# Oracle® Database Oracle Al Vector Search User's Guide





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

F87786-03

Copyright © 2023, 2024, Oracle and/or its affiliates.

Primary Author: Jean-Francois Verrier

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

Contributors: Aleksandra Czarlinska, Agnivo Saha, 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, Sebastian DeLaHoz, Shasank Chavan, Tirthankar Lahiri, Teck Hua Lee, Vinita Subramanian, Weiwei Gong, Yuan Zhou

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 form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software, software documentation, data (as defined in the Federal Acquisition Regulation), or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software," "commercial computer software documentation," or "limited rights data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle®, Java, MySQL, and NetSuite are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

## Contents

		_	
D	$r \alpha$	t 🔿	ce
		$\boldsymbol{\alpha}$	

Audience	vi
Documentation Accessibility	vi
Diversity and Inclusion	vi
Conventions	vii
Overview	
Overview of Oracle Al Vector Search	1-1
Why Use Oracle AI Vector Search?	1-5
Oracle AI Vector Search Workflow	1-6
Get Started	
SQL Quick Start	2-1
Generate Vector Embeddings	
Import Pretrained Models in ONNX Format for Vector Generation Within the Database	3-1
Import ONNX Models and Generate Embeddings	3-2
Alternate Method to Import ONNX Models	3-7
Convert Pretrained Models to ONNX Format	3-11
Convert Pretrained Models to ONNX Model: End-to-End Instructions	3-17
Python Classes to Convert Pretrained Models to ONNX Models	3-20
Generate Vector Embeddings Using Vector Utilities Leveraging Third-Party REST APIs	3-25
Understand the Stages of Data Transformations	3-26
Use SQL Functions to Generate Embeddings	3-27
Use PL/SQL Packages to Generate Embeddings	3-28
Terms of Using Vector Utility PL/SQL Packages	3-28
About Chainable Utility Functions and Common Use Cases	3-29
About Vector Helper Procedures	3-32
Supplied Vector Utility PL/SQL Packages	3-32
Supported Third-Party Provider Operations	3-33



	vector Generation Examples	3-31
	Generate Embeddings: SQL and PL/SQL Examples	3-37
	Convert Text String to Embedding	3-38
	Convert File to Text to Chunks to Embeddings	3-45
	Convert File to Embeddings	3-52
	Generate and Use Embeddings for End-to-End Search	3-53
	Perform Text Processing: PL/SQL Examples	3-61
	Convert Text String to Summary	3-61
	Create and Use Custom Vocabulary	3-70
	Create and Use Custom Language Data	3-72
	Perform Chunking: SQL and PL/SQL Examples	3-74
	Convert Text to Chunks With Custom Chunking Specifications	3-74
	Explore Chunking Techniques and Examples	3-78
	Generate Text for a Prompt: PL/SQL Example	3-94
4	Store Vector Embeddings	
	Create Tables Using the VECTOR Data Type	4-1
	Insert Vectors in a Database Table Using the INSERT Statement	4-3
	Load Vector Data Using SQL*Loader	4-7
	Load Character Vector Data Using SQL*Loader Example	4-8
	Load Binary Vector Data Using SQL*Loader Example	4-12
	Unload and Load Vectors Using Oracle Data Pump	4-13
5	Create Vector Indexes	
	Size the Vector Pool	5-1
	Manage the Different Categories of Vector Indexes	5-3
	In-Memory Neighbor Graph Vector Index	5-4
	Understand Hierarchical Navigable Small World Indexes	5-4
	Hierarchical Navigable Small World Index Syntax and Parameter	5-8
	Neighbor Partition Vector Index	5-8
	Understand Inverted File Flat Vector Indexes	5-9
	Inverted File Flat Index Syntax and Parameter	5-15
	Guidelines for Using Vector Indexes	5-16
	Index Accuracy Report	5-18
6	Use SQL Functions for Vector Operations	
	Vector Distance Functions	6-1
	Vector Distance Metrics	6-1
	Euclidean and Euclidean Squared Distances	6-2



Cosine Simil	iarity	6-3
Dot Product	Similarity	6-3
Manhattan D	Distance	6-4
Hamming Si	imilarity	6-5
Vector Distance	Operand to the VECTOR_DISTANCE Function	6-6
Shorthand Opera	ators for Distances	6-7
Other Basic Vector F	Functions	6-8
Vector Construct	tors	6-8
TO_VECTO	R	6-8
VECTOR		6-9
Parameters		6-9
Examples		6-9
Vector Serializer	rs .	6-10
FROM_VEC	CTOR	6-10
VECTOR_S	ERIALIZE	6-11
Parameters		6-11
Examples		6-11
VECTOR_NORM	M	6-12
VECTOR_DIME	NSION_COUNT	6-12
VECTOR_DIME	NSION_FORMAT	6-12
Oracle Al Vector Sea	arch SQL Functions	6-13
7 Query Data wit	th Similarity Searches	
Perform Exact Simila	arity Search	7-1
	e Similarity Search Using Vector Indexes	7-2
• • •	roximate Similarity Search Using Vector Indexes	7-3
	for Vector Indexes	7-5
·	ans for HNSW Vector Indexes	7-5
•	ans for IVF Vector Indexes	7-7
•	nilarity Search Examples	7-9
Approximate	e Search Using HNSW	7-9
• •	e Search Using IVF	7-11
Perform Multi-Vector	Similarity Search	7-12
8 Work with Retr	rieval Augmented Generation	
Compliment LLMs w	rith Oracle Al Vector Search	8-1
SQL RAG Example		8-3



## 9 Supported Clients and Languages

## 10 Vector Diagnostics

	Oracle AI Vector Search Views	10-1
	Vector Utilities-Related Views	10-1
	ALL_VECTOR_ABBREV_TOKENS	10-2
	ALL_VECTOR_LANG	10-2
	USER_VECTOR_ABBREV_TOKENS	10-2
	USER_VECTOR_LANG	10-3
	USER_VECTOR_VOCAB	10-3
	USER_VECTOR_VOCAB_TOKENS	10-3
	ALL_VECTOR_VOCAB	10-3
	ALL_VECTOR_VOCAB_TOKENS	10-4
	Vector Memory Pool Views	10-4
	V\$VECTOR_MEMORY_POOL	10-4
	Vector Index Views	10-5
	VECSYS.VECTOR\$INDEX	10-6
	Oracle Al Vector Search Statistics	10-8
	Oracle Al Vector Search Dictionary Statistics	10-8
	Oracle Machine Learning Static Dictionary Views	10-11
	Oracle Al Vector Search Parameters	10-11
11	Vector Search PL/SQL APIs	
	Oracle Al Vector Search PL/SQL Packages	11-1



## **Preface**

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

- Audience
- Documentation Accessibility
- Diversity and Inclusion
- Conventions

## **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 customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

## **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

## Overview

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

- Overview of Oracle Al 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 Al 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 Al Vector Search Workflow
   A typical Oracle Al Vector Search workflow follows the included primary steps.

## Overview of Oracle Al 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.

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

```
CREATE TABLE docs (INT doc id, CLOB doc text, VECTOR doc vector);
```

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

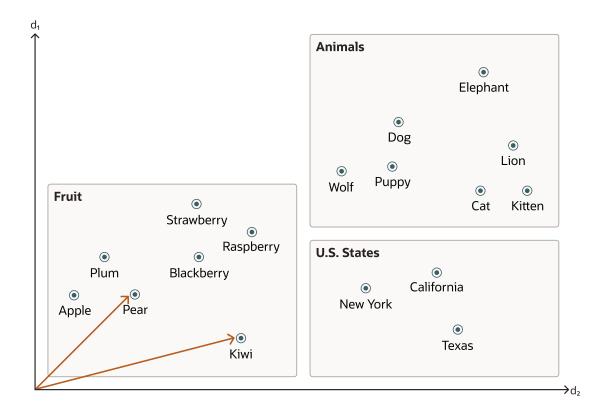
#### **Vector Embeddings**

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.

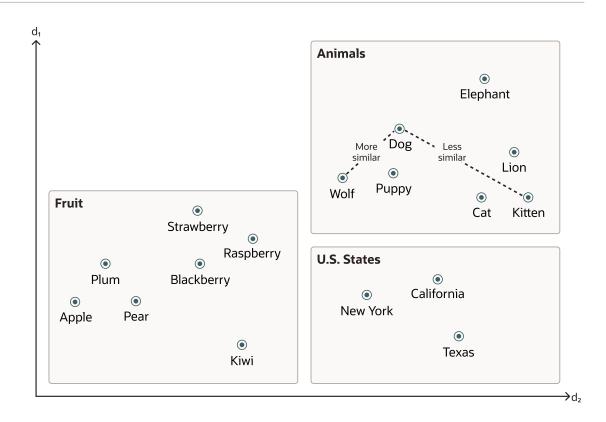
This simplified diagram illustrates a vector space where words are encoded as 2-dimensional vectors.



#### **Similarity Search**

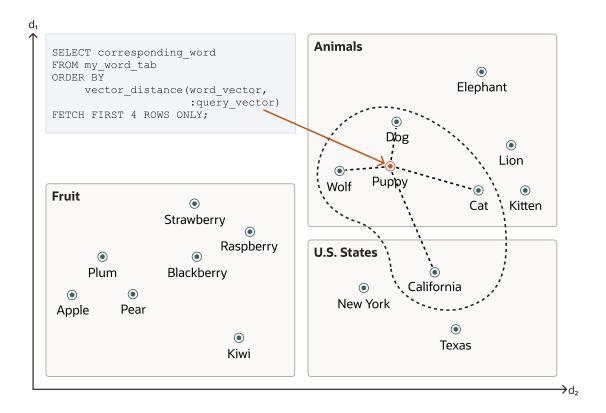
Searching semantic similarity in a data set is now equivalent to searching nearest neighbors in a vector space instead of using traditional keyword searches using query predicates. As illustrated in the following diagram, the distance between *dog* and *wolf* in this vector space is shorter than the 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.

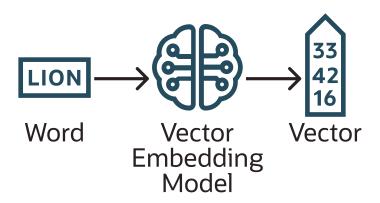
Approximate searches using *vector indexes* can limit the searches to specific clusters, whereas exact searches visit vectors across all clusters. See Use Vector Indexes for more information.

#### **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 1-1 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 Al 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 guery to answer your basic question:

```
SELECT house_photo, city, price
FROM house_for_sale
WHERE price <= :input price AND</pre>
```



```
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.

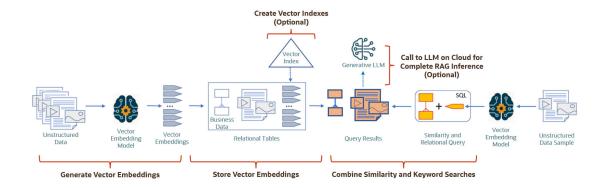
Al Smart Scan automatically accelerates Oracle Database 23ai Al Vector Search with optimizations that deliver low-latency parallelized scans across massive volumes of vector data. Al Smart Scan processes vector data at memory speed, leveraging ultra-fast Exadata RDMA Memory (XRMEM) and Exadata Smart 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 Al Vector Search Workflow

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

This is illustrated in the following diagram:

Figure 1-2 Oracle Al Vector Search Use Case Flowchart



To understand the diagram, consider this high level workflow description. Vector embeddings are generated by passing unstructured data through an embedding model. Vector embeddings can then be stored alongside business data in relational tables and vector indexes can optionally be created. Once you have the vector representations of your unstructured data stored in your database table(s), a sample of unstructured data can be passed through the embedding model to create a query vector. With the query vector, you can perform similarity searches against the vectors that are already stored in the database, in combination with relational queries if desired. To form a complete Retrieval Augmented Generation (RAG) pipeline, it is also possible to make a call to a generative Large Language Model (LLM) as part of the query step.

#### **Primary workflow steps:**

Generate Vector Embeddings from Your Unstructured Data
 You can perform this step either outside or within Oracle Database. For more information, see Generate Vector Embeddings.

#### Store Vector Embeddings, Unstructured Data, and Relational Business Data in Oracle Database

You store the resulting vector embeddings and associated unstructured data with your relational business data in Oracle Database. For more information, see Store Vector Embeddings.

#### 3. Create Vector Indexes

You may want to create vector indexes on your vector embeddings. This is beneficial for running similarity searches over huge vector spaces. For more information, see Create Vector Indexes.

#### 4. Query Data with Similarity Searches

You can then use Oracle AI Vector Search native SQL operations to combine similarity with relational searches to retrieve relevant data. For more information, see Query Data with Similarity Searches.

#### 5. Generate a Prompt and Send it to an LLM for a Full RAG Inference You can use the similarity search results to generate a prompt and send it to your generative LLM of choice for a complete RAG pipeline. For more information, see Work with Retrieval Augmented Generation.



## **Get Started**

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

SQL Quick Start
 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**

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

To convert pretrained models to ONNX model, see Convert Pretrained Models to ONNX Model: End-to-End Instructions.

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 Format.
- database-concepts23ai.pdf, which is the PDF file for Oracle Database 23ai Oracle Database Concepts manual.
- oracle-ai-vector-search-users-guide.pdf, which is the PDF file for this guide that you are reading.



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 directory.

There is no script and you can use scp. For example, to copy the three files to a directory on your server. You can call that directory /my local dir.

```
scp my_embedding_model.onnx /my_local_dir
scp database-concepts23ai.pdf /my_local_dir
scp oracle-ai-vector-search-users-guide.pdf /my local dir
```

Create storage, user, and privileges.

Here you create a new tablespace and a new user. You grant that user the <code>DB\_DEVELOPER\_ROLE</code> 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 the <code>DBMS\_VECTOR</code> 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 for Vector Generation Within the Database.

```
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', JSON('{"function" : "embedding", "embeddingOutput" :
"embedding", "input": {"input": ["DATA"]}}'));
```

4. 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',
'database-concepts23ai.pdf')));
insert into documentation_tab values(2, to_blob(bfilename('VEC_DUMP',
'oracle-ai-vector-search-users-guide.pdf')));
commit;
select dbms_lob.getlength(data) from documentation_tab;
```

5. 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 <code>DOCUMENTATION\_TAB</code>, 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 <code>INSERT SELECT</code> statement.

Here you choose to use Vector Utility PL/SQL package <code>DBMS\_VECTOR\_CHAIN</code> 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 <code>DBMS\_VECTOR\_CHAIN</code> 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
- 6. 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 Use 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
```

7. 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 <code>VECTOR\_DISTANCE</code> SQL function, see Vector Distance Functions.

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;
```

8. 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
                    I
                                  | 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 |
```

9. 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;
```



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.



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;
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';
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"
```

11. 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;
```

**12.** 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 <code>DOC\_CHUNKS</code> table that are similar to your query vector. For this, you use the <code>VECTOR\_DISTANCE</code> function and the <code>FETCH\_APPROX\_SQL</code> 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;
```

13. 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 |
      | 1 (0)| 00:00:01|
  5 | VECTOR INDEA 1110... -
| 1 (0)| 00:00:01 |
      VECTOR INDEX HNSW SCAN | DOCS HNSW IDX | 5014 |
19M|
______
```

14. 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.

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

## Generate Vector Embeddings

You must generate vector embeddings from your unstructured data either outside or within Oracle Database.

To get vector embeddings, you can either use ONNX embedding machine learning models or access third-party REST APIs.

- Import Pretrained Models in ONNX Format for Vector Generation Within the Database You can download pretrained embedding machine learning models, convert them into ONNX format if they are not already in ONNX format, import the ONNX format models into Oracle Database, and generate vector embeddings from your data within the database.
- Generate Vector Embeddings Using Vector Utilities Leveraging Third-Party REST APIs
   Oracle AI Vector Search offers vector utilities (SQL and PL/SQL tools) to automatically
   generate vector embeddings from your unstructured data.
- Vector Generation Examples
   Review these examples to see how you can generate vectors within and outside the database.

## Import Pretrained Models in ONNX Format for Vector Generation Within the Database

You can download pretrained embedding machine learning models, convert them into ONNX format if they are not already in ONNX format, import the ONNX format models into Oracle Database, and generate vector embeddings from your data within the database.

- Import ONNX Models and Generate Embeddings
   Learn to import a pretrained embedding model that is in ONNX format and generate vector embeddings.
- Convert Pretrained Models to ONNX Format
   OML4Py enables the use of text transformers from Hugging Face by converting them into
   ONNX format models. OML4Py also adds the necessary tokenization and post-processing.
   The resulting ONNX pipeline is then imported into the database and can be used to
   generate embeddings for AI Vector Search.

### Import ONNX Models and Generate Embeddings

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

Follow the steps below to import a pertained 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.

Choose from:

- 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.
- 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 cpassword>;
```

4. Grant CREATE MINING MODEL privilege to dmuser.

```
GRANT create mining model TO dmuser;
```

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;
```

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.



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.

See LOAD\_ONNX\_MODEL Procedure to learn about the PL/SQL procedure.

#### **Query Model Statistics**

You can view model attributes and learn about the model by querying machine learning dictionary views and model detail views.



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\$VMDOC 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\$VPDOC MODEL model detail view.

```
SELECT * FROM DM$VPDOC MODEL ORDER BY NAME;
```

**5.** Query the DM\$VJDOC\_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>@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"}'));
--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
_____
                   INPUT_STRING TEXT
ORA$ONNXTARGET VECTOR
DOC MODEL
DOC_WODET
                                                                VARCHAR2
                                                                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$VMDOC_MODEL ORDER BY NAME;
OUTPUT:
NAME
                                         VALUE
Graph Description
                                         Graph combining g_8_torch_jit and
torch
                                         jit
                                         g_8_torch_jit
                                         torch jit
Graph Name
                                         g_8_torch_jit_torch_jit
Input[0]
                                         input:string[1]
Output[0]
                                         embedding:float32[?,128]
Producer Name
                                         onnx.compose.merge models
Version
6 rows selected.
SQL> select * from DM$VPDOC MODEL ORDER BY NAME;
OUTPUT:
                                         VALUE
                                         False
batching
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-002,
-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-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
```

#### Oracle Al 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. The following procedure uses a PL/SQL helper block that facilitates the process of importing ONNX format model into the Oracle Database. The function reads the model file from the server's file system and imports it into the Database.

## Alternate Method to Import ONNX Models

Use the DBMS\_DATA\_MINING.IMPORT\_ONNX\_MODEL procedure to import the model and declare the input name. The following procedure uses a PL/SQL helper block that facilitates the process of importing ONNX format model into the Oracle Database. The function reads the model file from the server's file system and imports it into the Database.

Perform the following steps to import ONNX model into the Database using <code>DBMS\_DATA\_MINING</code> 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

- 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.

stored in the Oracle Database.

- "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 <code>DBMS\_DATA\_MINING.IMPORT\_ONNX\_MODEL</code> procedure can also accept a <code>BLOB</code> 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 <code>model\_source</code> variable as a <code>BLOB</code> type, initially set to NULL. It then retrieves an ONNX model from Oracle Cloud Object Storage using the <code>DBMS\_CLOUD.GET\_OBJECT</code> procedure, specifying the credentials (<code>OBJ\_STORE\_CRED</code>) and the URI of the model. The ONNX model resides in a specific bucket named <code>bucketname</code> in this case, and is accessible through the provided URL. Then, the script loads the ONNX model into the <code>model\_source BLOB</code>. The <code>DBMS\_DATA\_MINING.IMPORT\_ONNX\_MODEL</code> procedure then imports this model into the Oracle Database as <code>myonnxmodel</code>. During the import, a JSON metadata specifies the model's function as <code>embedding</code>, 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

VECTOR INFO

INPUT\_STRING TEXT
ORA\$ONNXTARGET VECTOR DOC MODEL VARCHAR2 DOC\_MODEL 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
NATIVE 17762137 EMBEDDING ONNX

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]

embedding:float32[?,128] Output[0]

```
Producer Name
                                                                                                                               onnx.compose.merge models
Version
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
5.32859489E-002,6.57707453E-002,-1.47056757E-002,-4.18472625E-002,4.1588001E-0
-2.86354572E-002, -7.56499246E-002, -4.16395674E-003, -1.52879998E-001, 6.60010576
-3.9013084E-002, 3.15719917E-002, 1.2428958E-002, -2.47651711E-002, -1.16851285E-002, -1.16851285E-0
-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
```

## Convert Pretrained Models to ONNX Format

OML4Py enables the use of text transformers from Hugging Face by converting them into ONNX format models. OML4Py also adds the necessary tokenization and post-processing.

The resulting ONNX pipeline is then imported into the database and can be used to generate embeddings for AI Vector Search.



This feature will **only** work on OML4Py client. It is not supported on the OML4Py server.

If you do not have a pretrained embedding model in ONNX-format to generate embeddings for your data, Oracle offers a Python utility 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 OML4Py's export2db() function to import the file as a mining model.. Then leverage the in-database ONNX Runtime with the ONNX model to produce vector embeddings.

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

- Downloads the pretrained model from external source 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 mining model or optionally exports to a file

The Python utility 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-based models. To understand what a preconfigured list, what is a built-in template is, and how to use them, read further.

#### Limitations

This table describes the limitations of the Python utility package.



This feature is available with the OML4Py client only.

Parameter	Description
Transformer Model Type	Currently supported only for text transformers.
Model Size	Model size should be less than 1GB. Quantization can help reduce the size.
Tokenizers	Must be either BERT, GPT2, SENTENCEPIECE, or ROBERTA.

#### **Preconfigured List of Models**

Preconfigured list of models are common models from external resource repositories that are provided with the Python utility. 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 <code>show\_preconfigured</code> function.



#### **Templates**

The Python utility 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.

The Python utility package provides the following classes:

- EmbeddingModelConfig
- EmbeddingModel

To learn more about the Python classes, their properties, and to configure the properties, see Python Classes to Convert Pretrained Models to ONNX Models.

To use the Python utility, ensure that you have the following:

- OML4Py Client running on Linux X64 for On-Premises Databases
- Python 3.12 (the earlier versions are not compatible)
- Start Python in your work directory.

```
Python3

Python 3.12.2 | (main, Feb 27 2024, 17:35:02) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
```

On the OML4Py client, load the Python classes:

```
from oml.utils import EmbeddingModel, EmbeddingModelConfig
```

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

```
EmbeddingModelConfig.show preconfigured()
```

4. To get a list of available templates:

```
EmbeddingModelConfig.show templates()
```

- 5. Choose from:
  - Generate an ONNX file from the preconfigureded model "sentence-transformers/all-MiniLM-L6-v2":

```
#generate from preconfigureded 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=".")
```



 Generate an ONNX model from the preconfigured model "sentence-transformers/all-MiniLM-L6-v2" in the database:

```
#generate from preconfigureded model "sentence-transformers/all-MiniLM-
L6-v2"
em = EmbeddingModel(model_name="sentence-transformers/all-MiniLM-L6-v2")
em.export2db("your preconfig model name")
```

Generate an ONNX file using the provided text template:

```
#generate using the "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=".")
```

#### Let's understand the code:

```
from oml.utils import EmbeddingModel, EmbeddingModelConfig
```

This line imports two classes, EmbeddingModel and EmbeddingModelConfig.

#### In the preconfigured models first example:

- em = EmbeddingModel (model\_name="sentence-transformers/all-MiniLM-L6-v2") creates an instance of the EmbeddingModel class, loading a pretrained model specified by the model\_name parameter. em is the 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 database. 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:

- em = EmbeddingModel (model\_name="sentence-transformers/all-MinilM-L6-v2") creates an instance of the EmbeddingModel class, loading a pretrained model specified by the model\_name parameter. em is the 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:

config = EmbeddingModelConfig.from\_template("text", max\_seq\_length=512):
 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.



- em = EmbeddingModel (model\_name="intfloat/e5-small-v2", config=config) initializes an EmbeddingModel instance with a specific model and the previously defined configuration. The model\_name="intfloat/e5-small-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 database. your\_template\_file\_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 <code>export2file</code> is used. If <code>export2db</code> 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.** Verify the models is in the database:



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

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.

- Convert Pretrained Models to ONNX Model: End-to-End Instructions
   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.
- Python Classes to Convert Pretrained Models to ONNX Models
   Explore the functions and attributes of the EmbeddingModelConfig class and
   EmbeddingModel class within Python. These classes are designed to configure pretrained embedding models.

#### 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

#### Convert Pretrained Models to ONNX Model: End-to-End Instructions

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.

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.

#### 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 $HOME/python wget https://www.python.org/ftp/python/3.12.0/Python-3.12.0.tgz tar -xvzf Python-3.12.0.tgz --strip-components=1 -C /home/$USER/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 symlimk 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.

Create a file named requirements.txt that contains the required thid-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==2.0.1
scikit-learn==1.4.1post1
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
```



Install OML4Py client. Download OML4Py 2.0 client from OML4Py download page and upload it to the Linux machine.

```
unzip oml4py-client-linux-x86_64-2.0.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 EmbeddingModelConfig from oml.utils.

```
python3
from oml.utils import EmbeddingModelConfig
EmbeddingModelConfig.show preconfigured()
['sentence-transformers/all-mpnet-base-v2', 'sentence-transformers/all-
MiniLM-L6-v2',
      'sentence-transformers/multi-ga-MiniLM-L6-cos-v1', 'ProsusAI/
finbert',
      'medicalai/ClinicalBERT', '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', 'prajjwal1/bert-tiny',
      'thenlper/gte-base', 'thenlper/gte-small', 'TaylorAI/gte-tiny',
'infgrad/stella-base-en-v2',
      'sentence-transformers/paraphrase-multilingual-mpnet-base-v2',
      'intfloat/multilingual-e5-base', 'intfloat/multilingual-e5-small',
      'sentence-transformers/stsb-xlm-r-multilingual']
```

#### 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 EmbeddingModel from oml.utils.

```
from oml.utils import EmbeddingModel

# Export to file
em = EmbeddingModel(model_name="sentence-transformers/all-MiniLM-L6-v2")
em.export2file("all-MiniLM-L6-v2", output dir=".")
```

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





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 EmbeddingModel
   \# 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")
  em = EmbeddingModel(model name="sentence-transformers/all-MiniLM-L6-v2")
  em.export2db("ALL MINILM L6")
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';
   ______
                   ALGORITHM
  MODEL NAME
                                                    MINING FUNCTION
   ______
                         ONNX
                                                     EMBEDDING
  ALL MINILM L6
```

## Python Classes to Convert Pretrained Models to ONNX Models

Explore the functions and attributes of the EmbeddingModelConfig class and EmbeddingModel class within Python. These classes are designed to configure pretrained embedding models.

#### 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
<pre>from_template(name,  **kwargs)</pre>	<ul> <li>name (String): The name of the template</li> <li>**kwargs: template properties to override or add</li> </ul>	<pre>Instance of EmbeddingModelConfi g</pre>	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.
<pre>show_templates()</pre>	NA	List of existing templates	A static function that returns a list of existing templates by name.
<pre>show_preconfigured( )</pre>	<ul> <li>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.</li> <li>model_name         (str, optional): A         model name to filter         by when including         properties. This         argument will be         ignored if         include_propert         ies is False.         Otherwise only the         properties of this         model will be         included in the         results.</li> </ul>	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"}]
```





All other properties in the Properties table will take the default values. Any property without a default value must be provided when creating the  ${\tt EmbeddingModelConfig}$  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"]. The default value is an empty array.
languages	A 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	<ul> <li>name: Pooling.</li> <li>type: Valid values should be mean(Default), max, cls</li> </ul>	The Pooling post_processor summarizes the output of the transformer model into a fixed-length vector.



post_processor	Parameters	Description
Normalize	• name: Specify Normalize	The Normalize post_processor bounds the vector values to a range using L2 normalization.
Dense	• name: Dense	Applies transformation to the
	<ul><li>in_features: Input feature size</li></ul>	incoming data.
	<ul> <li>out_features: Output feature size</li> <li>bias: Whether to learn an additive bias. The default value is True.</li> </ul>	
	<ul> <li>activation_function:         Activation function of the dense layer. Currently only supports Tanh as the activation function.     </li> </ul>	

#### **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"}]
```

#### **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.

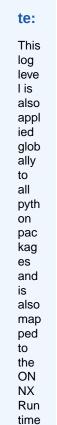
Functions	Parameters	Description
<pre>EmbeddingModel(model_name, configuration=None, setting s={})</pre>	<del>_</del>	Creates a new instance of the EmbeddingModel class.
	<ul> <li>configuration: An initialized         EmbeddingModelConfig object. This parameter must be specified when using a template. If not specified, the model will be assumed to be a preconfigured model.     </li> <li>settings: A dictionary of various settings that are global and control various operations such as logging levels and locations for files.</li> </ul>	



#### 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'].



libra ries.

No

force download	False	Forces download of model files
_		instead of reloading from cache.



Property	Default Value	Description
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
<pre>export2file(export_name,ou tput_dir=None)</pre>	export_name(string):     The name of the file. The file will be saved with the file extension .onnx	Exports the model to a file.
	<ul> <li>output_dir(string): An optional output directory. If not specified the file will be saved to the current directory</li> </ul>	
<pre>export2db(export_name)</pre>	<ul> <li>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.</li> </ul>	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"]
}
```

# Generate Vector Embeddings Using Vector Utilities Leveraging Third-Party REST APIs

Oracle AI Vector Search offers vector utilities (SQL and PL/SQL tools) to automatically generate vector embeddings from your unstructured data.

You can call either *Vector Utility SQL functions* or *Vector Utility PL/SQL packages* to transform unstructured data into vector embeddings.

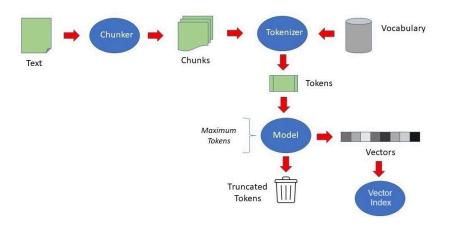
- Understand the Stages of Data Transformations
   Your input data may travel through different stages before turning into a vector.
- Use SQL Functions to Generate Embeddings
   Choose to implement Vector Utility SQL functions to perform parallel or on-the-fly chunking and embedding operations, within Oracle Database.
- Use 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, within and
   outside Oracle Database.

## Understand the Stages of Data Transformations

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

For example, you may first transform textual data (such as a PDF document) into plain text, then break the resulting text into smaller pieces of text (chunks) to finally create vector embeddings on each chunk.

As shown in the following image, input data passes through a pipeline of optional stages from *Text* to *Chunks* to *Tokens* to *Vectors*, with *Vector Index* as the endpoint:



#### **Prepare: 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 first transform the data into plain text and then pass the resulting text through *Chunker*. The chunker then splits the text into smaller chunks using a process known as **chunking**. A single document may be split into many chunks, each transformed into a vector.

**Chunks** are pieces of words (to capture specific words or word pieces), sentences (to capture a specific meaning), or paragraphs (to capture broader themes). Later, you will learn about several chunking parameters and techniques to define so that each chunk contains relevant and meaningful context.

#### **Embed: Tokens and Vectors**

You now pass the extracted chunks as input to *Model* (a declared vector embedding model) for generating vector embeddings on each chunk. This stage makes these chunks searchable.

The chunks are first passed through *Tokenizer* of the model to split into words or word pieces, known as **tokens**. The model then embeds each token into a vector representation.

Tokenizers used by embedding models frequently have limits on the size of the input text they can deal with, so chunking is needed to avoid loss of text when generating embeddings. If input text is larger than the maximum input limit imposed by the model, then the text gets truncated unless it is split up into appropriate-sized segments or chunks. Chunking is not needed for smaller documents, text strings, or summarized texts that meet 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 the 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, that is, the BERT multilingual model uses Word-Piece encoding and the GPT model uses Byte-Pair encoding.

For example, the BERT model tokenizes the following four words:

Embedding usecase for chunking

as the following eight tokens and also includes ## (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 when chunking.

#### Populate and Query: Vector Indexes

Finally, you store the semantic content of your input data as vectors in *Vector Index*. You can now implement combined similarity and relational queries to retrieve relevant results using the extracted vectors.

## Use SQL Functions to Generate Embeddings

Choose to implement Vector Utility SQL functions to perform parallel or on-the-fly chunking and embedding operations, within Oracle Database.

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

To get chunks, this function uses the in-house implementation with Oracle Database. To get an embedding, this function uses ONNX embedding models that you load into the database (and not third-party REST providers).

#### **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 amount of text into smaller yet semantically meaningful chunks. You can experiment with your chunks by inspecting and accordingly amending the chunking results and then proceed further.



For detailed information, see VECTOR\_CHUNKS in *Oracle Database SQL Language Reference*.

#### **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 can generate a query vector on the fly from a user's natural language text input.

For detailed information, see VECTOR\_EMBEDDING in *Oracle Database SQL Language Reference*.

## Use 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, within and outside Oracle Database.

Vector Utility PL/SQL APIs work with both the ONNX embedding models (loaded into the database) and third-party REST providers, such as Cohere, Google AI, Hugging Face, Oracle Cloud Infrastructure (OCI) Generative AI, OpenAI, or Vertex AI. You can create, run, and schedule end-to-end data transformation and search pipelines.

These packages are made up of subprograms, such as *chainable utility functions* and *vector helper procedures*.

- 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.
- 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 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.
- Supported Third-Party Provider Operations
   Review the list of third-party REST providers that are supported with Vector Utility PL/SQL packages and the corresponding API calls allowed for each of those.
- 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.

## 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.

## 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
UTL_TO_TEXT()	Converts a document (for example, MS Word, HTML, or PDF) to plain text
UTL_TO_CHUNKS()	Converts plain text to chunks
UTL_TO_EMBEDDING()	Converts plain 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)
UTL_TO_SUMMARY()	Converts plain text to a summary
UTL_TO_GENERATE_TEXT()	Generates text for a prompt or input string

#### **Sequence of Chains**

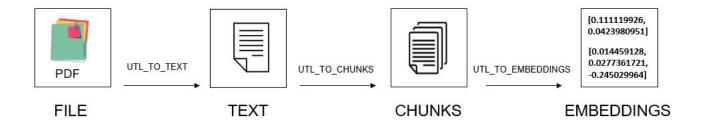
Chainable utility functions are designed to be flexible and modular. You can create transformation chains in various sequences, depending on your use case.

For example, you can directly extract vectors from a 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:



- Converts a PDF file to plain text (using UTL TO TEXT)
- 2. Splits the resulting text into appropriate-sized chunks (using UTL TO CHUNKS)
- Generates vector embeddings on each chunk (using UTL TO EMBEDDINGS)



#### **Common Use Cases**

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

#### Single-Step or Direct Transformation:

#### Document to vectors:

As discussed earlier, a common use case might be to automatically generate vectors from documents.

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 Convert File to 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  $\mathtt{UTL\_TO\_EMBEDDING}$ ) or chunk it and then generate multiple vectors (using  $\mathtt{UTL\_TO\_EMBEDDINGS}$ ).

- 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.

#### Prompt to text:



You can generate a response directly based on a prompt.

A prompt can be an input string, such as a question that you ask a Large Language Model (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.

See Generate Text for a Prompt: PL/SQL Example.

#### Step-by-Step or Parallel Transformation:

#### Text to vector:

A common use case might be information retrieval applications or chatbots, where you can on the fly generate an embedding from a user's natural language text guery.

You can convert the input text to a query vector (and then run it against the vector index for a fast similarity search), in a *text-to-embedding* chain.

See Convert Text String to Embedding.

#### 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.

#### Text to summary:

You can build a standalone Text Summarization system to convert a large amount of text into a summary, 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 Convert Text String to Summary.

#### 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 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 the database.
DROP_CREDENTIAL	Drops an existing credential name.

• 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 Utilities-Related 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 ONNX embedding model that you load into the database
		Third-party REST providers	Calls the specified third- party embedding model
	UTL_TO_GENERATE_TEXT to generate text for prompts	Third-party REST providers	Calls the specified third- party text generation 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 in *Oracle Database PL/SQL Packages and Types Reference*.

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, using functionality that can be pipelined together for an end-to-end search.

This 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.

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 ONNX embedding model that you load into 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	Third-party REST providers	Calls the specified third-party text generation 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
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

Due to underlying dependance on the text processing capabilities of Oracle Text, note that both the  $\mathtt{UTL}\_\mathtt{TO}\_\mathtt{TEXT}$  and  $\mathtt{UTL}\_\mathtt{TO}\_\mathtt{SUMMARY}$  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 in *Oracle Database PL/SQL Packages and Types Reference*.

#### **Related Topics**

Supported Third-Party Provider Operations
 Review the list of third-party REST providers that are supported with Vector Utility PL/SQL packages and the corresponding API calls allowed for each of those.

## Supported Third-Party Provider Operations

Review the list of third-party REST providers that are supported with Vector Utility PL/SQL packages and the corresponding API calls allowed for each of those.

The supported third-party REST providers are:

Cohere



- Generative AI
- Google Al
- Hugging Face
- OpenAl
- Vertex AI

The corresponding REST calls allowed for each operation are:

- DBMS VECTOR.UTL TO EMBEDDING and DBMS VECTOR.UTL TO EMBEDDINGS
- DBMS\_VECTOR\_CHAIN.UTL\_TO\_EMBEDDING and DBMS\_VECTOR\_CHAIN.UTL\_TO\_EMBEDDINGS
- DBMS\_VECTOR\_CHAIN.UTL\_TO\_SUMMARY
- DBMS\_VECTOR.UTL\_TO\_GENERATE\_TEXT
- DBMS VECTOR CHAIN.UTL TO GENERATE TEXT

## Validate JSON Input Parameters

You can optionally validate the structure of your JSON input to the <code>DBMS\_VECTOR.UTL</code> and <code>DBMS\_VECTOR\_CHAIN.UTL</code> 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 <code>DBMS\_JSON\_SCHEMA</code> package to test whether the input data to be specified in the <code>PARAMS</code> 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:

```
"string" } },
                   : {"type" : "boolean" } },
      "extended"
     "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" },
       "table name"
       "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" : {
                    : {"type" : "boolean" },
       "plaintext"
       "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:

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);
```

 To read a detailed validation report of errors, use the PL/SQL procedure DBMS JSON SCHEMA.validate report. 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);
```

• If you use the procedure (not function) <code>is\_valid</code>, then you have access to the validation errors report as an <code>OUT</code> parameter. If you use the function <code>is\_valid</code>, then you do not have access to such a report. Instead of using the function <code>is\_valid</code>, you can use the PL/SQL function <code>DBMS\_JSON\_SCHEMA.validate\_report</code> in a SQL query to validate and return the same full validation information that the reporting <code>OUT</code> parameter of the procedure <code>is\_valid</code> provides, as a JSON type instance.

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

#### **Related Topics**

- DBMS\_VECTOR
- DBMS VECTOR CHAIN
- DBMS\_JSON\_SCHEMA

## **Vector Generation Examples**

Review these examples to see how you can generate vectors within and outside the database.

- Generate Embeddings: SQL and PL/SQL Examples
   In these examples, you can see how to generate one or more vector embeddings from text strings and PDF documents.
- Perform Text Processing: PL/SQL Examples
   In these examples, you can see how to use some of the text processing features enabled by Oracle Text.
- Perform Chunking: SQL and PL/SQL Examples
   In these examples, you can see how to extract chunks from text strings and PDF documents.
- Generate Text for a Prompt: PL/SQL Example
   In this example, you can see how to generate text for a given prompt by accessing third-party text generation models.

## Generate Embeddings: SQL and PL/SQL Examples

In these examples, you can see how to generate one or more vector embeddings from text strings and PDF documents.

- Convert Text String to Embedding
  - You can vectorize text strings like this for chatbots or information-retrieval applications, where you want to directly convert a user's input text to a query vector and then run it against vector index for a fast similarity search.
- Convert File to Text to Chunks to Embeddings

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.



Convert File to Embeddings

You can directly extract vector embeddings from a PDF document, using a single-step statement.

Generate and Use Embeddings for End-to-End Search

In this example, you first generate embeddings from textual content by using an ONNX model, and then populate and query a vector index. At query time, you also vectorize the query criteria on the fly.

## Convert Text String to Embedding

You can vectorize text strings like this for chatbots or information-retrieval applications, where you want to directly convert a user's input text to a query vector and then run it against vector index for a fast similarity search.

You can perform a text-to-embedding transformation using the UTL\_TO\_EMBEDDING PL/SQL API (note the singular "embedding") or the VECTOR\_EMBEDDING SQL function. Both UTL TO EMBEDDING and VECTOR EMBEDDING directly return a VECTOR type (not an array).

#### Determine which API to use:

 If you want to access a third-party embedding model, then you can use UTL\_TO\_EMBEDDING from either the DBMS VECTOR OF DBMS VECTOR CHAIN package.

This scenario uses the DBMS VECTOR.UTL TO EMBEDDING API.

 If you are using an ONNX format embedding model, then you can use both VECTOR EMBEDDING and UTL TO EMBEDDING.

To generate a vector embedding with "hello" as the input:

- 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, to a pluggable database (PDB) within your multitenant container database (CDB):

```
conn sys/password@CDB_PDB as sysdba
```

```
CREATE TABLESPACE tbs1

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

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, create credential to docuser;
```



c. Connect to Oracle Database as the test user and alter the environment settings for your session:

```
CONN docuser/password@CDB_PDB

SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 10000
SET LONG 10000
```

d. Set the HTTP proxy server, if configured:

```
EXEC UTL_HTTP.SET_PROXY('proxy-hostname>:cproxy-port>');
```

e. 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.

- 2. If you are using a third-party embedding model and need to make a REST call, set up your credentials for the REST provider and then call <code>UTL\_TO\_EMBEDDING</code>.
  - Using Cohere, Google Al, Hugging Face, OpenAl, and Vertex Al:
    - 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',
                        => json(jo.to string));
       params
   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;
```

Replace the provider, credential\_name, url, and model values. Optionally, you can specify additional REST provider parameters.

```
Cohere example:
```

```
"provider": "cohere",
  "credential name": "COHERE CRED",
  "url": "https://api.cohere.example.com/embed",
  "model": "embed-model",
  "input type": "search query"
Google AI example:
  "provider": "googleai",
  "credential name": "GOOGLEAI CRED",
  "url": "https://googleapis.example.com/models/",
  "model": "embed-model"
Hugging Face example:
  "provider": "huggingface",
  "credential name": "HF CRED",
  "url": "https://api.huggingface.example.com/",
  "model": "embed-model",
  "wait for model": "true"
OpenAI example:
  "provider": "openai",
 "credential name": "OPENAI CRED",
  "url": "https://api.openai.example.com/embeddings",
  "model": "embed-model"
}
Vertex AI example:
  "provider": "vertexai",
  "credential name": "VERTEXAI CRED",
  "url": "https://googleapis.example.com/models/",
  "model": "embed-model"
}
```



#### Using Generative Al:

 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  ${\tt UTL}\ {\tt to}\ {\tt EMBEDDING}\ {\tt call}.$ 

#### Note:

The generated private key may appear as:

```
----BEGIN RSA PRIVATE KEY-----
<private key string>
----END RSA PRIVATE KEY-----
```

You pass the rivate 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_output.put_line(jo.to_string);
  dbms vector.create credential(
   credential_name => 'OCI_CRED',
                    => json(jo.to_string));
    params
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
   bababab1111222aba11ab');
     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
     dbms output.put line(jo.to string);
     dbms vector.create credential (
       credential name => 'OCI CRED',
                         => json(jo.to string));
       parameters
   end;
b. Call DBMS VECTOR.UTL TO EMBEDDING:
   -- select example
   var params clob;
   exec :params := '
     "provider": "ocigenai",
     "credential name": "OCI CRED",
     "url": "<REST endpoint URL for embedding service>",
     "model": "<REST provider embedding model name>"
   }';
   select dbms vector.utl to embedding('hello', json(:params)) from
   dual;
   -- PL/SQL example
   declare
     input clob;
     params clob;
     v vector;
     input := 'hello;
     params := '
     "provider": "ocigenai",
     "credential name": "OCI CRED",
     "url": "<REST endpoint URL for embedding service>",
     "model": "<REST provider 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;
/
```

Replace the url and model values. Optionally, you can specify additional REST provider-specific parameters.

For example:

```
{
  "provider": "ocigenai",
  "credential_name": "OCI_CRED",
  "url": "https://generativeai.oci.example.com/embedText",
  "model": "embed-modelname",
  "batch_size": 10
}
```

- 3. If you are using a declared embedding model, then call either VECTOR\_EMBEDDING or UTL TO EMBEDDING.
  - a. Load your ONNX model into Oracle Database.

For detailed information on how to perform this step, see Import ONNX Models and Generate Embeddings.

Here, doc\_model specifies the name under which the imported model is stored in Oracle Database.

- b. Call VECTOR EMBEDDING or UTL TO EMBEDDING:
  - 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;
```

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
```

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
- VECTOR\_EMBEDDING

## Convert File to Text to Chunks to Embeddings

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.

Using a set of functions from the <code>DBMS\_VECTOR\_CHAIN</code> package, you first convert a PDF file to text (<code>UTL\_TO\_TEXT</code>), split the text into chunks (<code>UTL\_TO\_CHUNKS</code>), and then create vector embeddings on each chunk (<code>UTL\_TO\_EMBEDDINGS</code>).

To generate embeddings using a declared embedding model, through step-by-step transformation chains:

- Start SQL\*Plus and connect to Oracle Database as a local test user:
  - Log in to SQL\*Plus as the sys user, connecting as sysdba, to a pluggable database (PDB) within your multitenant container database (CDB):

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

conn sys/password@CDB PDB as sysdba

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. Create a local directory on your server (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 to Oracle Database as the test user and alter the environment settings for your session:

```
conn docuser/password@CDB_PDB;

SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 10000
SET LONG 10000
```

#### 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. Run 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)
   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. Run UTL TO CHUNKS to chunk the text document:
      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
\n\n\n\nData
\n\n\nRaw Data\n\n\nMetadata\n\n\n\nSee Also:\n\n\n\nOracle
Database Data
Warehousing Guide to learn about transformation
\n\n\nmechanisms\n\n\nOvervie
w of Extraction, Transformation, and Loading (ETL) \n\n\nThe process
of extrac
ting data from source systems and bringing it into the warehouse is
\n \n \n \n
only called ETL: extraction, transformation, and loading. ETL refers to
a broad
process \n\n\nrather than three well-defined steps.\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
can occur
during this \n\n\
directly accesse
s a remote target through a \n\n\ngateway can concatenate two columns
as part
of the \n\n\n\
ETL tool. However, Oracle Database provides a rich set of
\n\n\ncapabilities
usable by ETL tools and customized ETL solutions. ETL capabilities
provided by \
tablespaces"}
3728 rows selected.
```

Notice the extra spaces and newline characters ( $\n$ ) in the chunked output. You can set the normalize chunking parameter 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.** Supply the following JSON parameters to use normalization and some of the custom chunking specifications:

```
"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
parallel execution. Oracle Database Data Warehousing Guide to learn
recommended"}
{"chunk id":2537,"chunk offset":1373160,"chunk length":701,"chunk data":
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. Drop and load your ONNX 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"]}}'));
```

Replace my\_embedding\_model.onnx with an ONNX export of your embedding model. Here, doc\_model specifies the name under which the imported 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.

b. Declare embedding parameters:

```
var embed_params clob;
exec :embed params := '{"provider":"database", "model":"doc model"}';
```

c. Run UTL\_TO\_EMBEDDINGS to generate a set of vector embeddings corresponding to the chunks:

```
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
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
0.0442310236, 0.0139067639, 0.054659389, -0.29653421, -0.0988782048, 0.079434923
0758788213,0.0152856084,-0.0260562375,0.0652966872,-0.0782724097,-0.0226081
.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,
82886E-002,3.41602638E-002,3.18607911E-002,-3.07250433E-002,-3.60006578E-00
26940455E-002,-5.13980947E-002,-9.18597169E-003,-2.40122043E-002,2.15246622
E - 0.02
,-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-0031"}
{"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
to function as a multitenant container database (CDB). XStream, 20-38\nZero
Loss Recovery Appliance See Recovery Appliance zone maps, An application
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.
157589E-002,-1.89648587E-002,-9.02655348E-002,1.97774544E-002,-9.39233322E-
5.06882742E-002,2.0078931E-002,-1.28898735E-003,-4.10606936E-002,2.09831214
,-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,-
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]"}
```

#### 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 Package
- DBMS\_VECTOR\_CHAIN Package

# Convert File to Embeddings

You can directly extract vector embeddings from a PDF document, using a single-step statement.

Perform a file-to-text-to-chunks-to-embeddings transformation (using a declared embedding model), by calling a set of <code>DBMS\_VECTOR\_CHAIN.UTL</code> functions in a single <code>CREATE TABLE</code> statement.

This statement creates a relational table (doc\_chunks) from unstructured text chunks and the corresponding vector embeddings:

```
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, where you embed two Oracle Database Documentation books in the doc\_chunks table and perform similarity searches using vector indexes.

# Generate and Use Embeddings for End-to-End Search

In this example, you first generate embeddings from textual content by using an ONNX model, 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 described 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 using a declared embedding model:

- Start SQL\*Plus and connect to Oracle Database as a local test user:
  - Log in to SQL\*Plus as the sys user, connecting as sysdba, to a pluggable database (PDB) within your multitenant container database (CDB):

```
conn sys/password@CDB_PDB as sysdba

CREATE TABLESPACE tbs1

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

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. Create a local directory on your server (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 to Oracle Database as the test user and alter the environment settings for your session:

```
conn docuser/password@CDB_PDB;

SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 10000
SET LONG 10000
```

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 ONNX 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"]}}')
):
```

Replace my\_embedding\_model.onnx with an ONNX export of your embedding model. Here, doc\_model specifies the name under which the imported 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.

4. Create a relational table (doc\_chunks) to store unstructured data chunks and associated vector embeddings, by using doc model.

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 <code>BLANKLINE</code>, <code>NEWLINE</code>, <code>SPACE</code>, <code>NONE</code> 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 1 418 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 <u>for you, your workgroup, and your enterprise.</u>

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 subscribe through Universal Credits. You can subscribe to Professional Edition or Enterprise Edition.

- 3 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 the Oracle Cloud Infrastructure.
- 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 <u>centers. It closely resembles a tradition al network, with</u>

5 2 329 474 centers. It closely resembles a traditional network, with firewall rules and specific types of communication gate ways that you can choose to us e. A VCN resides in a single O racle Cloud Infrastructure region and covers one or more CID R blocks
(IPv4 and IPv6, if enable

(IPv4 and IPv6, if enable d). See Allowed VCN Size and A ddress Ranges. The terms virtu al cloud network, VCN, and cloud network are used interchangeably in this documentation. For more information, see VCNs and Subnets.

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;</pre>

# 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-002,-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 gueries 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
```

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 embedding(doc model using 'banking, money' as data),
     MANHATTAN) results
 FROM doc chunks order by results;
      RESULTS
   ID
 ____
    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 Package

# Perform Text Processing: PL/SQL Examples

In these examples, you can see how to use some of the text processing features enabled by Oracle Text.

- Convert Text String to Summary
   You can extract a concise summary from large and complex documents.
- Create and Use Custom Vocabulary
   You can see how to create and use your own vocabulary of tokens when chunking data.
- Create and Use Custom Language Data
   You can create and use your own language-specific conditions (such as common abbreviations) when chunking data.

# Convert Text String to Summary

You can extract a concise summary from large and complex documents.

You can see how to access third-party text summarization models to perform a text-to-summary transformation, by using the DBMS VECTOR CHAIN.UTL TO SUMMARY API.

- 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, to a pluggable database (PDB) within your multitenant container database (CDB):

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

conn sys/password@CDB PDB as sysdba

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, create credential to docuser;
```



c. Connect to Oracle Database as the test user and alter the environment settings for your session:

```
SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 10000
SET LONG 10000
```

CONN docuser/password@CDB PDB

d. Set the HTTP proxy server if configured:

```
EXEC UTL HTTP.SET PROXY('proxy-hostname>:cproxy-port>');
```

e. 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.

- Set up your credentials for the REST provider that you want to access and then call UTL\_TO\_SUMMARY:
  - Using Cohere, Google Al, Hugging Face, OpenAl, and Vertex Al:
    - 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',
                         => json(jo.to string));
       params
   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
       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
       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
   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;
/
Replace the provider, credential name, url, and model values. Optionally, you
can specify additional REST provider parameters.
Cohere example:
  "provider": "cohere",
  "credential name": "COHERE CRED",
  "url": "https://api.cohere.example.com/summarize",
  "model": "summarize-model",
  "length": "medium",
  "format": "paragraph",
  "temperature": 1.0
Google AI example:
  "provider": "googleai",
  "credential name": "GOOGLEAI CRED",
  "url": "https://googleapis.example.com/models/",
  "model": "summarize-model",
  "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": "summarize-model",
  "wait for model": "true"
OpenAI example:
  "provider": "openai",
  "credential name": "OPENAI CRED",
```

```
"url": "https://api.openai.example.com",
  "model": "summarize-model",
  "max tokens": 256,
  "temperature": 1.0
Vertex AI example:
  "provider": "vertexai",
  "credential_name": "VERTEXAI_CRED",
  "url": "https://googleapis.example.com/models/",
  "model": "summarize-model",
  "generation config": {
    "temperature": 0.9,
    "topP": 1,
    "candidateCount": 1,
    "maxOutputTokens": 256
  }
}
```

#### Using Generative Al:

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  ${\tt UTL}\ {\tt TO}\ {\tt SUMMARY}$  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 output.put line(jo.to string);
  dbms vector chain.create credential(
   credential name => 'OCI CRED',
                    => json(jo.to string));
   params
end;
```

Replace all the authentication parameter values.

#### For example:

```
declare
   jo json_object_t;
begin
   jo := json_object_t();

jo.put('user_ocid','ocidl.user.ocl..aabbalbbaa1112233aabbaabb1111222
aa1111bb');

jo.put('tenancy_ocid','ocidl.tenancy.ocl..aaaaalbbbb1112233aaaabbaa1
111222aaa111a');

jo.put('compartment_ocid','ocidl.compartment.ocl..ababalabab1112233a
bababab1111222aba11ab');
   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 output.put line(jo.to string);
     dbms vector chain.create credential(
       credential_name => 'OCI CRED',
                         => json(jo.to string));
       params
   end;
b. Run DBMS VECTOR CHAIN.UTL TO SUMMARY:
   -- select example
   var params clob;
   exec :params := '
     "provider": "ocigenai",
     "credential name": "OCI CRED",
     "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
       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
   vou
       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
    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": "<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;
```

Replace the url and model values. Optionally, you can specify additional REST provider-specific parameters.

## For example:

```
{
  "provider": "ocigenai",
    "credential_name": "OCI_CRED",
  "url": "https://generativeai.oci.example.com/summarizeText",
    "model": "summarize.modelname",
    "length": "MEDIUM",
    "format": "PARAGRAPH",
    "temperature": 1.0
}
```

#### The generated summary appears as follows:

```
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.
```

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\_SUMMARY

# Create and Use Custom Vocabulary

You can see how to create and use your own vocabulary of tokens when chunking data.

Here, you use the chunker helper function <code>CREATE\_VOCABULARY</code> from the <code>DBMS\_VECTOR\_CHAIN</code> package to load custom vocabulary. This vocabulary file contains a list of tokens, recognized by your model's tokenizer.

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, to a pluggable database (PDB) within your multitenant container database (CDB):

```
conn sys/password@CDB_PDB as sysdba

CREATE TABLESPACE tbs1

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

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. Create a local directory on your server (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  ${\tt VEC\_DUMP}$  directory.

For example, if using WordPiece tokenization, you can download and transfer the vocab.txt vocabulary file for "bert-base-uncased".

e. Connect to Oracle Database as the test user and alter the environment settings for your session:

```
conn docuser/password@CDB_PDB;

SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 10000
SET LONG 10000
```

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. Run DBMS\_VECTOR\_CHAIN.CREATE\_VOCABULARY to create a vocabulary (doc\_vocab).

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
- VECTOR\_CHUNKS
- UTL\_TO\_CHUNKS

# Create and Use Custom Language Data

You can create and use your own language-specific conditions (such as common abbreviations) when chunking data.

Here, you use the chunker helper function <code>CREATE\_LANG\_DATA</code> from the <code>DBMS\_VECTOR\_CHAIN</code> package to load the data file for Simplified Chinese. This data file contains abbreviation tokens for your chosen language.

- 1. Start SQL\*Plus and connect to Oracle Database as a local test user:
  - Log in to SQL\*Plus as the sys user, connecting as sysdba, to a pluggable database (PDB) within your multitenant container database (CDB):

```
conn sys/password@CDB_PDB as sysdba

CREATE TABLESPACE tbs1

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

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. Create a local directory on your server (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 to Oracle Database as the test user and alter the environment settings for your session:

```
conn docuser/password@CDB_PDB;

SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 10000
SET LONG 10000
```

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. Run DBMS VECTOR CHAIN.CREATE LANG DATA to create language data (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
- VECTOR\_CHUNKS
- UTL\_TO\_CHUNKS

# Perform Chunking: SQL and PL/SQL Examples

In these examples, you can see how to extract chunks from text strings and PDF documents.

- 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.
- 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 while chunking your data.

# 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 <code>VECTOR\_CHUNKS</code> SQL function or the <code>UTL\_TO\_CHUNKS()</code> PL/SQL function from the <code>DBMS VECTOR CHAIN</code> package.

- Start SQL\*Plus and connect to Oracle Database as a local test user.
  - Log in to SQL\*Plus as the sys user, connecting as sysdba, to a pluggable database (PDB) within your multitenant container database (CDB):

```
conn sys/password@CDB_PDB as sysdba

CREATE TABLESPACE tbs1

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



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 to Oracle Database as the test user and alter the environment settings for your session:

```
CONN docuser/password@CDB_PDB

SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 10000
SET LONG 10000
```

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 guery 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:

```
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 <code>UTL\_TO\_CHUNKS</code> function from the <code>DBMS\_VECTOR\_CHAIN</code> package, run:

This returns a set of three chunks, which are split by words recursively using blank lines, new lines, and spaces:

```
DOC POS SIZ TXT

1 108 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.

4. To further clean up the chunking results, supply JSON parameters for UTL TO CHUNKS:

#### This returns chunk texts in JSON format:

```
COLUMN VALUE
______
{"chunk id":1,"chunk offset":1,"chunk length":108,"chunk data":"Oracle AI
Search stores and indexes vector embeddings for fast retrieval and
similarity s
earch."}
{"chunk id":2, "chunk offset":109, "chunk length":234, "chunk data": "About
Oracle A
I Vector Search\nVector Indexes are a new classification of specialized
that are designed for Artificial Intelligence (AI) workloads that allow
you to q
uery data based on semantics, rather than keywords."}
{"chunk id":3,"chunk offset":343,"chunk length":204,"chunk data":"Why Use
AI Vector Search?\nThe biggest benefit of Oracle AI Vector Search is that
tic search on unstructured data can be combined with relational search on
ss data in one single system."}
```

#### 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

### **Related Topics**

- VECTOR CHUNKS
- UTL TO 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 while 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 Al 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.

- 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 3-1 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 3-2 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)

С;

#### 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 3-3 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 Al 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 embeddings	104 for fast retr	Oracle AI Vector Search stores and indexes vector ieval 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 3-4 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.

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.

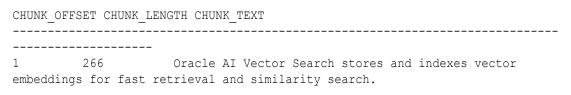
#### Syntax:

```
SELECT C.*

FROM documentation_tab D, VECTOR_CHUNKS(D.text BY words MAX 40 OVERLAP 0

SPLIT BY none LANGUAGE american NORMALIZE none) C;
```

#### Output:



About Oracle AI Vector Search



 $\begin{tabular}{ll} Vector Indexes are a new classification of specialized indexes that are designed for Artificial \\ Intelligence (AI) workloads \end{tabular}$ 

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 3-5 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.

# 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 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 3-6 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 Al 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 3-7 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 3-4. This is because vocabulary tokens include pieces of words, so the chunk text is generally smaller than simple word splitting.

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 none LANGUAGE american NORMALIZE none) C;

#### Output:

1 157 Oracle AI Vector Search stores and indexes vector embeddings for fast retrieval and similarity search.

eddings 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

ORACLE<sup>®</sup>

data can be combined with relational search on business data in one single system.

## Example 3-8 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 3-5.

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 3-5, 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
        138
                      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
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 3-9 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 3-6.

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.

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 recursively LANGUAGE american NORMALIZE

none) C;
```

#### Output:

CHUNK OFFSET CHUNK LENGTH CHUNK TEXT Oracle AI Vector Search stores and indexes vector embeddings for fast retrieval and similarity search. 109 30 About Oracle AI Vector Search 143 Vector Indexes are a new classification of specialized indexes that are designed for Artificial Intelligence (AI) workloads that allow you to query 291 data based on semantics, rather than keywords. 48 343 33 Why Use Oracle AI Vector Search? The biggest benefit of Oracle AI Vector Search is 162 that semantic search on unstructured data can be combined with relational

search on business data in one single 539 8 system.

#### Example 3-10 BY words MAX 40 OVERLAP 5 SPLIT BY none

This example is the similar to Example 3-4, except an overlap of 5 is used.

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 <u>Intelligence (AI) workloads</u> that allow you to query data based on semantics, rather than keywords.

Why Use Oracle Al 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 3-11 BY words MAX 40 OVERLAP 5 SPLIT BY newline

This example is the similar to Example 3-5, except an overlap of 5 is used. 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.

# 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 newline LANGUAGE american NORMALIZE none)
C;
```

#### Output:

```
CHUNK OFFSET CHUNK LENGTH CHUNK TEXT
         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 3-12 BY words MAX 40 OVERLAP 5 SPLIT BY recursively

This example is the similar to Example 3-6, except an overlap of 5 is used. 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 Al 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, <u>rather than keywords.</u>

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
_____
                    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  ${\it combined with relational search on business data in } \\$  one single system.

#### Example 3-13 BY chars MAX 200 OVERLAP 0 SPLIT BY none NORMALIZE none

This example is the same as Example 3-1, to contrast with Example 3-13 with normalization.

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
        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 0
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 3-14 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.

Oracle Al 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

whitespace) C;
```

#### Output:

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 3-15 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 Al 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
                         Oracle AI Vector Search stores and indexes vector
            102
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 3-16 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。<mark>客户服务号是标识您所在组织的唯一参考号。</mark>使用 My Oracle Support 门户向您的概要信息中添加一个客户服务<mark>号。</mark>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, <code>documentation\_tab</code> is a <code>CLOB</code> inserted with the following <code>ChineseDoc.txt</code> document:

#### 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);
```

#### 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
- UTL\_TO\_CHUNKS

## Generate Text for a Prompt: PL/SQL Example

In this example, you can see how to generate text for a given prompt by accessing third-party text generation models.

A prompt can be an input string (such as a question that you ask an LLM or a command), and can include results from a search.

You can use the described functions either from the DBMS\_VECTOR or DBMS\_VECTOR\_CHAIN package, depending on your use case.

To generate text using "What is Oracle Text?" as the prompt:

1. Start SQL\*Plus and connect to Oracle Database as a local test user.

 Log in to SQL\*Plus as the sys user, connecting as sysdba, to a pluggable database (PDB) within your multitenant container database (CDB):

```
conn sys/password@CDB_PDB as sysdba

CREATE TABLESPACE tbs1

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

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, create credential to docuser;
```

c. Connect to Oracle Database as the test user and alter the environment settings for your session:

```
SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 10000
SET LONG 10000
```

CONN docuser/password@CDB PDB

d. Set the HTTP proxy server, if configured:

```
EXEC UTL HTTP.SET PROXY('proxy-hostname>:cproxy-port>');
```

e. 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.

- 2. Set up your credentials for the REST provider that you want to access and then call UTL TO GENERATE TEXT:
  - 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;
/
```

Replace the provider, credential\_name, url, and model values. Optionally, you can specify additional REST provider parameters.

#### Cohere example:

```
{
  "provider": "Cohere",
  "credential_name": "COHERE_CRED",
  "url": "https://api.cohere.example.com/generateText",
  "model": "generate-text-model"
}
```

#### Google AI example:

```
{
  "provider": "googleai",
  "credential_name": "GOOGLEAI_CRED",
  "url": "https://googleapis.example.com/models/",
  "model": "generate-text-model"
}
```



#### Hugging Face example:

```
"provider": "huggingface",
  "credential_name": "HF_CRED",
  "url": "https://api.huggingface.example.com/models/",
  "model": "generate-text-model",
  "wait_for_model": "true"
}
```

#### OpenAI example:

```
"provider": "openai",
  "credential_name": "OPENAI_CRED",
  "url": "https://api.openai.example.com",
  "model": "generate-text-model",
  "max_tokens": 60,
  "temperature": 1.0
}
```

#### Vertex AI example:

```
{
  "provider": "vertexai",
  "credential_name":"VERTEXAI_CRED",
  "url": "https://googleapis.example.com/models/",
  "model": "generate-text-model",
  "generation_config": {
    "temperature": 0.9,
    "topP": 1,
    "candidateCount": 1,
    "maxOutputTokens": 256
  }
}
```

#### 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  $\verb"UTL_TO_GENERATE_TEXT"$  call.

#### Note:

The generated private key may appear as:

```
----BEGIN RSA PRIVATE KEY-----
<private key string>
----END RSA PRIVATE KEY-----
```

You pass the rivate 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 output.put line(jo.to string);
  dbms vector chain.create credential(
   credential name => 'OCI CRED',
                    => json(jo.to string));
   params
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..aaaaalbbbb1112233aaabbaa1
11222aaa111a');

jo.put('compartment_ocid', 'ocid1.compartment.oc1..ababalabab1112233a
bababab1111222aba11ab');
   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 output.put line(jo.to string);
     dbms vector chain.create credential(
       credential name => 'OCI CRED',
                        => json(jo.to_string));
       params
   end;
b. Call utl to generate text:
   -- select example
   var params clob;
   exec :params := '
     "provider": "ocigenai",
     "credential name": "OCI CRED",
     "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": "ocigenai",
     "credential name": "OCI CRED",
     "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;
```

Replace the  ${\tt url}$  and  ${\tt model}$  values. Optionally, you can specify additional REST provider-specific parameters.

#### For example:

```
{
  "provider": "OCIGenAI",
  "credential_name": "GENAI_CRED",
  "url": "https://generativeai.oci.example.com/generateText",
  "model": "generate-text-model",
  "inferenceRequest": {
    "maxTokens": 300,
    "temperature": 1
  }
}
```

#### The generated text appears as follows:

```
BMS_VECTOR_CHAIN.UTL_TO_GENERATE_TEXT(:INPUT, JSON(:PARAMS))
-----
Oracle Text is a powerful tool that enhances Oracle Database with integrated
```

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  $\ensuremath{\mathsf{und}}$ 

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  $% \left\{ 1\right\} =\left\{ 1\right\} =\left\{$ 

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**

- DBMS\_VECTOR Package
- DBMS\_VECTOR\_CHAIN Package

4

# Store Vector Embeddings

You store the resulting 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.



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 no practical upper limit.

The possible dimension formats are:

- INT8 (8-bit integers)
- FLOAT32 (32-bit IEEE floating-point numbers)
- FLOAT64 (64-bit IEEE floating-point numbers)

Oracle Database automatically casts the values as needed.

The following table guides you through the possible declaration format for a VECTOR data type:

Possible Declaration Format	Explanation
VECTOR	Vectors can have an arbitrary number of dimensions and formats.
VECTOR(*, *)	Vectors can have an arbitrary number of dimensions and formats. VECTOR and VECTOR (*, *) are equivalent.
VECTOR(number_of_dimensions, *)	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)	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, *) and VECTOR(number_of_dimensions) are equivalent.
VECTOR(*, dimension_element_format)	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).

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]).

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(1, *),
    v5 VECTOR(*, FLOAT32),
    v6 VECTOR(*, *),
    v7 VECTOR
  );
Table created.
DESC my_vect_tab;
                             Null?
                                      Type
V1
                                      VECTOR(3 , FLOAT32)
V2
                                      VECTOR(2 , FLOAT64)
```

V3	VECTOR(1 ,	INT8)
V4	VECTOR(1 ,	*)
V5	VECTOR(* ,	FLOAT32)
V6	VECTOR(* ,	*)
V7	VECTOR(* ,	*)

You currently cannot define **VECTOR** columns in/as:

- External Tables
- 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
- MSSM tablespace (only SYS user can create VECTORs as Basicfiles in MSSM tablespace)
- 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</li>

### 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
'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
'7E-002,-7.5879097E-002,-2.72218436E-002,7.01764375E-002,-1.32512336E-003,3.14
728'11
'022E-002,-1.39147148E-001,-7.52705336E-002,2.62449421E-002,1.91645715E-002,4.
'73137E-002,5.83701171E-002,-3.26474942E-002,2.0509012E-002,-3.81141738E-003,-
'8656182E-002,-1.95893757E-002,-2.56917924E-002,-6.57705888E-002,-4.39117625E-
002'11
',-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
307'11
'76E-002,5.13443872E-002,-1.40985977E-002,-1.87066793E-002,-1.11725368E-002,9.
'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,-
'6833269E-002,3.13266069E-002,1.2377765E-002,7.64856935E-002,-3.77447419E-002,
'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-
002'11
',-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
'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
'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,
'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-
002'11
',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
'937E-002,-4.99385223E-003,3.37120108E-002,8.99377018E-002,1.09540671E-001,5.8
'102E-002,1.71627291E-002,-3.26152593E-002,8.36912021E-002,5.05600758E-002,-9.
737'11
'63615E-002,-1.40264994E-002,-2.07926836E-002,-4.20163684E-002,-5.97197041E-00
'.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
'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
899'11
'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
'67188111E-003,-1.10706768E-003,-6.34015873E-002,-4.80389707E-002,6.84534572E-
003'11
',-1.1547043E-002,-3.44865513E-003,1.18979132E-002,-4.31232266E-002,-5.9022788
'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
082'11
'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,-
'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
'02,-7.02963024E-002,-3.91333885E-002,1.76661201E-002,-5.09723537E-002,2.37749
107'11
'E-002,-1.83419678E-002,-1.2693027E-002,-1.14232123E-001,-6.68751821E-002,7.52
'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,
'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-
002'11
',-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
'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
'-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
057'11
'17E-002,4.06873561E-002,4.74951901E-002,-1.22502812E-033,-4.6441108E-002,3.74
079'11
'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
'5.00682592E-002,5.90037219E-002,-2.89178211E-002,8.01460445E-003,-3.41945067E
'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 generate vectors by calling services outside the database or generate vectors directly from within the database after you have imported pretrained embedding models.

#### See Also:

- Import Pretrained Models in ONNX Format for Vector Generation Within the Database
- Convert Text String to Embedding

## 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 fivedimension vector space based on the count of important words appearing in each document:

Table 4-1 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)
)
```

After you have created the two files <code>galaxies\_vec.csv</code> and <code>galaxies\_vec.ctl</code>, 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
           Messier 31 is a barred spiral galaxy in the Andromeda ...
[0,2.0E+000,2.0E+000,0,0]
           Messier 33 is a spiral galaxy in the Triangulum ...
[0,0,1.0E+000,0,0]
          Messier 58 is an intermediate barred spiral galaxy ...
[1.0E+000,1.0E+000,1.0E+000,0,0]
           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]
           Messier 91 is a barred spiral galaxy in the Coma ...
  6 M91
[0,1.0E+000,1.0E+000,0,0]
           Messier 49 is a giant elliptical galaxy in the Virgo ...
[0,0,0,1.0E+000,1.0E+000]
           Messier 60 is an elliptical galaxy in the Virgo ...
  8 M60
[0,0,0,0,1.0E+000]
  9 NGC1073 NGC 1073 is a barred spiral galaxy in Cetus \dots
[0,1.0E+000,1.0E+000,0,0]
9 rows selected.
SOL>
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.
- O Rows not loaded due to data errors.
- O 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 <code>embedding char (32000)</code> vectors. For very large vectors, you can use the <code>LOBFILE</code> 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 4-2** 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 fivecs 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":



SQL\*Loader supports loading <code>VECTOR</code> columns from character data and binary floating point array <code>fvec</code> files. The format for <code>fvec</code> 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 <code>galaxies\_vec.csv</code> in this case does not have the vector data. Instead, the vector data will be read from the secondary <code>LOBFILE</code> in the <code>/u01/data/vector</code> directory (<code>/u01/data/vector/embedding.fvecs</code>), which contains the same information in <code>float32</code> floating point binary numbers, but is in <code>fvecs</code> 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>impdp <username>/<password>@<Database-instance-TNS-alias> dumpfile=<dumpfile-</pre>
```

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



5

## Create Vector Indexes

You may want to create vector indexes on your vector embeddings and use these indexes for running similarity searches over huge vector spaces.

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.

- 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
   Learn how vector indexing methods make vector searches faster and how to enable vector indexes creation.

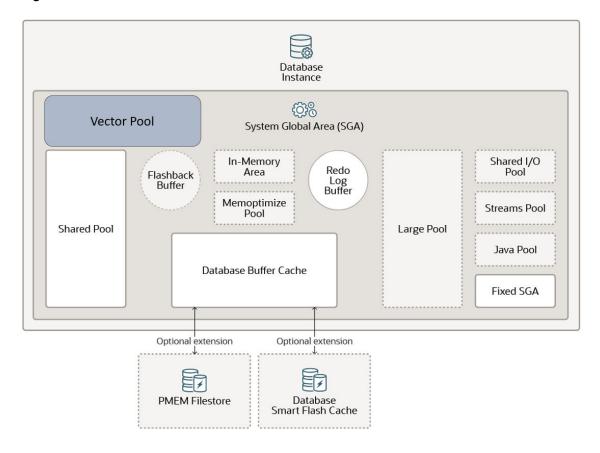
### Size the Vector Pool

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 Flat File (IVF) index creation as well as DML operations on base tables with IVF indexes.

Enabling a Vector Pool is illustrated in the following diagram:

Figure 5-1 Vector Pool



To size the Vector Pool, 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 <code>VECTOR\_MEMORY\_SIZE</code> at the PDB level if you are using an SPFILE:



```
SOL> show user
USER is "SYS"
SQL> show parameter vector_memory_size
        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
                          VALUE
                TYPE
-----
vector memory size big integer 1G
SQL>
```

For more information about changing initialization parameter values, see Managing Initialization Parameters Using a Server Parameter File

V\$VECTOR MEMORY POOL, is the view you can query to monitor the Vector Pool.

#### See Also:

Vector Memory Pool Views

## Manage the Different Categories of Vector Indexes

Learn how vector indexing methods make vector searches faster and how to enable vector indexes creation.

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 which 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.

### 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.

- Understand Hierarchical Navigable Small World Indexes
   The default type of index created for an In-Memory Neighbor Graph vector index is
   Hierarchical Navigable Small World (HNSW). Use these examples to understand how to
   create HNSW indexes for vector approximate similarity searches.
- Hierarchical Navigable Small World Index Syntax and Parameter Syntax for Hierarchical Navigable Small World Index

### Understand Hierarchical Navigable Small World Indexes

The default type of index created for an In-Memory Neighbor Graph vector index is Hierarchical Navigable Small World (HNSW). 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:

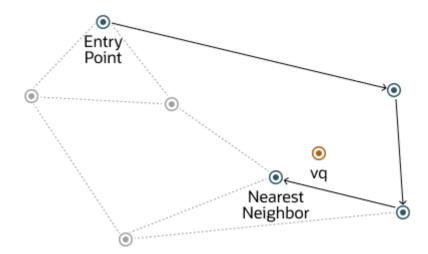


Figure 5-2 Navigable Small World Graph

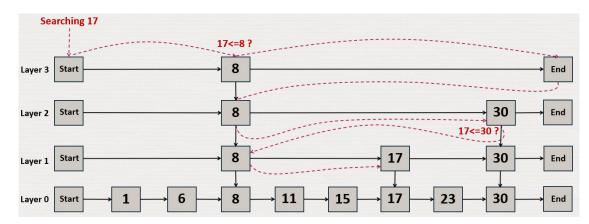
The described method demonstrates robust performance up to a certain scale of vector insertion into the graph. Beyond this threshold, the Hierarchical Navigable Small World (HNSW) approach enhances the NSW model by introducing a multi-layered hierarchy, akin to the structure observed in Probabilistic 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 5-3 Probability Skip List Structure

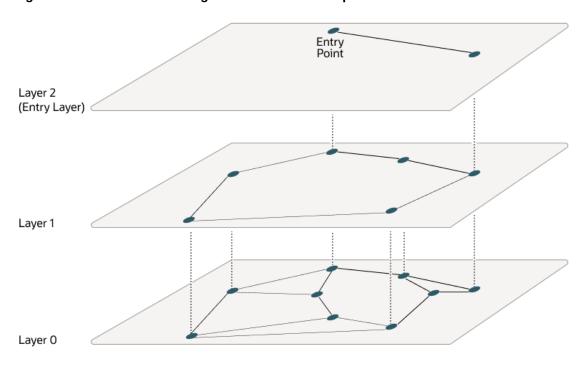


The Probability Skip List structure uses multiple layers of linked lists where the above layers are skipping more numbers than the lower ones. In this example, you are trying to search for number 17. You start with the top layer and jump to the next element until you either find 17, reach the end of 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 5-4 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:

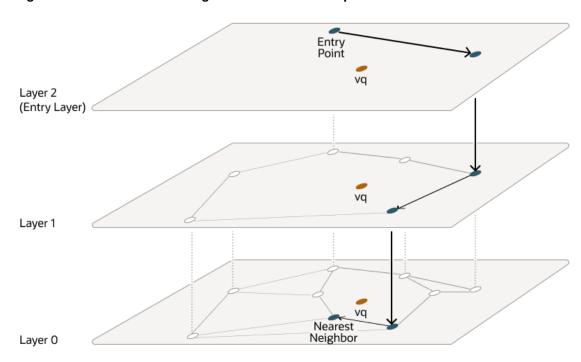


Figure 5-5 Hierarchical Navigable Small World Graphs Search

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.

### Hierarchical Navigable Small World Index Syntax and Parameter

Syntax for Hierarchical Navigable Small World Index

#### **Syntax**

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 which balance high search quality with reasonable speed.

#### Understand Inverted File Flat Vector Indexes

The default type of index created for a Neighbor Partition vector index is Inverted File Flat (IVF) vector index. The IVF 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 Index Syntax and Parameter Syntax for Inverted File Flat Index

### Understand Inverted File Flat Vector Indexes

The default type of index created for a Neighbor Partition vector index is Inverted File Flat (IVF) vector index. The IVF 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 5-6 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  ${\tt k}$  partitions. This division is conceptually illustrated as expanding circles from the centroids that stop growing as they meet, forming distinct partitions.

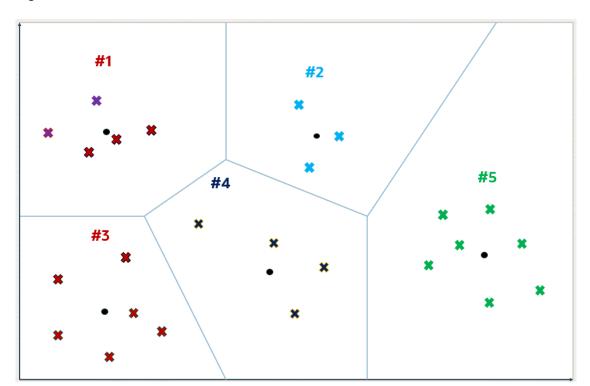


Figure 5-7 Inverted File Flat Index

Except for those on the periphery, each vector falls within a specific partition associated with a centroid.



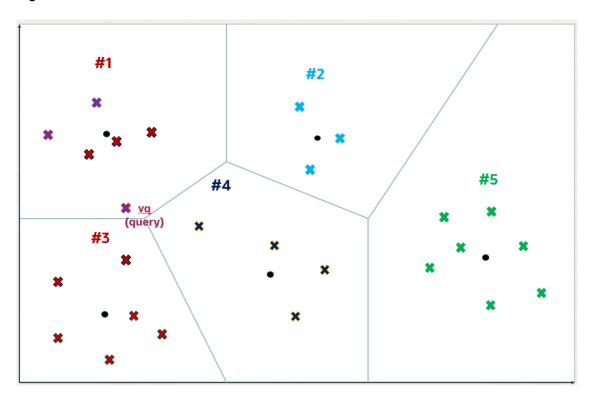


Figure 5-8 Inverted File Flat Index

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 <code>NEIGHBOR</code> Partition <code>PROBES</code> 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,  ${\tt i}$  is set to 2 and the two identified partitions are partitions number 1 and 3.

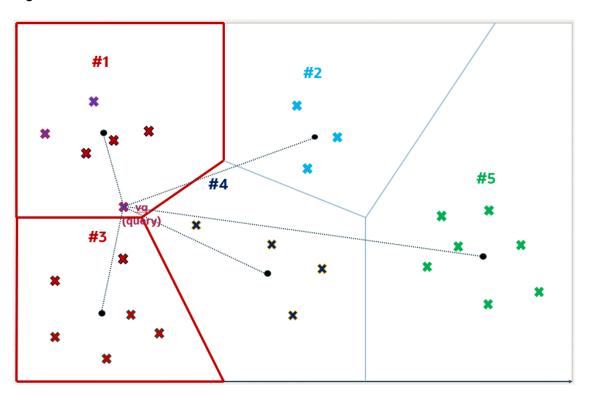


Figure 5-9 Inverted File Flat Index

Once the  ${\tt i}$  partitions are determined, they are fully scanned to identify, in this example, the top 5 nearest vectors. This number 5 can be different from  ${\tt k}$  and you specify this number in your query. The five nearest vectors to  ${\tt vq}$  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.



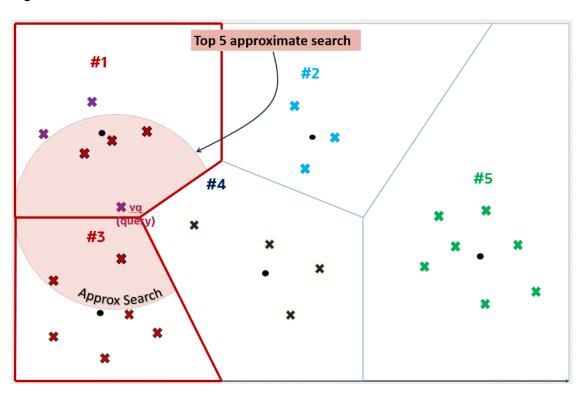


Figure 5-10 Inverted File Flat Index

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.

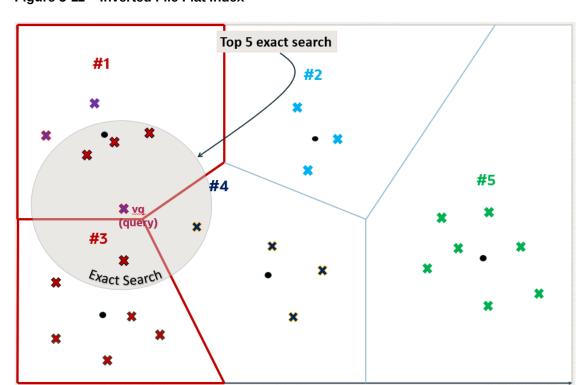


Figure 5-11 Inverted File Flat Index

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.

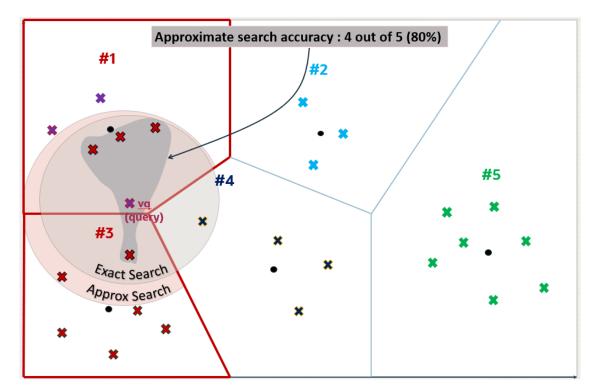


Figure 5-12 Inverted File Flat Index

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 Index Syntax and Parameter

Syntax for Inverted File Flat Index

#### **Syntax**

#### **Parameter**

NEIGHBOR PARTITIONS determines the target number of centroid partitions that are created by the index.

SAMPLE\_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</li>



- SAMPLE 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**

```
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);
```

# **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
- Materialized views
- Non-vector columns (VARCHAR, NUMBER, and so on.)
- Function-based vector index
- Sharded tables



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.
- You can only create one type of vector index per vector column.
- Oracle recommends that you allocate larger, temporary tablespaces for proper Inverted File Flat (IVF) vector index creation with big vector spaces and vector sizes. In such cases, the system internally makes extensive use of temporary space.
- If you are running your Oracle Database on a RAC environment, you cannot create HNSW indexes. On a non-RAC single instance environment, you can create both HNSW and IVF indexes. Using HNSW is the preferred type if it fits entirely into memory.
- On a RAC environment you can set up a vector pool on each instance for the best performance of IVF indexes.

#### **Use Index Guidelines**

For the Oracle Database Optimizer to consider a vector index, you must ensure to verify 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 COSINE, 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 rebuild using the REBUILD INDEX function also provided by the DBMS\_VECTOR



package. See *Oracle Database PL/SQL Packages and Types Reference* for more information about DBMS VECTOR and its subprograms.

• After an HNSW vector index is created, DML operations are not permitted on the indexed table. DML operations are supported when using IVF indexes.

### Note:

When the database instance is restarted, the existing HNSW indexes must be rebuilt as these structures are inmemory-only structures. There are three options for you to recreate an HNSW vector index after a restart:

- Manually drop the index and create it again. This requires you to remember and provide the original index creation parameters.
- DBMS\_VECTOR.REBUILD\_INDEX: This procedure uses the DBMS\_METADATA.GET\_DDL function to get all of the index creation parameters. You may override the parameters by passing their values into the procedure. For more information about the DBMS\_VECTOR.REBUILD procedure, see *Oracle Database PL/SQL Packages and Types Reference*.
- Automatic reload: The VECTOR\_INDEX\_NEIGHBOR\_GRAPH\_RELOAD initialization
  parameter is set to OFF by default. If set to RESTART, the system automatically
  loads the HNSW indexes one by one through a background task. It will use the
  same parallelism and the index creation parameters that you had specified
  during the original HNSW index creation.

### 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 <code>DBMS\_VECTOR.INDEX\_ACCURACY\_QUERY</code> procedure. This procedure provides an accuracy report for a top-K index search for a specific query vector and a specific target accuracy.

Here is a usage example of this procedure using our galaxies scenario:

```
declare
  q_v VECTOR;
```



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 <code>DBMS\_VECTOR.INDEX\_ACCURACY\_REPORT</code> 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?	Туре
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

				. <b>_</b> _	
	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	0.0	79	
70	71	0.7	80	3	
79	0.1	87	0.0	89 3	
70	81	71	90	78	
70	91	/ 1	100	4	
67	91	79.5	100	88	
0 /		, , , ,		0.0	

Each row in the output represents a bucket of 10 target accuracy values: 1-10, 11-20, 21-30,  $\dots$ , 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 <code>NUM\_VECTORS</code> would be increased by one in row 6 (bucket values between 61 and 70) of the results of the select statement on the <code>DBA\_VECTOR\_INDEX\_ACCURACY\_REPORT</code> view for your task. <code>NUM\_VECTORS</code> 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 <code>VECTOR\_QUERY\_CAPTURE</code> is used to enable and disable capture of query vectors. The parameter value is set to <code>ON</code> by default. You can turn off this background functionality by setting <code>VECTOR\_QUERY\_CAPTURE</code> to <code>OFF</code>. When <code>VECTOR\_QUERY\_CAPTURE</code> is <code>ON</code>, the database captures some of the query vectors through sampling. The captured query vectors are retrained for a week and then purged automatically.



6

# Use SQL Functions for Vector Operations

There are a number of SQL functions and operators that you can use with vectors in Oracle Al Vector Search.

#### Vector Distance Functions

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. Distances determine similarity or dissimilarity between vectors.

#### Other Basic Vector Functions

Other basic vector operations for Oracle Al Vector Search involve creating, converting, and describing vectors.

Oracle Al Vector Search SQL Functions

Oracle AI Vector utilities provide the VECTOR\_CHUNKS and VECTOR\_EMBEDDING SQL functions for chunking and embedding data, respectively.

# **Vector Distance Functions**

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. 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 Operand to the VECTOR\_DISTANCE Function
 VECTOR\_DISTANCE() is the main function that allows you to calculate distances between two vectors.

#### Shorthand Operators for Distances

Oracle AI Vector Search provides shorthand operators that you can use for distance functions.

# **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 Similarity

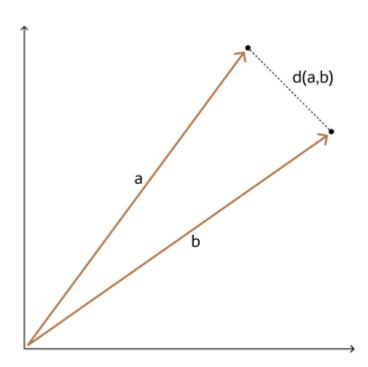
The Hamming distance between two vectors represents the number of dimensions where they differ.

### 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).

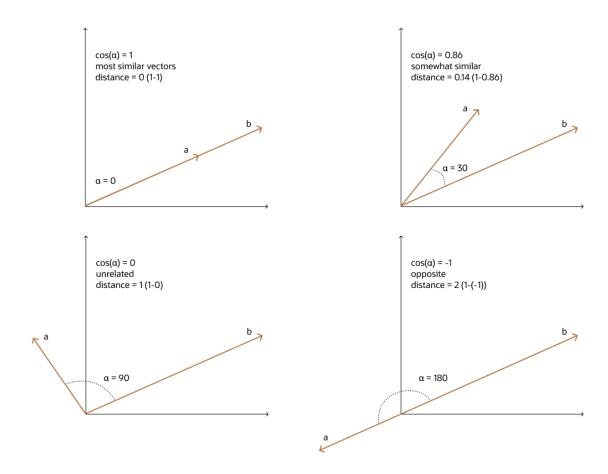




## **Cosine Similarity**

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.

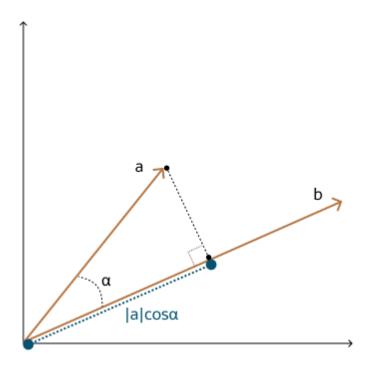
As illustrated in the following diagram, you project one vector on the other and multiply the resulting vector sizes.

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.



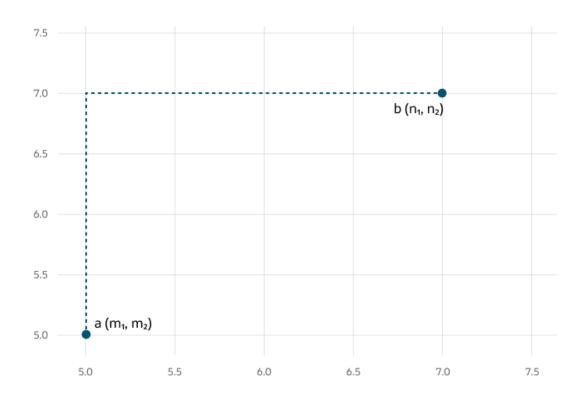
### Manhattan Distance

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.





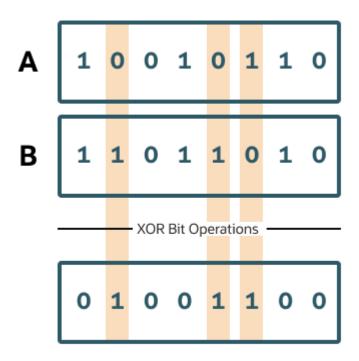
## **Hamming Similarity**

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, you need to compare the position of each bit in the sequence. You can do this by using <code>exclusive</code> or (also called the XOR bit operation), which outputs 1 if the bits in the sequence do not match, and 0 otherwise. 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 = 3

# Vector Distance Operand to the VECTOR DISTANCE Function

 ${\tt VECTOR\_DISTANCE}\,() \ \ is the \ main \ function \ that \ allows \ you \ to \ calculate \ distances \ between \ two \ vectors.$ 

The <code>VECTOR\_DISTANCE()</code> function takes two vectors as parameters. You can optionally specify a distance metric to calculate the distance desirably. If you do not specify a distance metric, then the default distance metric is the <code>COSINE</code> metric.

You can optionally use the following shorthand functions too:  $\verb"l1_DISTANCE", \verb"l2_DISTANCE", "l2_DISTANCE", "l2_DIST$ 

All of these functions return the vectors' distance as a BINARY DOUBLE.

When using the <code>VECTOR\_DISTANCE()</code> function to perform a similarity search, note the following caveats:

- If a similarity search query does not specify a distance metric in the VECTOR\_DISTANCE()
  function, then the default COSINE metric will be used for both exact and approximate (index-based) searches.
- If a similarity search query does specify a distance metric in the VECTOR\_DISTANCE()
  function, then an exact search with that distance metric is used if it conflicts with the
  distance metric specified in a vector index. If the two distance metrics are the same, then
  that will be used for both exact and approximate (index-based) searches.



#### **Syntax**

For detailed information on the vector functions, see Vector Functions in *Oracle SQL Language Reference*.

#### **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.

# **Shorthand Operators for Distances**

Oracle Al Vector Search provides shorthand operators that you can use for distance functions.

#### **Purpose**

Oracle provides the following shorthand distance operators in lieu of their corresponding distance functions:

- <-> is the Euclidian distance operator: expr1 <-> expr2 is equivalent to
   L2 DISTANCE (expr1, expr2) or VECTOR DISTANCE (expr1, expr2, EUCLIDEAN)
- <=> is the cosine distance operator: expr1 <=> expr2 is equivalent to
   COSINE DISTANCE(expr1, expr2) or VECTOR DISTANCE(expr1, expr2, COSINE)
- <#> is the negative dot product operator: expr1 <#> expr2 is equivalent to -1\*INNER PRODUCT (expr1, expr2) or VECTOR DISTANCE (expr1, expr2, DOT)

#### **Syntax**

```
<expr1> <-> <expr2>
<expr1> <#> <expr2>
<expr1> <=> <expr2>
```

#### **Parameters**

expr1 and expr2 must evaluate to vectors and have the same number of dimensions. These operations return NULL if either expr1 or expr2 is NULL.

#### **Example**

- '[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)



# Other Basic Vector Functions

Other basic vector operations for Oracle Al 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.

#### VECTOR NORM

The VECTOR\_NORM() function returns the Euclidean norm of a vector (SQRT(SUM((xi-yi)2))) in the format of BINARY DOUBLE.

### VECTOR\_DIMENSION\_COUNT

The <code>VECTOR\_DIMENSION\_COUNT()</code> function returns the number of dimensions of a vector in the format of an Oracle number.

#### VECTOR\_DIMENSION\_FORMAT

The VECTOR\_DIMENSION\_FORMAT() returns the storage format of the vector. It returns a VARCHAR2, which can be one of the following values: 'INT8', 'FLOAT32', or 'FLOAT64'.

### **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() 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.

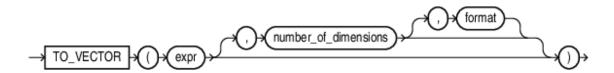
For detailed information, see the following in the Oracle SQL Language Reference:

- TO VECTOR
- VECTOR
- TO VECTOR
- VECTOR
- Parameters
- Examples

### TO VECTOR

#### **Syntax**

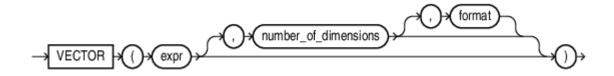
TO VECTOR(expr [ , number of dimensions [ , format ] ] )



### **VECTOR**

#### **Syntax**

```
VECTOR ( expr [ , number of dimensions [ , format ] ] )
```



### **Parameters**

- expr must evaluate to either (a) a string that represents a vector (accepted input data types for the string are character types, JSON, CLOB, and BLOB) or (b) a vector. The valid string representation of a vector is 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]'. If it evaluates to a vector (the input's data type is VECTOR), the function serves the purpose of converting it to the specified or default format. The expression can be NULL, in which case the result is NULL as well. If a BLOB is used as the input parameter, it must represent the vector binary bytes.
- 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, or \*. This is the target internal storage format of the vector. If \* is used, the format will be FLOAT32. Note that this behavior is a bit different than declaring a vector column. If you declare a column of type VECTOR(3, \*), then all inserted vectors will NOT have their storage format modified (stored as is).

## Examples

```
SELECT TO_VECTOR('[34.6, 77.8]');
SELECT TO_VECTOR('[34.6, 77.8]', 2, FLOAT32);

SELECT TO_VECTOR('[34.6, 77.8]', 2, FLOAT32) FROM dual;

TO_VECTOR('[34.6, 77.8]', 2, FLOAT32)

[3.45999985E+001, 7.78000031E+001]

1 row selected.

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]
```

1 row selected.

### 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]');
```

 For applications using pre-Oracle Client 23ai libraries connected to Oracle Database 23ai, use the TO\_VECTOR() SQL function to insert vector data. For example:

```
INSERT INTO vecTab VALUES(TO VECTOR('[1.1, 2.9, 3.14]'));
```

### **Vector Serializers**

FROM VECTOR() and VECTOR SERIALIZE() are synonymous serializers of vectors.

Both functions take a vector as input and return a string of type VARCHAR2 or CLOB as output. They optionally take a RETURNING clause to specify the returned data type. If VARCHAR2 is specified without size, the returned value size is 32767. There is no support to convert to CHAR, NCHAR, and NVARCHAR2.

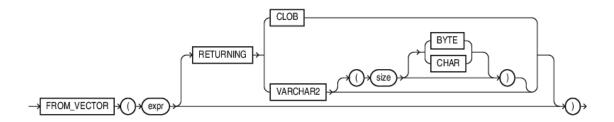
For detailed information, see the following in the Oracle SQL Language Reference:

- FROM\_VECTOR
- VECTOR\_SERIALIZE
- FROM\_VECTOR
- VECTOR\_SERIALIZE
- Parameters
- Examples

# FROM\_VECTOR

### **Syntax**

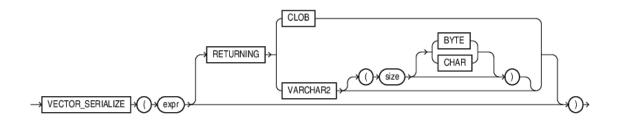
```
FROM_VECTOR ( expr [ RETURNING ( CLOB | VARCHAR2 [ ( size [BYTE | CHAR] ) ] ) ] )
```



# VECTOR\_SERIALIZE

#### **Syntax**

```
VECTOR_SERIALIZE ( expr [ RETURNING ( CLOB | VARCHAR2 [ ( size [BYTE |
CHAR] ) ] ) ] )
```



### **Parameters**

expr must have a vector type. The function returns <code>NULL</code> if <code>expr</code> is <code>NULL</code>.

# **Examples**

```
SELECT FROM_VECTOR(TO_VECTOR('[1, 2, 3]'));

SELECT FROM_VECTOR(TO_VECTOR('[1.1, 2.2, 3.3]', 3, FLOAT32));

VECTOR_SERIALIZE(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));

VECTOR_SERIALIZE(VECTOR('[...]', 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 );
```

Note:

1 row selected.

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 pre-Oracle Client 23ai libraries connected to Oracle Database 23ai, use the FROM\_VECTOR() SQL function to fetch vector data, as shown by the following example:

SELECT FROM VECTOR(dataVec) FROM vecTab;

# **VECTOR NORM**

The  $VECTOR\_NORM()$  function returns the Euclidean norm of a vector (SQRT(SUM((xi-yi)2))) in the format of BINARY DOUBLE.

The  $VECTOR\_NORM()$  function returns the Euclidean norm of a vector (SQRT(SUM((xi-yi)2))) in the format of BINARY DOUBLE.

For detailed information, see VECTOR\_NORM of Oracle SQL Language Reference.

# VECTOR\_DIMENSION\_COUNT

The <code>VECTOR\_DIMENSION\_COUNT()</code> function returns the number of dimensions of a vector in the format of an Oracle number.

The <code>VECTOR\_DIMENSION\_COUNT()</code> function returns the number of dimensions of a vector in the format of an Oracle number.

For detailed information, see VECTOR\_DIMENSION\_COUNT and VECTOR\_DIMS of *Oracle SQL Language Reference*.

# VECTOR\_DIMENSION\_FORMAT

The <code>VECTOR\_DIMENSION\_FORMAT()</code> returns the storage format of the vector. It returns a <code>VARCHAR2</code>, which can be one of the following values: <code>'INT8'</code>, <code>'FLOAT32'</code>, or <code>'FLOAT64'</code>.

The VECTOR\_DIMENSION\_FORMAT() returns the storage format of the vector. It returns a VARCHAR2, which can be one of the following values: 'INT8', 'FLOAT32', or 'FLOAT64'.

For detailed information, see VECTOR\_DIMENSION\_FORMAT of *Oracle SQL Language Reference*.

# Oracle Al Vector Search SQL Functions

Oracle AI Vector utilities provide the <code>VECTOR\_CHUNKS</code> and <code>VECTOR\_EMBEDDING</code> SQL functions for chunking and embedding data, respectively.

For detailed information, see the following sections in *Oracle Database SQL Language Reference*:

### VECTOR\_CHUNKS

The VECTOR\_CHUNKS function enables you 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.

### VECTOR\_EMBEDDING

The VECTOR\_EMBEDDING function enables you to generate embedding for different data types according to an embedding ONNX model.



7

# Query Data with Similarity Searches

Use Oracle AI Vector Search native SQL operations from your development environment to combine similarity with relational searches.

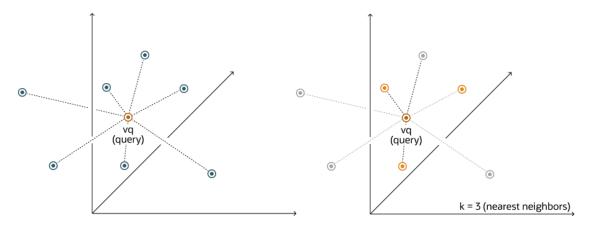
- Perform Exact Similarity Search
  - A similarity search looks for the relative order of vectors compared to a query vector. Naturally, the comparison is done using a particular distance metric but what is important is the result set of your top closest vectors, not the distance between them.
- Perform 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 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 7-1 Exact Search



With an exact search, you compare the query vector  $\mathbf{v}\mathbf{q}$  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  $\mathbf{k}$  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 that EXACT is optional.



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  ${\tt ROW\ LIMITING\ CLAUSE}$ 

# 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

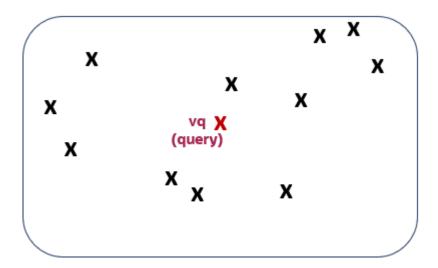
When search quality is your high priority and search speed is less important, Exact Similarity search is a good option. Search speed can be irrelevant for smaller vector spaces, or when you perform searches with high performance servers. However, ML algorithms often perform similarity searches on vector 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.

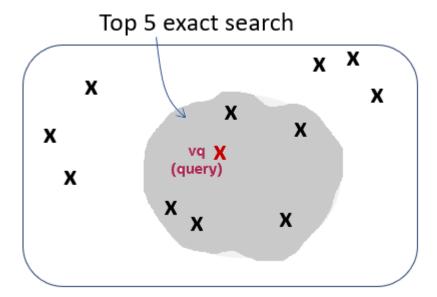
#### Target accuracy example

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.

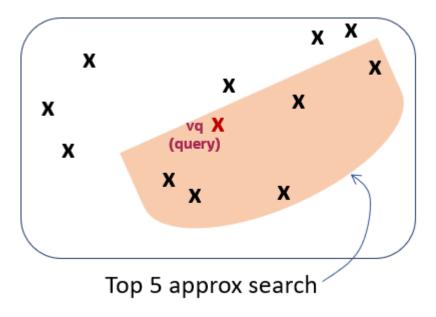




Running a top-5 exact similarity search in that context would return the five vectors shown on the second diagram:

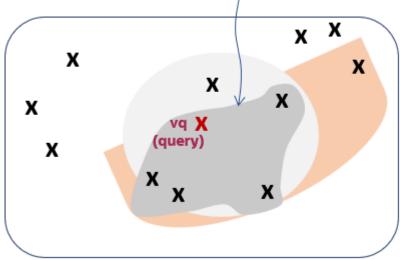


Depending on how your vector index was constructed, running a top-5 approximate similarity search in that context could return the five vectors shown on the third diagram. This is because the index creation process is using heuristics. So searching through the vector index may lead to different results 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.

## **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  * 1   2  * 3   4   5		SELECT STATEMENT

### 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 much less 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.

The following plan shows one of the four possibilities just described. Specifically, it shows the execution plan when in-filtering is chosen with join back.



#### 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   Ope	ration	 	Name	
* 1   CO	ECT STATEMENT UNT STOPKEY IEW SORT ORDER BY STOPKEY TABLE ACCESS BY INDEX ROWID VECTOR INDEX HNSW SCAN IN-FILTER VIEW TABLE ACCESS BY USER ROWID		DOC_CHUNKS DOCS_HNSW_IDX3 VW_HIJ_B919B0A0 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 7-1 HNSW Options

Operation	Options	Object_name
TABLE ACCESS	FULL	VECTOR\$ <index- name&gt;\$HNSW_ROW_VID_MAP</index- 
TABLE ACCESS	STORAGE FULL	VECTOR\$ <index- name&gt;\$HNSW_ROW_VID_MAP</index- 

For the HNSW index itself, which is an In-Memory object, the plan uses a VECTOR INDEX operation. The object name <code>GALAXIES\_HSNW\_INX</code> is provided as an example of the user-specified HNSW index name:

Operation	Options	Object_name
VECTOR INDEX	HNSW SCAN	GALAXIES_HSNW_INX
VECTOR INDEX	HNSW SCAN PRE-FILTER	GALAXIES_HSNW_INX
VECTOR INDEX	HNSW SCAN IN-FILTER	GALAXIES_HSNW_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.

#### 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 execution 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. The base table is scanned to look for all the rows that pass the WHERE clause
  filter.
- Plan line id 4: Once both sets are identified, the rows are joined to extract only the relevant ones
- Plan line id 3: This step extracts the top-K rows.

```
| Id | Operation
Name
  0 | SELECT STATEMENT
|* 1 | COUNT STOPKEY
   2 | VIEW
1* 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
          TABLE ACCESS FULL
|* 11 |
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
```



#### 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 7-2 Centroids and Centroid Partition Table Options

Operation	Options	Object_name
TABLE ACCESS	FULL	VECTOR\$ <vector-index- name&gt;\$<id>\$IVF_FLAT_CENTRO IDS</id></vector-index- 
		VECTOR\$DOCS_IVF_IDX2\$ <id>\$ IVF_FLAT_CENTROID_PARTITIO NS</id>
TABLE ACCESS	STORAGE FULL	VECTOR\$ <vector-index- name&gt;\$<id>\$IVF_FLAT_CENTRO IDS</id></vector-index- 
		VECTOR\$DOCS_IVF_IDX2\$ <id>\$ IVF_FLAT_CENTROID_PARTITIO NS</id>

## 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);

See Also:

Oracle Database SQL Language Reference for the full syntax of the  ${\tt ROW\_LIMITING\_CLAUSE}$ 

### 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);
```

```
See Also:
```

Oracle Database SQL Language Reference for the full syntax of the  ${\tt ROW\ LIMITING\ CLAUSE}$ 

# 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  ${\tt VECTOR\_DISTANCE}\,()$  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  ${\tt ROW\_LIMITING\_CLAUSE}$ 



8

# Work with 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. By communicating with LLMs through the implementation of RAG, the knowledge of LLMs is increased with business data found through AI Vector Search.

#### Compliment LLMs with Oracle AI Vector Search

Using Retrieval Augmented Generation (RAG) can mitigate the inaccuracies and hallucinations faced when using Large Language Models (LLMs). Oracle AI Vector Search enables RAG through the use of popular frameworks and PL/SQL APIs.

#### 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.

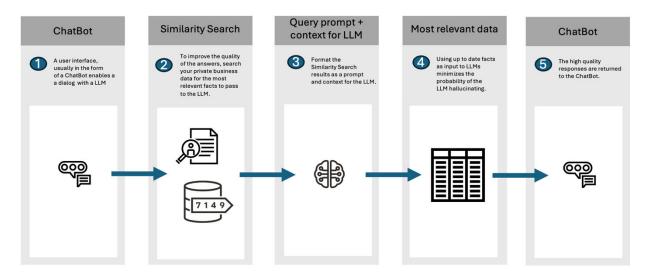
## Compliment LLMs with Oracle AI Vector Search

Using Retrieval Augmented Generation (RAG) can mitigate the inaccuracies and hallucinations faced when using Large Language Models (LLMs). Oracle AI Vector Search enables RAG through the use of popular frameworks and PL/SQL APIs.

The primary problem with Large Language Models (LLMs) like 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 make up answers (hallucinate).

Figure 8-1 Example RAG Workflow



**Retrieval Augmented Generation (RAG)** 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. 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 Al Vector Search enables RAG within Oracle Database using the <code>DBMS\_VECTOR\_CHAIN PL/SQL</code> 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.

- 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.

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).



#### OCIGenAl 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_OUTPUT.PUT_LINE(jo.to_string);
  DBMS VECTOR CHAIN.CREATE CREDENTIAL (
   credential name => 'OCID CRED',
            => 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:



For information about loading the ONNX format model into the database as doc model, see Import ONNX Models and Generate Embeddings.

```
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 input), 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 GenAl call:

```
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;
```



9

# 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;
```

10

# **Vector Diagnostics**

Al Vector Search includes several views, statistics, and parameters that can be used to help understand how vector search is performing for your workload.

Oracle Al Vector Search Views

These are a set of data dictionary views related to Oracle Al Vector Search.

Oracle Al Vector Search Statistics

These are a set of statistics related to Oracle AI Vector Search.

Oracle Al Vector Search Parameters

This is a set of parameters related to Oracle Al Vector Search.

## Oracle Al Vector Search Views

These are a set of data dictionary views related to Oracle AI Vector Search.

Vector Utilities-Related Views

These views display language-specific data (abbreviation token details) and vocabulary data related to the Oracle Al Vector Search SQL and PL/SQL utilities.

Vector Memory Pool Views

Review the various vector memory pool views.

Vector Index Views

Review the vector index views.

### **Vector Utilities-Related 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.

USER VECTOR ABBREV TOKENS

The  ${\tt USER\_VECTOR\_ABBREV\_TOKENS}$  view displays a list of abbreviation tokens from all languages loaded by 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**

Oracle Database PL/SQL Packages and Types Reference

### 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)

## USER\_VECTOR\_ABBREV\_TOKENS

The <code>USER\_VECTOR\_ABBREV\_TOKENS</code> 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\_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  ${\tt USER\_VECTOR\_VOCAB}$  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 <code>USER\_VECTOR\_VOCAB\_TOKENS</code> 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)
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



Column Name	Data Type	Description
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
POPULATE_STATUS	VARCHAR2 (26)	Status of the vector memory store —whether it is being populated, is done populating etc.



Column Name	Data Type	Description	
CON_ID	NUMBER	The ID of the container to which the data pertains. Possible values are:	
		<ul> <li>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.</li> <li>1: This value is used for rows containing data that pertain to only the root.</li> <li>n: Where n is the applicable container ID for the rows containing data.</li> </ul>	

#### **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 Views**

Review the vector index views.

VECSYS.VECTOR\$INDEX

This dictionary table contains detailed information about vector indexes.

### **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 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 Flat File vector indexes.
IDX_SPARE1	NUMBER	
IDX_SPARE2	JSON	

#### **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>
```

## **Oracle AI Vector Search Statistics**

These are a set of statistics related to Oracle Al Vector Search.

- Oracle Al Vector Search Dictionary Statistics
   A set of dictionary statistics related to Oracle Al 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 Al 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 inmemory 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 inmemory 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 inmemory 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 Al Vector Search.

VECTOR\_MEMORY\_SIZE

```
Syntax: vector_memory_size = [ON | OFF] (default ON)
```

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). The possible 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 OFF)
```

The initialization parameter <code>VECTOR\_INDEX\_NEIGHBOR\_GRAPH\_RELOAD</code> automatically loads HNSW indexes one by one after instance restart through a background task. For more information about this parameter, see Guidelines for Using Vector Indexes.

VECTOR QUERY CAPTURE

```
Syntax: vector_query_capture = [ON | OFF] (default ON)
```

The initialization parameter <code>VECTOR\_QUERY\_CAPTURE</code> is used to enable and disable capture of query vectors. For more information about this parameter and about capturing query vectors, see <a href="Index Accuracy Report">Index Accuracy Report</a>.

11

# Vector Search PL/SQL APIs

Explore Oracle AI Vector Search APIs and reference materials.

Oracle Al Vector Search PL/SQL Packages
 These are a set of PL/SQL packages related to Oracle Al Vector Search.

# Oracle Al Vector Search PL/SQL Packages

These are a set of PL/SQL packages related to Oracle Al Vector Search.

For detailed information, see the following sections in *Oracle Database PL/SQL Packages and Types Reference*:

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 creating vector indexes.

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, using functionality that can be pipelined together for an end-to-end search.