this value in uppercase.
`max_tokens`	Maximum number of tokens in the output text.
`topP`	Probability of tokens in the output, in the range of `0.0-1.0`. A lower value provides less random responses and a higher value provides more random responses.
`candidateCount`	Number of response variations to return, in the range of `1-4`.
`maxOutputTokens`	Maximum number of tokens to generate for each response.

> **Note:**
>
> When specifying the `length`, `format`, and `extractiveness` parameters for Generative AI, ensure to enter the values in uppercase letters.

Let us look at some example configurations for all third-party providers:

> **Important:**
>
> - The following examples are for illustration purposes. For accurate and up-to-date information on additional parameters to use, refer to your third-party provider's documentation.
> - For a list of all supported REST endpoint URLs, see Supported Third-Party Provider Operations and Endpoints.

Cohere example:

```
{
  "provider"          : "cohere",
  "credential_name"   : "COHERE_CRED",
  "url"               : "https://api.cohere.example.com/summarize",
  "model"             : "command",
  "length"            : "medium",
  "format"            : "paragraph",
  "temperature"       : 1.0
}
```

Generative AI example:

```
{
 "provider"           : "ocigenai",
 "credential_name"    : "OCI_CRED",
 "url"                : "https://generativeai.oci.example.com/summarizeText",
 "model"              : "cohere.command-r-16k",
 "length"             : "MEDIUM",
 "format"             : "PARAGRAPH"
}
```

Google AI example:

```
{
  "provider"          : "googleai",
  "credential_name"   : "GOOGLEAI_CRED",
  "url"               : "https://googleapis.example.com/models/",
  "model"             : "gemini-pro:generateContent",
  "generation_config" : {
    "temperature"     : 0.9,
    "topP"            : 1,
    "candidateCount"  : 1,
```

```
    "maxOutputTokens" : 256
  }
}
```

Hugging Face example:

```
{
  "provider"          : "huggingface",
  "credential_name"   : "HF_CRED",
  "url"               : "https://api.huggingface.example.co/models/",
  "model"             : "facebook/bart-large-cnn"
}
```

Ollama example:

```
{
  "provider"          : "ollama",
  "host"              : "local",
  "url"               : "http://localhost:11434/api/generate",
  "model"             : "phi3:mini"
}
```

OpenAI example:

```
{
  "provider"          : "openai",
  "credential_name"   : "OPENAI_CRED",
  "url"               : "https://api.openai.example.com",
  "model"             : "gpt-4o-mini",
  "max_tokens"        : 256,
  "temperature"       : 1.0
}
```

Vertex AI example:

```
{
  "provider"          : "vertexai",
  "credential_name"   : "VERTEXAI_CRED",
  "url"               : "https://googleapis.example.com/models/",
  "model"             : "gemini-1.0-pro:generateContent",
  "generation_config" : {
    "temperature"     : 0.9,
    "topP"            : 1,
    "candidateCount"  : 1,
    "maxOutputTokens" : 256
  }
}
```

Examples

- **Generate summary using Oracle Database**:

This statement specifies `database` as the provider. Here, the Oracle Text PL/SQL procedure `CTX_DOC.GIST` is internally called to generate a summary of an extract on "Transactions".

```
-- select example

set serveroutput on

var params clob;
begin
  :params := '
{
  "provider": "database",
  "glevel": "sentence",
  "numParagraphs": 1
}';
end;
/

select dbms_vector_chain.utl_to_summary(
 'A transaction is a logical, atomic unit of work that contains one or
more SQL statements.  An RDBMS must be able to group SQL statements so
that they are either all committed, which means they are applied to the
database, or all rolled back, which means they are undone.  An
illustration of the need for transactions is a funds transfer from a
savings account to a checking account. The transfer consists of the
following separate operations:
    1. Decrease the savings account.
    2. Increase the checking account.
    3. Record the transaction in the transaction journal.
    Oracle Database guarantees that all three operations succeed or fail
as a unit. For example, if a hardware failure prevents a statement in the
transaction from executing, then the other statements must be rolled back.
    Transactions set Oracle Database apart from a file system. If you
perform an atomic operation that updates several files, and if the system
fails halfway through, then the files will not be consistent. In contrast,
a transaction moves an Oracle database from one consistent state to
another. The basic principle of a transaction is "all or nothing": an
atomic operation succeeds or fails as a whole.',
 json(:params)) from dual;

-- PL/SQL example

declare
  input clob;
  params clob;
  output clob;
begin
  input := 'A transaction is a logical, atomic unit of work that contains
one or more SQL statements.  An RDBMS must be able to group SQL statements
so that they are either all committed, which means they are applied to the
database, or all rolled back, which means they are undone.  An
illustration of the need for transactions is a funds transfer from a
savings account to a checking account. The transfer consists of the
following separate operations:
    1. Decrease the savings account.
```

```
    2. Increase the checking account.
    3. Record the transaction in the transaction journal.
    Oracle Database guarantees that all three operations succeed or fail
as a unit. For example, if a hardware failure prevents a statement in the
transaction from executing, then the other statements must be rolled back.
    Transactions set Oracle Database apart from a file system. If you
perform an atomic operation that updates several files, and if the system
fails halfway through, then the files will not be consistent. In contrast,
a transaction moves an Oracle database from one consistent state to
another. The basic principle of a transaction is "all or nothing": an
atomic operation succeeds or fails as a whole.';

  params := '
  {
   "provider": "database",
   "glevel": "sentence",
   "numParagraphs": 1
  }';

  output := dbms_vector_chain.utl_to_summary(input, json(params));
  dbms_output.put_line(output);
  if output is not null then
    dbms_lob.freetemporary(output);
  end if;
exception
  when OTHERS THEN
    DBMS_OUTPUT.PUT_LINE (SQLERRM);
    DBMS_OUTPUT.PUT_LINE (SQLCODE);
end;
/
```

- **Generate summary using Generative AI**:

  These statements generate a summary of an extract on "Transactions" by accessing Generative AI as the provider.

  Here, the cohere.command-r-16k model is used for the summarization operation. You can replace the `model` value with any other supported model that you want to use with Generative AI, as listed in Supported Third-Party Provider Operations and Endpoints.

```
-- select example

var params clob;
begin
  :params := '
{
 "provider": "ocigenai",
 "credential_name": "OCI_CRED",
 "url": "https://inference.generativeai.us-chicago-1.oci.oraclecloud.com/
20231130/actions/chat",
 "model": "cohere.command-r-16k",
 "temperature": "0.0",
 "extractiveness": "LOW"
}';
end;
/
```

```
select dbms_vector_chain.utl_to_summary(
 'A transaction is a logical, atomic unit of work that contains one or
more SQL statements.  An RDBMS must be able to group SQL statements so
that they are either all committed, which means they are applied to the
database, or all rolled back, which means they are undone.  An
illustration of the need for transactions is a funds transfer from a
savings account to a checking account. The transfer consists of the
following separate operations:
    1. Decrease the savings account.
    2. Increase the checking account.
    3. Record the transaction in the transaction journal.
    Oracle Database guarantees that all three operations succeed or fail
as a unit. For example, if a hardware failure prevents a statement in the
transaction from executing, then the other statements must be rolled back.
    Transactions set Oracle Database apart from a file system. If you
perform an atomic operation that updates several files, and if the system
fails halfway through, then the files will not be consistent. In contrast,
a transaction moves an Oracle database from one consistent state to
another. The basic principle of a transaction is "all or nothing": an
atomic operation succeeds or fails as a whole.',
 json(:params)) from dual;

-- PL/SQL example

declare
  input clob;
  params clob;
  output clob;
begin
  input := 'A transaction is a logical, atomic unit of work that contains
one or more SQL statements.  An RDBMS must be able to group SQL statements
so that they are either all committed, which means they are applied to the
database, or all rolled back, which means they are undone.  An
illustration of the need for transactions is a funds transfer from a
savings account to a checking account. The transfer consists of the
following separate operations:
    1. Decrease the savings account.
    2. Increase the checking account.
    3. Record the transaction in the transaction journal.
    Oracle Database guarantees that all three operations succeed or fail
as a unit. For example, if a hardware failure prevents a statement in the
transaction from executing, then the other statements must be rolled back.
    Transactions set Oracle Database apart from a file system. If you
perform an atomic operation that updates several files, and if the system
fails halfway through, then the files will not be consistent. In contrast,
a transaction moves an Oracle database from one consistent state to
another. The basic principle of a transaction is "all or nothing": an
atomic operation succeeds or fails as a whole.';

  params := '
  {
   "provider": "ocigenai",
   "credential_name": "OCI_CRED",
   "url": "https://inference.generativeai.us-chicago-1.oci.oraclecloud.com/
20231130/actions/chat",
   "model": "cohere.command-r-16k",
```

```
    "length": "MEDIUM",
    "format": "PARAGRAPH",
    "temperature": 1.0
   }';

   output := dbms_vector_chain.utl_to_summary(input, json(params));
   dbms_output.put_line(output);
   if output is not null then
     dbms_lob.freetemporary(output);
   end if;
exception
   when OTHERS THEN
     DBMS_OUTPUT.PUT_LINE (SQLERRM);
     DBMS_OUTPUT.PUT_LINE (SQLCODE);
end;
/
```

- **End-to-end examples**:

  To run end-to-end example scenarios using this function, see Generate Summary.

UTL_TO_TEXT

Use the `DBMS_VECTOR_CHAIN.UTL_TO_TEXT` chainable utility function to convert an input document (for example, PDF, DOC, JSON, XML, or HTML) to plain text.

Purpose

To perform a file-to-text transformation by using the Oracle Text component (`CONTEXT`) of Oracle Database.

Syntax

```
DBMS_VECTOR_CHAIN.UTL_TO_TEXT (
    DATA          IN CLOB | BLOB,
    PARAMS        IN JSON default NULL
) return CLOB;
```

DATA

This function accepts the input data type as `CLOB` or `BLOB`. It can read documents from a remote location or from files stored locally in the database tables.

It returns a plain text version of the document as `CLOB`.

Oracle Text supports around 150 file types. For a complete list of all the supported document formats, see *Oracle Text Reference*.

PARAMS

Specify the following input parameter in JSON format:

```
{
    "plaintext" : "true or false",
    "charset"   : "UTF8"
}
```

Table 12-31 Parameter Details

Parameter	Description
`plaintext`	Plain text output. The default value for this parameter is `true`, that is, by default the output format is plain text. If you do not want to return the document as plain text, then set this parameter to `false`.
`charset`	Character set encoding. Currently, only `UTF8` is supported.

Example

```
select DBMS_VECTOR_CHAIN.UTL_TO_TEXT (
    t.blobdata,
     json('{
            "plaintext": "true",
            "charset"  : "UTF8"
           }')
) from tab t;
```

End-to-end example:

To run an end-to-end example scenario using this function, see Convert File to Text to Chunks to Embeddings Within Oracle Database.

Supported Languages and Data File Locations

These are the supported languages for which language data files are distributed by default in the specified directories.

Language Name	Abbreviation	Data File
AFRIKAANS	af	`ctx/data/eos/dreosaf.txt`
AMERICAN	us	`ctx/data/eos/dreosus.txt`
ARABIC	ar	`ctx/data/eos/dreosar.txt`
BASQUE	eu	`ctx/data/eos/dreoseu.txt`
BELARUSIAN	be	`ctx/data/eos/dreosbe.txt`
BRAZILIAN PORTUGUESE	ptb	`ctx/data/eos/dreosptb.txt`
BULGARIAN	bg	`ctx/data/eos/dreosbg.txt`
CANADIAN FRENCH	frc	`ctx/data/eos/dreosfrc.txt`
CATALAN	ca	`ctx/data/eos/dreosca.txt`
CROATIAN	hr	`ctx/data/eos/dreoshr.txt`
CYRILLIC SERBIAN	csr	`ctx/data/eos/dreoscsr.txt`
CZECH	cs	`ctx/data/eos/dreoscs.txt`
DANISH	dk	`ctx/data/eos/dreosdk.txt`

Language Name	Abbreviation	Data File
DARI	prs	ctx/data/eos/dreosprs.txt
DUTCH	nl	ctx/data/eos/dreosnl.txt
EGYPTIAN	eg	ctx/data/eos/dreoseg.txt
ENGLISH	gb	ctx/data/eos/dreosgb.txt
ESTONIAN	et	ctx/data/eos/dreoset.txt
FINNISH	sf	ctx/data/eos/dreossf.txt
FRENCH	f	ctx/data/eos/dreosf.txt
GALICIAN	ga	ctx/data/eos/dreosga.txt
GERMAN	d	ctx/data/eos/dreosd.txt
GERMAN DIN	din	ctx/data/eos/dreosdin.txt
GREEK	el	ctx/data/eos/dreosel.txt
HEBREW	iw	ctx/data/eos/dreosiw.txt
HINDI	hi	ctx/data/eos/dreoshi.txt
HUNGARIAN	hu	ctx/data/eos/dreoshu.txt
ICELANDIC	is	ctx/data/eos/dreosis.txt
INDONESIAN	in	ctx/data/eos/dreosin.txt
ITALIAN	i	ctx/data/eos/dreosi.txt
JAPANESE	ja	ctx/data/eos/dreosja.txt
KOREAN	ko	ctx/data/eos/dreosko.txt
LATIN AMERICAN SPANISH	esa	ctx/data/eos/dreosesa.txt
LATIN BOSNIAN	lbs	ctx/data/eos/dreoslbs.txt
LATIN SERBIAN	lsr	ctx/data/eos/dreoslsr.txt
LATVIAN	lv	ctx/data/eos/dreoslv.txt
LITHUANIAN	lt	ctx/data/eos/dreoslt.txt
MACEDONIAN	mk	ctx/data/eos/dreosmk.txt
MALAY	ms	ctx/data/eos/dreosms.txt
MEXICAN SPANISH	esm	ctx/data/eos/dreosesm.txt
NORWEGIAN	n	ctx/data/eos/dreosn.txt
NYNORSK	nn	ctx/data/eos/dreosnn.txt
PERSIAN	fa	ctx/data/eos/dreosfa.txt
POLISH	pl	ctx/data/eos/dreospl.txt
PORTUGUESE	pt	ctx/data/eos/dreospt.txt
ROMANIAN	ro	ctx/data/eos/dreosro.txt
RUSSIAN	ru	ctx/data/eos/dreosru.txt
SIMPLIFIED CHINESE	zhs	ctx/data/eos/dreoszhs.txt
SLOVAK	sk	ctx/data/eos/dreossk.txt
SLOVENIAN	sl	ctx/data/eos/dreossl.txt

Language Name	Abbreviation	Data File
SPANISH	e	ctx/data/eos/dreose.txt
SWEDISH	s	ctx/data/eos/dreoss.txt
THAI	th	ctx/data/eos/dreosth.txt
TRADITIONAL CHINESE	zht	ctx/data/eos/dreoszht.txt
TURKISH	tr	ctx/data/eos/dreostr.txt
UKRAINIAN	uk	ctx/data/eos/dreosuk.txt
URDU	ur	ctx/data/eos/dreosur.txt

Related Topics

- CREATE_LANG_DATA
  Use the `DBMS_VECTOR_CHAIN.CREATE_LANG_DATA` chunker helper procedure to load your own language data file into the database.

DBMS_HYBRID_VECTOR

The `DBMS_HYBRID_VECTOR` package contains a JSON-based query API `SEARCH`, which lets you query against hybrid vector indexes.

- SEARCH
  Use the `DBMS_HYBRID_VECTOR.SEARCH` PL/SQL function to run textual queries, vector similarity queries, or hybrid queries against hybrid vector indexes.
- GET_SQL
  Use the `DBMS_HYBRID_VECTOR.GET_SQL` PL/SQL function to return the internal SQL query that is generated from the parameters.
- SEARCHPIPELINE
  Use the standard table function `DBMS_HYBRID_VECTOR.SEARCHPIPELINE` to return a pipeline of row records.

SEARCH

Use the `DBMS_HYBRID_VECTOR.SEARCH` PL/SQL function to run textual queries, vector similarity queries, or hybrid queries against hybrid vector indexes.

Purpose

To search by vectors and keywords. This function lets you perform the following tasks:

- Facilitate a combined (hybrid) query of textual documents and vectorized chunks:

  You can query a hybrid vector index in multiple vector and keyword search combinations called search modes, as described in Understand Hybrid Search. This API accepts a JSON specification for all query parameters.
- Fuse and reorder the search results:

  The search results of a hybrid query are fused into a unified result set as `CLOB` using the specified fusion set operator, and reordered by a combined score using the specified scoring algorithm.
- Run a default query for a simplified search experience:

The minimum input parameters required are the `hybrid_index_name` and `search_text`. The same text string is used to query against a vectorized chunk index and a text document index.

Syntax

```
DBMS_HYBRID_VECTOR.SEARCH(
   json(
     '{  "hybrid_index_name"     :
"<hybrid_vector_index_name>",
         "search_text"           :  "<query string for keyword-and-semantic
search>",
         "search_fusion"         :  one of these values : "INTERSECT | UNION
| TEXT_ONLY | VECTOR_ONLY | MINUS_TEXT |
                                    MINUS_VECTOR | RERANK",
         "search_scorer"         :  one of these values : "RRF | RSF",
         "vector":
           {
            "search_text"             :  "<query string for semantic
search>",
            "search_vector"           :  "<vector_embedding>",
            "search_mode"             :  one of thse values : "DOCUMENT |
CHUNK",
            "aggregator"              :  one of these values : "COUNT | SUM |
MIN | MAX | AVG | MEDIAN | BONUSMAX | WINAVG |
                                       ADJBOOST | MAXAVGMED",
            "result_max"              :  <maximum number of vector results>,
            "score_weight"            :  <weight of vector score for RSF>,
            "rank_penalty"            :  <penalty of vector ranking for RRF>,
            "inpath"                  :  <an array of valid JSON paths>,
            "accuracy"                :  <target accuracy for semantic
search>,
            "index_probes"            :  <neighbor partitions for semantic
search>,
            "index_efsearch"          :  <efsearch for semantic search>,
            "filter_type"             :  one of these values : "IN_WO | IN_W
| PRE_WO | PRE_W | POST_WO | DEFAULT"
           },
         "text":
           {
            "contains"                :  "<query string for keyword
search>",
            "search_text"             :  "<alternative text to use to
construct a contains query automatically>",
            "json_textcontains"       :  <an array of valid JSON path and a
query string>,
            "score_weight"            :  <weight of text score for RSF>,
            "rank_penalty"            :  <penalty of text ranking for RRF>,
            "result_max:              :  <maximum number of document results>,
            "inpath"                  :  <array of valid JSON paths>
           },
         "filter_by":
           {
            "op"                      :  one of these values: "< | > | <= |
>= | = | != | ^- | <> | LIKE | LIKEC | LIKE2 | LIKE4 |
                                       REGEXP_LIKE | BETWEEN | EXISTS |
```

```
INSTR | INSTRC | INSTR2 | INSTR4 | STSTR | STSTR2 | STSTR4 |
                                          STSTRB | STSTRC | <ANY | >ANY |
<=ANY | >=ANY | =ANY | !=ANY | <SOME | >SOME |
                                          <=SOME | >=SOME | =SOME | !=SOME |
<ALL | >ALL | <=ALL | >=ALL | =ALL | !=ALL | IN |
                                          AND | OR | NOT | NOTOR |
NOTAND",
            "type"                    :  one of these values : "number |
string | date | timestamp",
            "col"                     :  "<base table column name>",
            "path"                    :  "<JSON path dot notation within a
base table JSON column>",
            "func"                    :  one of these values : "ABS | FLOOR |
LENGTH | CEILING | UPPER | LOWER | TO_BOOLEAN |
                                          TO_DATE | TO_DOUBLE |
TO_BINARYDOUBLE | TO_NUMBER | TO_CHAR | TO_TIMESTAMP",
            "args"                    :  <an array of arguments to the
operator>
          }
        "return":
          {
           "topN"                     :
<topN_value>,
            "values"                  :  one or more of these values : "rowid
| score | vector_score | text_score | vector_rank |
                                          text_rank | chunk_text | chunk_id |
paths",
            "format"                  :  one of these values : "JSON |
XML"
          }
    }'
  )
)
```

> **Note:**
>
> This API supports two constructs of search. One where you specify a single `search_text` field for both semantic search and keyword search (default setting). Another where you specify separate `search_text` and `contains` query fields using `vector` and `text` sub-elements for semantic search and keyword search, respectively. You cannot use both of these search constructs in one query.

hybrid_index_name

Specify the name of the hybrid vector index to use.

For information on how to create a hybrid vector index if not already created, see Manage Hybrid Vector Indexes.

search_text

Specify a search text string (your query input) for both semantic search and keyword search.

The same text string is used for a keyword query on document text index (by converting the `search_text` into a `CONTAINS ACCUM` operator syntax) and a semantic query on vectorized chunk index (by vectorizing or embedding the `search_text` for a `VECTOR_DISTANCE` search).

For example:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json('{ "hybrid_index_name" : "my_hybrid_idx",
            "search_text"       : "C, Python"
          }'))
FROM DUAL;
```

search_fusion

Specify a fusion sort operator to define what you want to retain from the combined set of keyword-and-semantic search results.

> **Note:**
>
> This search fusion operation is applicable only to non-pure hybrid search cases. Vector-only and text-only searches do not fuse any results.

Parameter	Description
`INTERSECT`	Returns only the rows that are common to both text search results and vector search results. Score condition: `text_score > 0 AND vector_score > 0`
`UNION` (default)	Combines all distinct rows from both text search results and vector search results. Score condition: `text_score > 0 OR vector_score > 0`
`TEXT_ONLY`	Returns all distinct rows from text search results plus the ones that are common to both text search results and vector search results. Thus, the fused results contain the text search results that appear in text search, including those that appear in both. Score condition: `text_score > 0`
`VECTOR_ONLY`	Returns all distinct rows from vector search results plus the ones that are common to both text search results and vector search results. Thus, the fused results contain the vector search results that appear in vector search, including those that appear in both. Score condition: `vector_score > 0`
`MINUS_TEXT`	Returns all distinct rows from vector search results minus the ones that are common to both text search results and vector search results. Thus, the fused results contain the vector search results that appear in vector search, excluding those that appear in both. Score condition: `text_score = 0`
`MINUS_VECTOR`	Returns all distinct rows from text search results minus the ones that are common to both text search results and vector search results. Thus, the fused results contain the text search results that appear in text search, excluding those that appear in both. Score condition: `vector_score = 0`
`RERANK`	Returns all distinct rows from text search ordered by the aggregated vector score of their respective vectors. There is no score condition for this field since the text search is followed by the use of the aggregated document vector scores.

For example:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json('{ "hybrid_index_name"     : "my_hybrid_idx",
            "search_fusion"         : "UNION",
            "vector":
                    { "search_text" : "leadership experience" },
             "text":
                    { "contains"    : "C and Python" }
          }'))
FROM DUAL;
```

search_scorer

Specify a method to evaluate the combined "fusion" search scores from both keyword and semantic search results.

- `RSF` (default) to use the Relative Score Fusion (RSF) algorithm
- `RRF` to use the Reciprocal Rank Fusion (RRF) algorithm

For a deeper understanding of how these algorithms work in hybrid search modes, see Understand Hybrid Search.

For example:

With a single search text string for hybrid search:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "search_text"       : "C, Python",
         "search_scorer"     : "rsf"
      }'))
FROM DUAL;
```

With separate vector and text search strings:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "search_scorer"     : "rsf",
         "vector":
          { "search_text"    : "leadership experience" },
         "text":
          { "contains"       : "C and Python" }
      }'))
FROM DUAL;
```

vector

Specify query parameters for semantic search against the vector index part of your hybrid vector index:

- `search_text`: Search text string (query text).

This string is converted into a query vector (embedding), and is used in a `VECTOR_DISTANCE` query to search against the vectorized chunk index.

For example:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json('{ "hybrid_index_name" : "my_hybrid_idx",
            "vector":
                    { "search_text" : "C, Python" }
          }'))
FROM DUAL;
```

- `search_vector`: Vector embedding (query vector).

  This embedding is directly used in a `VECTOR_DISTANCE` query to search against the vectorized chunk index.

  > **Note:**
  >
  > `search_vector` is an alternative to the above mentioned `search_text` when the semantic query is already available as a vector. The vector embedding that you pass here must be generated using the same embedding model used for semantic search by the specified hybrid vector index.

  For example:

```
SELECT JSON_SERIALIZE(
         DBMS_HYBRID_VECTOR.SEARCH(
            json_object( 'hybrid_index_name' value 'my_hybrid_idx',
                 'vector' value json_object( 'search_vector' value
vector_serialize(
                                              vector_embedding(doc_model
                                                          using
                                                          'C, Python,
Database'
                                                          as data)
                                                      RETURNING CLOB)
                                               RETURNING JSON)
                   RETURNING JSON))
           RETURNING CLOB PRETTY)
FROM dual;
```

- `search_mode`: Document or chunk search mode in which you want to query the hybrid vector index:

Parameter	Description
`DOCUMENT` (default)	Returns document-level results. In document mode, the result of your search is a list of document IDs from the base table corresponding to the list of best documents identified.
`CHUNK`	Returns chunk-level results. In chunk mode, the result of your search is a list of chunk identifiers and associated document IDs from the base table corresponding to the list of best chunks identified, regardless of whether the chunks come from the same document or different documents. The content from these chunk texts can be used as input for LLMs to formulate responses.

For example, semantic search in chunk mode:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "vector":
          {
             "search_text"   : "leadership experience",
             "search_mode"   : "CHUNK"
          }
      }'))
FROM DUAL;
```

- `aggregator`: Aggregate function to apply for ranking the vector scores for each document in `DOCUMENT SEARCH_MODE`.

Parameter	Description
`MAX` (default)	Standard database aggregate function that selects the top chunk score as the result score.
`AVG`	Standard database aggregate function that sums the chunk scores and divides by the count.
`MEDIAN`	Standard database aggregate function that computes the middle value or an interpolated value of the sorted scores.
`BONUSMAX`	This function combines the maximum chunk score with the remainder multiplied by the average score of the other top scores.
`WINAVG`	This function computes the maximum average of the rolling window (of size `windowSize`) of chunk scores.
`ADJBOOST`	This function computes the average "boosted" chunk score. The chunk scores are boosted with the `BOOSTFACTOR` multiplied by average score of the surrounding chunk's scores (if they exist).
`MAXAVGMED`	This function computes a weighted sum of the `MAX`, `AVGN`, and `MEDN` values.

For example:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "vector":
          {
             "search_text"   : "leadership experience",
             "search_mode"   : "DOCUMENT",
             "aggregator"    : "AVG"
          }
      }'))
FROM DUAL;
```

- `result_max`: The maximum number of vector results in distance order to fetch (approx) from the vector index.

  Value : Any positive integer greater then `0` (zero)

  Default: If the field is not specified, by default, the maximum is computed based on topN.

For Example:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "vector":
          {
             "search_text"   : "leadership experience",
             "search_mode"   : "DOCUMENT",
             "aggregator"    : "MAX",
             "score_weight"  : 5,
             "result_max"    : 100
          }
      }'))
FROM DUAL;
```

- `score_weight`: Relative weight (degree of importance or preference) to assign to the semantic `VECTOR_DISTANCE` query. This value is used when combining the results of RSF ranking.

  Value: Any positive integer greater than `0` (zero)

  Default: `10` (implies 10 times more importance to vector query than text query)

  For example:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "vector":
          {
             "search_text"   : "leadership experience",
             "search_mode"   : "DOCUMENT",
             "aggregator"    : "MAX",
             "score_weight"  : 5
          }
      }'))
FROM DUAL;
```

- `rank_penalty`: Penalty (denominator in RRF, represented as `1/(rank+penalty)`) to assign to vector query. This can help in balancing the relevance score by reducing the importance of unnecessary or repetitive words in a document. This value is used when combining the results of RRF ranking.

  Value: `0` (zero) or any positive integer

  Default: `1`

  For example:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "search_scorer"     : "rrf",
         "vector":
          {
             "search_text"   : "leadership experience",
             "search_mode"   : "DOCUMENT",
```

```
            "aggregator"    : "MAX",
            "score_weight"  : 5,
            "rank_penalty"  : 2
         }
      }'))
FROM DUAL;
```

- `inpath`: Valid JSON paths

  `vector.inpath` uses the vectorizer paths as you have in the document. Providing this parameter will restrict the search to the paths specified in this field. Accepts an array of paths in valid JSON format - (`$.a.b.c.d`).

  The list of paths match against VECTORIZER index path lists to form a query constraint on the vector index search. Simple wild cards on the paths are supported such as $.main.*.

  For example:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "search_scorer"     : "rrf",
         "vector":
          {
            "search_text"   : "leadership experience",
            "search_mode"   : "DOCUMENT",
            "aggregator"    : "MAX",
            "score_weight"  : 5,
            "rank_penalty"  : 2,
            "inpath"    : ["$.person.*", "$.product.*"]
          }
      }'))
FROM DUAL;
```

- `accuracy`: Target accuracy to assign to the semantic `VECTOR_DISTANCE` query.

  Value: Any positive integer between `0` (zero) and `100`.

  Default: `0`(zero). The value `0` indicates that the internal default for vector_distance query will be assigned to the field.

  For example:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "vector":
          {
            "search_text"   : "leadership experience",
            "search_mode"   : "DOCUMENT",
            "aggregator"    : "MAX",
            "score_weight"  : 5,
            "rank_penalty"  : 2,
            "inpath"        : ["$.person.*", "$.product.*"],
            "accuracy"      : 95
          }
      }'))
FROM DUAL;
```

- `index_probes`: Number of probes to assign to the semantic `VECTOR_DISTANCE` query.

  Value: Any positive integer greater than `0` (zero).

  Default: `0`(zero). The value `0` indicates that the internal default number of probes will be assigned to the field.

  For example:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "vector":
          {
             "search_text"   : "leadership experience",
             "search_mode"   : "DOCUMENT",
             "aggregator"    : "MAX",
             "score_weight"  : 5,
             "rank_penalty"  : 2,
             "inpath"        : ["$.person.*", "$.product.*"],
             "accuracy"      : 95,
             "index_probes"  : 3
          }
      }'))
FROM DUAL;
```

- `index_efsearch`: `efs` to assign to the semantic `VECTOR_DISTANCE` query.

  Value: Any positive integer greater than `0` (zero). The value `0` indicates that the internal default for vector_distance query will be assigned to the field.

  Default: `0`(zero)

  For example:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json(
      '{ "hybrid_index_name"   : "my_hybrid_idx",
         "vector":
          {
             "search_text"     : "leadership experience",
             "search_mode"     : "DOCUMENT",
             "aggregator"      : "MAX",
             "score_weight"    : 5,
             "rank_penalty"    : 2,
             "inpath"          : ["$.person.*", "$.product.*"],
             "accuracy"        : 95,
             "index_probes"    : 3,
             "index_efsearch"  : 500,
          }
      }'))
FROM DUAL;
```

- `filter_type`: Vector index hint filter type. For more information on optimizer plans, hints and filter types for vector indexes, please refer to Optimizer Plans for Vector Indexes and Vector Index Hints.

  Value: The `filter_type` field could take one of the following values:

- `PRE_W` - Pre-filter with join back. This applies only to HNSW indexes.
- `PRE_WO` - Pre-filter without join back. This applies to both HNSW and IVF indexes.
- `IN_W` - In-filter with join back. This applies only to HNSW indexes.
- `IN_WO` - In-filter without join back. This applies only to HNSW indexes.
- `POST_WO` - Post-filter without join back. This applies only to IVF indexes.

Default: No filter type hint.

For example:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json(
      '{ "hybrid_index_name"   : "my_hybrid_idx",
         "vector":
          {
             "search_text"     : "leadership experience",
             "search_mode"     : "DOCUMENT",
             "aggregator"      : "MAX",
             "score_weight"    : 5,
             "rank_penalty"    : 2,
             "inpath"          : ["$.person.*", "$.product.*"],
             "accuracy"        : 95,
             "index_probes"    : 3,
             "index_efsearch"  : 500,
             "filter_type"     : "IN_WO"
          }
      }'))
FROM DUAL;
```

text

Specify query parameters for keyword search against the Oracle Text index part of your hybrid vector index:

- `contains`: Search text string (query text).

  This string is converted into an Oracle Text `CONTAINS` query operator syntax for keyword search.

  You can use `CONTAINS` query operators to specify query expressions for full-text search, such as OR (`|`), AND (`&`), STEM (`$`), MINUS (`-`), and so on. For a complete list of all such operators to use, see *Oracle Text Reference*.

  For example:

  With a text contains string for pure keyword search:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json('{ "hybrid_index_name" : "my_hybrid_idx",
            "text":
                   { "contains" : "C and Python" }
          }'))
FROM DUAL;
```

  With separate search texts using `vector` and `text` sub-elements for hybrid search. One search text or a vector embedding to run a `VECTOR_DISTANCE` query for semantic search. A

second search text to run a `CONTAINS` query for keyword search. This query conducts two separate keyword and semantic queries, where keyword scores and semantic scores are combined:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json('{ "hybrid_index_name" : "my_hybrid_idx",
            "vector":
                    { "search_text" : "leadership experience" },
             "text":
                    { "contains" : "C and Python" }
          }'))
FROM DUAL;
```

- `search_text`: The alternative search text to use to construct a contains query automatically.

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json('{ "hybrid_index_name" : "my_hybrid_idx",
            "text":
                   { "contains"    : "C and Python",
                     "search_text" : "data science skills"
                   }
          }'))
FROM DUAL;
```

- `json_textcontains`: An alternate JSON expression to use instead of `contains` AND `search_text`.

> **Note:**
>
> It is an error to specify `json_textcontains` **WITH** either `text.contains` or `text.search_text`.

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json('{ "hybrid_index_name" : "my_hybrid_idx",
            "text":
                   { "json_textcontains"    : ["$.person", "$C
and $Python"]
                   }
          }'))
FROM DUAL;
```

- `score_weight`: Relative weight (degree of importance or preference) to assign to the text `CONTAINS` query. This value is used when combining the results of RSF ranking.

  Value: Any positive integer greater than `0` (zero)

  Default: `1` (implies neutral weight)

  For example:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
```

```
         "text":
          {
             "contains"      : "C and Python",
             "score_weight"  : 1
          }
      }'))
FROM DUAL;
```

- `rank_penalty`: Penalty (denominator in RRF, represented as `1/(rank+penalty)`) to assign to keyword query.

  This can help in balancing the relevance score by reducing the importance of unnecessary or repetitive words in a document. This value is used when combining the results of RRF ranking.

  Value: `0` (zero) or any positive integer

  Default: `5`

  For example:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "text":
          {
             "contains"      : "C and Python",
             "rank_penalty"  : 5
          }
      }'))
FROM DUAL;
```

- `inpath`: Valid JSON paths

  Providing this parameter will restrict the search to the paths specified in this field. Accepts an array of paths in valid JSON format - (`$.a.b.c.d`).

  For example:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "text":
          {
             "contains"      : "C and Python",
             "rank_penalty"  : 5,
             "inpath"    : ["$.person.*","$.product.*"]
          }
      }'))
FROM DUAL;
```

- `result_max`: The maximum number of document results (ordered by score) to retrieve from the document index. If not provided, the maximum is computed based on the topN.

  For example:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
```

```
            "text":
             {
                "contains"      : "C and Python",
                "rank_penalty"  : 5,
                "inpath"        : ["$.person.*","$.product.*"],
                "result_max"    : 100
             }
         }'))
FROM DUAL;
```

filter_by

To constrain the search results via standard relational logical constraints :

Parameter	Value
op	Logical comparison operator. Accepted values - One of these operators : • **Simple comparison operators** : '<', '>', '<=', '>=', '=', '!=', '^=', '<>', 'LIKE', 'LIKEC', 'LIKE2', 'LIKE4', 'INSTR', 'INSTR2', 'INSTR4','INSTRB', 'INSTRC', 'STSTR', 'STSTR2', 'STSTR4','STSTRB', 'STSTRC', 'REGEXP_LIKE', 'BETWEEN', 'EXISTS' **Note:** STSTR is the only non-standard operator in the list. It stands for "**START STRING**" and is analogous to INSTR, but the result must be equal to position 1. • **Group comparison operators** : – 18 combinations of '<', '>', '<=', '>=', '=', '!=' with ANY, SOME, ALL – IN • **Logical operators** : 'AND', 'OR', 'NOT', 'NOTAND', 'NOTOR'

	Note: "NOTAND" and "NOTOR" are short-hand for the following expressions, useful in reducing the JSON expression tree. NOTOR is NOT (arg1 OR arg2 ...). NOTAND is NOT (arg1 AND arg2 ...)
col	Base table column name. **Note:** • No column is required for logical operators. • Only one of col or path could be specified in the same element.
path	The JSON path dot notation within a base table JSON column.

	Note: • No `path` is required for logical operators. • Only one of `col` or `path` could be specified in the same element. • If the base table has a JSON column called `data`, then the syntax would be "`data.path`" where the path is case-sensitive matching the JSON data schema. For more details, see JSON dot notation.
`type`	The data type of the column. Accepted types include : `number, date, timestamp` and `string`.
`func`	For the comparison operators, an optional function can be applied to the column value before the comparison. These functions are the standard SQL functions. The one exception is "`TO_DOUBLE`" is provided as an alias to the full name "`TO_BINARY_DOUBLE`" Accepted values include : `ABS, FLOOR, LENGTH, CEILING, UPPER, LOWER, TO_BOOLEAN, TO_DATE, TO_DOUBLE, TO_BINARY_DOUBLE, TO_NUMBER, TO_CHAR, TO_TIMESTAMP`.
`args`	An array of arguments to the operator:

- For simple comparison operators, the `args` contains a single literal value. It is an error to provide 0 or more than 1 arguments.
- For group comparison operators, the `args` contains 1 or more literal values. It is an error to provide 0 arguments.
- For logical operators, the `args` contains sub-elements of the same structure, forming an expression tree.

For example: Using simple comparison operators

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json('{ "hybrid_index_name" : "my_hybrid_idx",
            "filter_by":
                    { "op"   : "<",
                      "col"  : "price",
                      "type" : "number",
                      "func" : "ABS"
                      "args" : ["10"] }
          }'))
FROM DUAL;
```

For example: Using group comparison operators

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json('{ "hybrid_index_name" : "my_hybrid_idx",
            "filter_by":
                    { "op"   :  "IN",
                      "path" : "DATA.brand",
                      "type" " "string",
                      "args" : ["nike", "adidas"] }
          }'))
FROM DUAL;
```

For example: Using logical operators

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json('{ "hybrid_index_name" : "my_hybrid_idx",
            "filter_by":
                    { "op"   :  "AND",
                      "args" : [
                      {"op" : "IN", "col" : "brand", "type" : "string",
"args" : ["nike", "adidas"]},
                      {"op" : "<", "col" : "price", "type" : "number",
"args" : ["10"]}]
                    }
          }'))
FROM DUAL;
```

return

Specify which fields to appear in the result set:

Parameter	Description
topN	Maximum number of best-matched results to be returned Value: Any integer greater than 0 (zero) Default: 20
values	Return attributes for the search results. Values for scores range between 100 (best) to 0 (worse). • rowid: Row ID associated with the source document. • score: Final score computed from keyword-and-semantic search scores. • vector_score: Semantic score from vector search results. • text_score: Keyword score from text search results. • vector_rank: Ranking of chunks retrieved from semantic or VECTOR_DISTANCE search. • text_rank: Ranking of documents retrieved from keyword or CONTAINS search. • chunk_text: Human-readable content from each chunk. • chunk_id: ID of each chunk text. • paths: Paths from which the result occurred. Default: All the above return attributes EXCEPT paths are shown by default. As there are no paths for non-JSON, you need to explicitly specify the paths field.
format	Format of the results as: • JSON (default) • XML

For example:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "search_text"       : "C, Python",
         "return":
          {
             "values"         : [ "rowid", "score", "paths" ],
             "topN"           : 3,
             "format"         : "JSON"
          }
      }'))
FROM DUAL;
```

Complete Example With All Query Parameters

The following example shows a hybrid search query that performs separate text and vector searches against my_hybrid_idx. This query specifies the search_text for vector search using the vector_distance function as prioritize teamwork and leadership experience and the keyword for text search using the contains operator as C and Python. The search mode is DOCUMENT to return the search results as topN documents.

```
SELECT JSON_SERIALIZE(
  DBMS_HYBRID_VECTOR.SEARCH(
```

```
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "search_fusion"     : "INTERSECT",
         "search_scorer"     : "rsf",
         "vector":
          {
             "search_text"   : "prioritize teamwork and leadership
experience",
             "search_mode"   : "DOCUMENT",
             "score_weight"  : 10,
             "rank_penalty"  : 1,
             "aggregator"    : "MAX",
             "inpath"        : ["$.main.body", "$.main.summary"],
             "accuracy"      : 95
          },
         "text":
          {
             "contains"      : "C and Python",
             "score_weight"  : 1,
             "rank_penalty"  : 5,
             "inpath"        : ["$.main.body"]
          },
         "return":
          {
             "format"        : "JSON",
             "topN"          : 3,
             "values"        : [ "rowid", "score", "vector_score",
                                 "text_score", "vector_rank",
                                 "text_rank", "chunk_text", "chunk_id",
"paths" ]
          }
      }'
    )
  ) pretty)
FROM DUAL;
```

The top 3 rows are ordered by relevance, with higher scores indicating a better match. All the return attributes are shown by default:

```
[
  {
    "rowid"         : "AAAR9jAABAAAQeaAAA",
    "score"         : 58.64,
    "vector_score"  : 61,
    "text_score"    : 35,
    "vector_rank"   : 1,
    "text_rank"     : 2,
    "chunk_text"    : "Candidate 1: C Master. Optimizes low-level system
(i.e. Database)
                      performance with C. Strong leadership skills in
guiding teams to
                      deliver complex projects.",
    "chunk_id"      : "1",
    "paths"         : ["$.main.body","$.main.summary"]
  },
```

```
  {
    "rowid"         : "AAAR9jAABAAAQeaAAB",
    "score"         : 56.86,
    "vector_score"  : 55.75,
    "text_score"    : 68,
    "vector_rank"   : 3,
    "text_rank"     : 1,
    "chunk_text"    : "Candidate 3: Full-Stack Developer. Skilled in
Database, C, HTML,
                      JavaScript, and Python with experience in building
responsive web
                      applications. Thrives in collaborative team
environments.",
    "chunk_id"      : "1",
    "paths"         : ["$.main.body", "$.main.summary"]
  },
  {
    "rowid"         : "AAAR9jAABAAAQeaAAD",
    "score"         : 51.67,
    "vector_score"  : 56.64,
    "text_score"    : 2,
    "vector_rank"   : 2,
    "text_rank"     : 3,
    "chunk_text"    : "Candidate 2: Database Administrator (DBA). Maintains
and secures
                      enterprise database (Oracle, MySql, SQL Server).
Passionate about
                      data integrity and optimization. Strong mentor for
junior DBA(s).",
    "chunk_id"      : "1",
    "paths"         : ["$.main.body", "$.main.summary"]
  }
]
```

End-to-end example:

To see how to create a hybrid vector index and explore all types of queries against the index, see Query Hybrid Vector Indexes End-to-End Example.

Related Topics

- Perform Hybrid Search

GET_SQL

Use the `DBMS_HYBRID_VECTOR.GET_SQL` PL/SQL function to return the internal SQL query that is generated from the parameters.

When calling the `DBMS_HYBRID_VECTOR` Search function, the API is called using the `JSON` Document format. Using the `GET_SQL` procedure shows the SQL that the `DBMS_HYBRID_VECTOR.SEARCH` API has generated. The resulting SQL can be used to view the query execution plan to view the index chosen for the hybrid search operation. An example is shown below:

```
SET LINESIZE 200;
SET PAGESIZE 1000;
```

```
SET TAB OFF;
SET TRIMSPOOL ON;
DECLARE
    res CLOB;
BEGIN
    res := dbms_hybrid_vector.get_sql(JSON('{"hybrid_index_name" :
"trecvol2j_idx",
                                           "search_text" : "offers",
                                           "return" : { "values" :
[ "score" ] } }'));
    execute immediate 'EXPLAIN PLAN FOR '||res;
END;
/


SELECT PLAN_TABLE_OUTPUT FROM TABLE(DBMS_XPLAN.DISPLAY(NULL,NULL,'ADVANCED'));
```

SEARCHPIPELINE

Use the standard table function `DBMS_HYBRID_VECTOR.SEARCHPIPELINE` to return a pipeline of row records.

This pipeline function accepts valid JSON query input and returns a pipeline of row records. The syntax is as shown below:

```
FUNCTION SEARCHPIPELINE(qparams JSON)
RETURN results PIPELINED;
```

The `results` is of type `RECORD`. The `results` contains the following fields:

Field	Type
`doc_rowid`	varchar2(18)
`score`	number
`vector_score`	number
`text_score`	number
`vector_rank`	number
`text_rank`	number
`chunk_text`	varchar2(32767)
`chunk_id`	varchar2(4000)
`paths`	varchar2(4000)

The record members are the column names in the `SELECT` statement. These names are the same as the JSON field names that are returned in `DBMS_HYBRID_VECTOR_SEARCH()`, except that `paths` is a list of field IDs in the record, where as the JSON result maps the ids to their actual paths (in an array). Also note that the result record could not have a member named `rowid` nor a member with a `rowid` type.

Example 12-1

```
SELECT
   chartorowid(doc_rowid) as doc_rowid,
   score,
   vector_score,
   text_score,
   vector_rank,
   text_rank,
   chunk_text,
   chunk_id,
   paths
FROM dbms_hybrid_vector.searchpipeline(JSON('{"hybrid_index_name" : "idx",
                                              "search_text" : "teamwork" }'));
```

If you do not wish to use the table function `DBMS_HYBRID_VECTOR.SEARCHPIPELINE()`, the original SEARCH API can be wrapped in a `JSON_TABLE` specification. This is shown in the example below:

```
SELECT jt.*
FROM
   JSON_TABLE(
        dbms_hybrid_vector.search(
                     json_object('hybrid_index_name' value 'idx',
                     'search_text' value 'teamwork'
                     RETURNING JSON)
                  ),
                  '$[*]' COLUMNS idx for ORDINALITY,
                               doc_rowid PATH '$.rowid',
                               score NUMBER PATH '$.score',
                               vector_score NUMBER PATH '$.vector_score',
                               text_score NUMBER PATH '$.text_score',
                               vector_rank NUMBER PATH '$.vector_rank',
                               text_rank NUMBER PATH '$.text_rank',
                               chunk_text PATH '$.chunk_text',
                               chunk_id PATH '$.chunk_id',
                               paths PATH '$.paths'
                   ) jt
ORDER by idx ASC
```

A

Python Classes to Convert Pretrained Models to ONNX Models (Deprecated)

This topic provides the functions, attributes and example usage of the deprecated python classes `EmbeddingModelConfig` and `EmbeddingModel`.

> **Note:**
>
> This API is updated for 23.7. The version used for 23.6 and below used python packages EmbeddingModel and EmbeddingModelConfig. These packages are replaced with ONNXPipeline and ONNXPipelineConfig respectively. Oracle recommends that you use the latest version. If a you choose to use a deprecated class, a warning message will be shown indicating that the classes will be removed in the future and advising the user to switch to the new class. You can find the details of the updated python classes in Import Pretrained Models in ONNX Format

Examples for converting pretrained text models to ONNX format using EmbeddingModel, EmbeddingModelConfig

1. To load the python classes on the OML4Py client :

   ```
   from oml.utils import EmbeddingModel, EmbeddingModelConfig
   ```

2. You can get a list of all preconfigured models by running the following:

   ```
   EmbeddingModelConfig.show_preconfigured()
   ```

3. To get a list of available templates:

   ```
   EmbeddingModelConfig.show_templates()
   ```

4. Generate an ONNX file from the preconfigured model "sentence-transformers/all-MiniLM-L6-v2":

   ```
   em = EmbeddingModel(model_name="sentence-transformers/all-MiniLM-L6-v2")
   em.export2file("your_preconfig_file_name",output_dir=".")
   ```

5. Generate an ONNX model from the preconfigured model "sentence-transformers/all-MiniLM-L6-v2" in the database:

   ```
   em = EmbeddingModel(model_name="sentence-transformers/all-MiniLM-L6-v2")
   em.export2db("your_preconfig_model_name")
   ```

6. Generate an ONNX file using the provided text template:

```
config = EmbeddingModelConfig.from_template("text",max_seq_length=512)
em = EmbeddingModel(model_name="intfloat/e5-small-v2",config=config)
em.export2file("your_template_file_name",output_dir=".")
```

Functions and Attributes of EmbeddingModelConfig

The `EmbeddingModelConfig` class contains the properties required for the package to perform downloading, exporting, augmenting, validation, and storing of an ONNX model. The class provides access to configuration properties using the dot operator. As a convenience, well-known configurations are provided as templates.

Parameters

This table describes the functions and properties of the `EmbeddingModelConfig` class.

Functions	Parameter Type	Returns	Description
`from_template(name, **kwargs)`	• `name` (`String`): The name of the template • `**kwargs`: template properties to override or add	Instance of `EmbeddingModelConfig`	A static function that creates an `EmbeddingModelConfig` object based on a predefined template given by the name parameter. You can use named arguments to override the template properties.
`show_templates()`	NA	List of existing templates	A static function that returns a list of existing templates by name.

Functions	Parameter Type	Returns	Description
`show_preconfigured()`	• `include_properties` (`bool,optional`): A flag indicating whether properties should be included in the results. Defaults to `False` so only names will be included by default. • `model_name` (`str,optional`): A model name to filter by when including properties. This argument will be ignored if `include_properties` is `False`. Otherwise only the properties of this model will be included in the results.	A list of preconfigured model names or properties.	Shows a list of preconfigured model names, or properties. By default, this function returns a list of names only. If the properties are required, pass the `include_properties` parameter as `True`. The returned list will contain a single dict where each key of the dict is the name of a preconfigured model and the value is the property set for that model. Finally, if only a single set of properties for a specific model is required, pass the name of the model in the `model_name` parameter (the `include_properties` parameter should also be `True`). This will return a list of a single dict with the properties for the specified model.

Template Properties

The text template has configuration properties shown below:

```
"do_lower_case": true,
"post_processors":[{"name":"Pooling","type":"mean"},{"name":"Normalize"}]
```

> **Note:**
>
> All other properties in the Properties table will take the default values. Any property without a default value must be provided when creating the `EmbeddingModelConfig` instance.

Properties

This table shows all properties that can be configured. preconfigured models already have these properties set to specific values. Templates will use the default values unless a user overrides it when using the `from_template` function on `EmbeddingModelConfig`.

Property	Description
post_processors	An array of post_processors that will be loaded after the model is loaded or initialized. The list of known and supported post_processors is provided later in this section. Templates may define a list of post_processors for the types of models they support. Otherwise, an empty array is the default.
max_seq_length	This property is applicable for text-based models only. The maximum length of input to the model as number of tokens. There is no default value. Specify this value for models that are not preconfigured.
do_lower_case	Specifies whether or not to lowercase the input when tokenizing. The default value is True.
quantize_model	Perform quantization on the model. This could greatly reduce the size of the model as well as speed up the process. It may however result in different results for the embedding vector (against the original model) and possibly small reduction in accuracy. The default value is False.
distance_metrics	An array of names of suitable distance metrics for the model. The names must be name of distance metrics used for Oracle vector distance operator. Only used when exporting the model to the database. Supported list is ["EUCLIDEAN","COSINE","MANHATTAN","HAMMING","DOT","EUCLIDEAN_SQUARED", "JACCARD"]. The default value is an empty array.
languages	An array of language (Abbreviation) supported in the Database. Only used when exporting the model to the database. For a supported list of languages, see Languages. The default value is an empty array.
use_float16	Specifies whether or not to convert the exported onnx model to float16. The default value is False.

Properties of post_processors

This table describes the built-in post_processors and their configuration parameters.

post_processor	Parameters	Description
Pooling	• name: Pooling. • type: Valid values should be mean(Default), max, cls	The Pooling post_processor summarizes the output of the transformer model into a fixed-length vector.
Normalize	• name: Specify Normalize	The Normalize post_processor bounds the vector values to a range using L2 normalization.

post_processor	Parameters	Description
`Dense`	• `name`: Dense • `in_features`: Input feature size • `out_features`: Output feature size • `bias`: Whether to learn an additive bias. The default value is `True`. • `activation_function`: Activation function of the dense layer. Currently only supports `Tanh` as the activation function.	Applies transformation to the incoming data.

Example: Configure post_processors

In this example, you override the post_processors in the sentence-transformers template with a `Max` Pooling post_processor followed by Normalization.

```
config = EmbeddingModelConfig.from_template("text")
config.post_processors = [{"name":"Pooling","type":"max"},
{"name":"Normalize"}]
```

Functions and Attributes of EmbeddingModel

Use the `EmbeddingModel` class to convert transformer models to the ONNX format with post_processing steps embedded into the final model.

Parameters

This table describes the signature and properties of the `EmbeddingModel` class.

Functions	Parameters	Description
`EmbeddingModel(model_name, configuration=None,settings={})`	• `model_name`: The name of the model to be used. For example, `medicalai/ClinicalBERT` • `configuration`: An initialized `EmbeddingModelConfig` object. This parameter must be specified when using a template. If not specified, the model will be assumed to be a preconfigured model. • `settings`: A dictionary of various settings that are global and control various operations such as logging levels and locations for files.	Creates a new instance of the `EmbeddingModel` class.

Settings

The settings object is a dictionary passed to the `EmbeddingModel` class. It provides global properties for the `EmbeddingModel` class that are used for non-model-specific operations, such as logging.

Property	Default Value	Description
`cache_dir`	`$HOME/.cache/OML`	The base directory used for downloads. Model files will be downloaded from the repository to directories relative to the `cache_dir`. If the `cache_dir` does not exist at time of execution, it will be created.
`logging_level`	`ERROR`	The level for logging. Valid values are ['DEBUG', 'INFO', 'WARNING', 'ERROR', 'CRITICAL']. **Note:** This log level is also applied globally to all python packages and is also mapped to the ONNXRuntime libraries.
`force_download`	`False`	Forces download of model files instead of reloading from cache.
`ignore_checksum_error`	`False`	Ignores any errors caused by mismatch in checksums when using preconfigured models.

Functions

This table describes the function and properties of the `EmbeddingModel` class.

Function	Parameters	Description
`export2file(export_name,output_dir=None)`	• `export_name(string)`: The name of the file. The file will be saved with the file extension `.onnx` • `output_dir(string)`: An optional output directory. If not specified the file will be saved to the current directory	Exports the model to a file.
`export2db(export_name)`	• `export_name(string)`: The name that will be used for the mining model object. This name must be compliant with existing rules for object names in the database.	Exports the model to the database.

Example: Preconfigured Model

This example illustrates the preconfigured embedding model that comes with the Python package. You can use this model without any additional configurations.

```
"sentence-transformers/distiluse-base-multilingual-cased-v2": {
        "max_seq_length": 128,
```

```
        "do_lower_case": false,
        "post_processors":[{"name":"Pooling","type":"mean"},
{"name":"Dense","in_features":768, "out_features":512, "bias":true,
"activation_function":"Tanh"}],
        "quantize_model":true,
        "distance_metrics": ["COSINE"],
        "languages": ["ar", "bg", "ca", "cs", "dk", "d", "us", "el", "et",
"fa", "sf", "f", "frc", "gu", "iw", "hi", "hr", "hu", "hy", "in", "i", "ja",
"ko", "lt",
                      "lv", "mk", "mr", "ms", "n", "nl", "pl", "pt", "ptb",
"ro", "ru", "sk", "sl", "sq", "lsr", "s", "th", "tr", "uk", "ur", "vn",
"zhs", "zht"]
    }
```

Glossary

dimension

A dimension refers to an array element of a vector.

dimension format

The dimensions of a vector can be represented using numbers of varying types and precisions, called the dimension format. The dimensions supported by vector embeddings in Oracle Database are `INT8` (1-byte signed integer), `FLOAT32` (4-byte single-precision floating point number), and `FLOAT64` (8-byte double-precision floating point number). All dimensions of a vector must have the same dimension format.

distance metric

Distance metric refers to the mathematical function used to compute distance between vectors. Popular distance metrics supported by Oracle AI Vector search include Euclidean distance, Cosine distance, and Manhattan distance, among others.

embedding model

Embedding models are Machine Learning algorithms that are trained to capture semantic information of unstructured data and represent it as vectors in multidimensional space. Different embedding models exist for different types of unstructured data, for example, BERT for Text data, ResNet-50 for Image data, and so on.

hybrid search

Hybrid search is an advanced information retrieval technique that lets you search documents by both *keywords* and *vectors*. Hybrid searches are run against hybrid vector indexes by querying it in various search modes. By integrating traditional keyword-based text search with vector-based similarity search, you can improve the overall search experience and provide users with more relevant information.

hybrid vector index

A hybrid vector index is a class of specialized Domain Index that combines the existing Oracle Text search indexes and Oracle AI Vector Search vector indexes into one unified structure. A

single index contains both textual and vector fields for a document, enabling you to perform a combination of keyword-based text search and vector-based similarity search simultaneously.

large language model

Large language models (LLMs) are advanced Machine Learning models designed to understand, process, and generate natural language for rich human interaction. They are typically built using deep learning algorithms and are pretrained on vast amounts of data. Popular examples include Open AI's GPT-4, Cohere's Command R+, and Meta's LLaMa 3.

multi-vector

Multi-vector refers to a scenario where multiple vectors correspond to a single entity. For example, a large text document can be chunked into paragraphs and every paragraph can be embedded into a separate vector. A similarity search query could retrieve matching documents based on the most similar paragraph (closest vectors) per document to a given query vector. Oracle AI Vector Search has the option of Partitioned Row Limiting Fetch syntax to enable efficient multi-vector searches.

neighbor graph

A neighbor graph is a graph-based data structure used in vector indexes. For example, a Hierarchical Navigable Small World (HNSW) vector index leverages a multilayer neighbor graph index. In a neighbor graph, each vertex of the graph represents a vector in the data set, and edges are created between vertices representing similar vectors.

query accuracy

Query accuracy is an intuitive indicator of the quality of an approximate query result obtained from a vector index search. Consider a query vector, for which an exact search, that searches through all vectors in the data set, returns Top 5 matches as `{ID1, ID3, ID5, ID7, ID9}`, and an approximate vector index search returns Top 5 matches as `{ID1, ID3, ID5, ID9, ID10}`. Since the approximate result has 4 out of 5 correct matches, the query accuracy is 80%.

query vector

Query vector refers to the vector embedding representing the item for which the user wants to find similar items using similarity search. For example, while searching for movies similar to a user's favorite movie, the vector embedding representing the user's favorite movie is the query vector.

Retrieval-Augmented Generation

Retrieval-Augmented Generation (RAG) is a popular technique for enhancing the accuracy of responses generated by Large Language Models by augmenting the user-provided prompt with relevant, up-to-date, enterprise-specific content retrieved using AI Vector Search.

Applications, such as Chat Assistants, built using RAG are often more accurate, reliable, and cost-effective.

similarity search

Similarity Search is a common operation in information retrieval to find items in a data set that are similar to a user-provided query item. For example, finding movies similar to a user's favorite movie is an example of similarity search. Vectors can enable efficient similarity searches by leveraging the property that the mathematical distance between vectors is a proxy for similarity, as in, the more similar two items are, the smaller the distance between the vectors.

vector

A vector is a mathematical entity that has a magnitude and a direction. It is typically represented as an array of numbers, which are coordinates that define its position in a multidimensional space.

vector distance

Vector distance refers to the mathematical distance between two vectors in a multidimensional space. The vector distance between similar items is smaller than the vector distance between dissimilar items. Vector distance is meaningful only if the vectors being compared are generated by the same embedding model.

vector embedding

A vector embedding is a numerical representation of text, image, audio, or video data that encodes the semantic content of the data, and not the underlying words or pixels. The terms *vector* and *vector embedding* are often used interchangeably in AI Vector Search.

vector index

Vector indexes are a class of specialized indexing data structures that are designed to accelerate similarity searches using high-dimensional vectors. They use techniques such as clustering, partitioning, and neighbor graphs to group vectors representing similar items, which drastically reduces the search space, thereby making the search process extremely efficient. Unlike traditional databases indexes, vector indexes enable approximate similarity searches, which allow users to trade off query accuracy for query performance to better suit application requirements.

Vector Memory Pool

Vector Memory Pool is a region of the System Global Area (SGA) that is dedicated to storing In-Memory Neighbor Graph Vector Indexes (HNSW index), as well as metadata for Neighbor Partition Vector Indexes. It can be specified by using the `VECTOR_MEMORY_SIZE` parameter.