216
DBMS_VECTOR

The DBMS_VECTOR package provides APIs to support common operations with Oracle AI Vector Search, such as extracting chunks or embeddings from user data, generating text for a given prompt, creating a vector index, or reporting on index accuracy.

These functions accept their respective input parameters in JSON format.

Related Topics

- *Oracle Database AI Vector Search User's Guide*

Summary of DBMS_VECTOR Subprograms

This table lists the DBMS_VECTOR subprograms and briefly describes them.

Table 216-1 ***DBMS_VECTOR Package Subprograms***

Subprogram	Description
ONNX Model Related Procedures:	
These procedures enable you to load an ONNX model into Oracle Database and drop the ONNX model.	
LOAD_ONNX_MODEL	Loads an ONNX model into the database
LOAD_ONNX_MODEL_CLOUD	Loads an ONNX model from object storage into the database
DROP_ONNX_MODEL Procedure	Drops the ONNX model
Chainable Utility (UTL) Functions:	
These functions are a set of modular and flexible functions within vector utility PL/SQL packages. You can chain these together to automate end-to-end data transformation and similarity search operations.	
UTL_TO_CHUNKS	Splits data into smaller pieces or chunks
UTL_TO_EMBEDDING and UTL_TO_EMBEDDINGS	Converts data to one or more vector embeddings
UTL_TO_GENERATE_TEXT	Generates text for a prompt (input string) or an image
Credential Helper Procedures:	
These procedures enable you to securely manage authentication credentials in the database. You require these credentials to enable access to third-party service providers for making REST calls.	
CREATE_CREDENTIAL	Creates a credential name
DROP_CREDENTIAL	Drops an existing credential name
Data Access Functions:	
These functions enable you to retrieve data, create index, and perform simple similarity search operations.	
CREATE_INDEX	Creates a vector index
REBUILD_INDEX	Rebuilds a vector index
GET_INDEX_STATUS	Describes the status of a vector index creation
ENABLE_CHECKPOINT	Enables the Checkpoint feature for a vector index user and index name

Table 216-1 (Cont.) *DBMS_VECTOR Package Subprograms*

Subprogram	Description
DISABLE_CHECKPOINT	Disables the Checkpoint feature for a vector index user and index name
INDEX_VECTOR_MEMORY_ADVISOR	Determines the vector memory size that is needed for a vector index
QUERY	Performs a similarity search query
RERANK	Reorders search results for more relevant output
Accuracy Reporting Function: These functions enable you to determine the accuracy of existing search indexes and to capture accuracy values achieved by approximate searches performed by past workloads.	
INDEX_ACCURACY_QUERY	Verifies the accuracy of a vector index
INDEX_ACCURACY_REPORT	Captures accuracy values achieved by approximate searches

> **Note:**
>
> DBMS_VECTOR is a lightweight package that does not support text processing or summarization operations. Therefore, the UTL_TO_TEXT and UTL_TO_SUMMARY chainable utility functions and all the chunker helper procedures are available only in the advanced DBMS_VECTOR_CHAIN package.

CREATE_CREDENTIAL

Use the DBMS_VECTOR.CREATE_CREDENTIAL credential helper procedure to create a credential name for storing user authentication details in Oracle Database.

Purpose

To securely manage authentication credentials in the database. You require these credentials to enable access during REST API calls to your chosen third-party service provider, such as Cohere, Google AI, Hugging Face, Oracle Cloud Infrastructure (OCI) Generative AI, OpenAI, or Vertex AI.

A credential name holds authentication parameters, such as user name, password, access token, private key, or fingerprint.

Note that if you are using Oracle Database as the service provider, then you do not need to create a credential.

WARNING:

Certain features of the database may allow you to access services offered separately by third-parties, for example, through the use of JSON specifications that facilitate your access to REST APIs.

Your use of these features is solely at your own risk, and you are solely responsible for complying with any terms and conditions related to use of any such third-party services. Notwithstanding any other terms and conditions related to the third-party services, your use of such database features constitutes your acceptance of that risk and express exclusion of Oracle's responsibility or liability for any damages resulting from such access.

Syntax

```
DBMS_VECTOR.CREATE_CREDENTIAL (
    CREDENTIAL_NAME     IN VARCHAR2,
    PARAMS              IN JSON DEFAULT NULL
);
```

CREDENTIAL_NAME

Specify a name of the credential that you want to create for holding authentication parameters.

PARAMS

Specify authentication parameters in JSON format, based on your chosen service provider.

Generative AI requires the following authentication parameters:

```
{
"user_ocid"       : "<user ocid>",
"tenancy_ocid"    : "<tenancy ocid>",
"compartment_ocid": "<compartment ocid>",
"private_key"     : "<private key>",
"fingerprint"     : "<fingerprint>"
}
```

Cohere, Google AI, Hugging Face, OpenAI, and Vertex AI require the following authentication parameter:

```
{ "access_token": "<access token>" }
```

Table 216-2 Parameter Details

Parameter	Description
`user_ocid`	Oracle Cloud Identifier (OCID) of the user, as listed on the User Details page in the OCI console.
`tenancy_ocid`	OCID of your tenancy, as listed on the Tenancy Details page in the OCI console.
`compartment_ocid`	OCID of your compartment, as listed on the Compartments information page in the OCI console.

Table 216-2 (Cont.) Parameter Details

Parameter	Description

`private_key`	OCI private key.
	Note: The generated private key may appear as:
	`-----BEGIN RSA PRIVATE KEY-----`
	`<private key string>`
	`-----END RSA PRIVATE KEY-----`
	You pass the *`<private key string>`* value (excluding the `BEGIN` and `END` lines), either as a single line or as multiple lines.

`fingerprint`	Fingerprint of the OCI profile key, as listed on the User Details page under API Keys in the OCI console.
`access_token`	Access token obtained from your third-party service provider.

Required Privilege

You need the `CREATE CREDENTIAL` privilege to call this API.

Examples

- For Generative AI:

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

jo.put('user_ocid','ocid1.user.oc1..aabbalbbaa1112233aabbaabb1111222aa1111b
b');

jo.put('tenancy_ocid','ocid1.tenancy.oc1..aaaaalbbbb1112233aaaabbaa1111222a
aa111a');

jo.put('compartment_ocid','ocid1.compartment.oc1..ababalabab1112233abababab
1111222aba11ab');
  jo.put('private_key','AAAaaaBBB11112222333...AAA111AAABBB222aaa1a/+');
  jo.put('fingerprint','01:1a:a1:aa:12:a1:12:1a:ab:12:01:ab:a1:12:ab:1a');
  dbms_vector.create_credential(
    credential_name   => 'OCI_CRED',
    params            => json(jo.to_string));
end;
/
```

- For Cohere:

```
declare
  jo json_object_t;
begin
  jo := json_object_t();
  jo.put('access_token', 'A1Aa0abA1AB1a1Abc123ab1A123ab123AbcA12a');
  dbms_vector.create_credential(
    credential_name   => 'COHERE_CRED',
    params            => json(jo.to_string));
end;
/
```

End-to-end examples:

To run end-to-end example scenarios using this procedure, see Use LLM-Powered APIs to Generate Summary and Text.

CREATE_INDEX

Use the `DBMS_VECTOR.CREATE_INDEX` procedure to create a vector index.

Purpose

To create a vector index such as Hierarchical Navigable Small World (HNSW) vector index or Inverted File Flat (IVF) vector index.

Syntax

```
DBMS_VECTOR.CREATE_INDEX (
    idx_name                    IN VARCHAR2,
    table_name                  IN VARCHAR2,
    idx_vector_col              IN VARCHAR2,
    idx_include_cols            IN VARCHAR2 DEFAULT NULL,
    idx_partitioning_scheme     IN VARCHAR2 default 'LOCAL',
    idx_organization            IN VARCHAR2,
    idx_distance_metric         IN VARCHAR2 DEFAULT COSINE,
    idx_accuracy                IN NUMBER DEFAULT 90,
    idx_parameters              IN CLOB,
    idx_parallel_creation       IN NUMBER DEFAULT 1
);
```

Parameters

Parameter	Description
`idx_name`	Name of the index to create.
`table_name`	Table on which to create the index.
`idx_vector_col`	Vector column on which to create the index.
`idx_include_cols`	A comma-separated list of column names to be covered by the index.
`idx_partitioning_scheme`	Partitioning scheme for IVF indexes: • `GLOBAL` • `LOCAL` IVF indexes support both global and local indexes on partitioned tables. By default, these indexes are globally partitioned by centroid. You can choose to create a local IVF index, which provides a one-to-one relationship between the base table partitions or subpartitions and the index partitions. For detailed information on these partitioning schemes, see Inverted File Flat Vector Indexes Partitioning Schemes.

Parameter	Description
`idx_organization`	Index organization: - `NEIGHBOR PARTITIONS` - `INMEMORY NEIGHBOR GRAPH` For detailed information on these organization types, see Manage the Different Categories of Vector Indexes.
`idx_distance_metric`	Distance metric or mathematical function used to compute the distance between vectors: - `COSINE` (default) - `MANHATTAN` - `HAMMING` - `JACCARD` - `DOT` - `EUCLIDEAN` - `L2_SQUARED` - `EUCLIDEAN_SQUARED` For detailed information on each of these metrics, see Vector Distance Functions and Operators.
`idx_accuracy`	Target accuracy at which the approximate search should be performed when running an approximate search query. As explained in Understand Approximate Similarity Search Using Vector Indexes, you can specify non-default target accuracy values either by specifying a percentage value or by specifying internal parameters values, depending on the index type you are using. - For an HNSW approximate search: In the case of an HNSW approximate search, you can specify a target accuracy percentage value to influence the number of candidates considered to probe the search. This is automatically calculated by the algorithm. A value of 100 will tend to impose a similar result as an exact search, although the system may still use the index and will not perform an exact search. The optimizer may choose to still use an index as it may be faster to do so given the predicates in the query. Instead of specifying a target accuracy percentage value, you can specify the `EFSEARCH` parameter to impose a certain maximum number of candidates to be considered while probing the index. The higher that number, the higher the accuracy. For detailed information, see Understand Hierarchical Navigable Small World Indexes. - For an IVF approximate search: In the case of an IVF approximate search, you can specify a target accuracy percentage value to influence the number of partitions used to probe the search. This is automatically calculated by the algorithm. A value of 100 will tend to impose an exact search, although the system may still use the index and will not perform an exact search. The optimizer may choose to still use an index as it may be faster to do so given the predicates in the query. Instead of specifying a target accuracy percentage value, you can specify the `NEIGHBOR PARTITION PROBES` parameter to impose a certain maximum number of partitions to be probed by the search. The higher that number, the higher the accuracy. For detailed information, see Understand Inverted File Flat Vector Indexes.

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td><code>idx_parameters</code></td><td>Type of vector index and associated parameters.
Specify the indexing parameters in JSON format:
<ul>
<li>For HNSW indexes:
<ul>
<li><code>type</code>: Type of vector index to create, that is, <code>HNSW</code></li>
<li><code>neighbors</code>: Maximum number of connections permitted per vector in the HNSW graph</li>
<li><code>efConstruction</code>: Maximum number of closest vector candidates considered at each step of the search during insertion</li>
</ul>
For example:
<pre>
{
    "type"   : "HNSW",
    "neighbors"   : 3,
    "efConstruction" : 4
}
</pre>
For detailed information on these parameters, see Hierarchical Navigable Small World Index Syntax and Parameters.</li>
<li>For IVF indexes:
<ul>
<li><code>type</code>: Type of vector index to create, that is, <code>IVF</code></li>
<li><code>partitions</code>: Neighbor partition or cluster in which you want to divide your vector space</li>
</ul>
For example:
<pre>
{
    "type"   : "IVF",
    "partitions" : 5
}
</pre>
For detailed information on these parameters, see Inverted File Flat Index Syntax and Parameters.</li>
</ul></td></tr>
<tr><td><code>idx_parallel_creation</code></td><td>Number of parallel threads used for index construction.</td></tr>
</table>

Examples

- Specify neighbors and efConstruction for HNSW indexes:

```
dbms_vector.create_index(
    'v_hnsw_01',
    'vpt01',
    'EMBEDDING',
     NULL,
     NULL,
    'INMEMORY NEIGHBOR GRAPH',
    'EUCLIDEAN',
     95,
    '{"type" : "HNSW", "neighbors" : 3, "efConstruction" : 4}');
```

- Specify the number of partitions for IVF indexes:

  ```
  dbms_vector.create_index(
      'V_IVF_01',
      'vpt01',
      'EMBEDDING',
       NULL,
       NULL,
      'NEIGHBOR PARTITIONS',
      'EUCLIDEAN',
       95,
      '{"type" : "IVF", "partitions" : 5}');
  ```

DISABLE_CHECKPOINT

Use the `DISABLE_CHECKPOINT` procedure to disable the Checkpoint feature for a given Hierarchical Navigable Small World (HNSW) index user and HNSW index name. This operation purges all older checkpoints for the HNSW index. It also disables the creation of future checkpoints as part of the HNSW graph refresh.

Syntax

```
DBMS_VECTOR.DISABLE_CHECKPOINT('INDEX_USER',['INDEX_NAME']);
```

INDEX_USER

Specify the user name of the HNSW vector index owner.

INDEX_NAME

Specify the name of the HNSW vector index for which you want to disable the Checkpoint feature.

The `INDEX_NAME` clause is optional. If you do not specify the index name, then this procedure disables the Checkpoint feature for all HNSW vector indexes under the given user.

Examples

- Using both the index name and index user:

  ```
  DBMS_VECTOR.DISABLE_CHECKPOINT('VECTOR_USER','VIDX1');
  ```

- Using only the index user:

  ```
  DBMS_VECTOR.DISABLE_CHECKPOINT('VECTOR_USER');
  ```

Related Topics

- *Oracle Database AI Vector Search User's Guide*
- ENABLE_CHECKPOINT
  Use the `ENABLE_CHECKPOINT` procedure to enable the Checkpoint feature for a given Hierarchical Navigable Small World (HNSW) index user and HNSW index name.

DROP_CREDENTIAL

Use the `DBMS_VECTOR.DROP_CREDENTIAL` credential helper procedure to drop an existing credential name from the data dictionary.

Syntax

```
DBMS_VECTOR.DROP_CREDENTIAL (
    CREDENTIAL_NAME      IN VARCHAR2
);
```

CREDENTIAL_NAME

Specify the credential name that you want to drop.

Examples

- For Generative AI:

  ```
  exec dbms_vector.drop_credential('OCI_CRED');
  ```

- For Cohere:

  ```
  exec dbms_vector.drop_credential('COHERE_CRED');
  ```

ENABLE_CHECKPOINT

Use the `ENABLE_CHECKPOINT` procedure to enable the Checkpoint feature for a given Hierarchical Navigable Small World (HNSW) index user and HNSW index name.

> **Note:**
>
> - This procedure only allows the index to create checkpoints. The checkpoint is created as part of the next HNSW graph refresh.
> - By default, HNSW checkpointing is enabled. If required, you can disable it using the `DBMS_VECTOR.DISABLE_CHECKPOINT` procedure.

Syntax

```
DBMS_VECTOR.ENABLE_CHECKPOINT('INDEX_USER',['INDEX_NAME']);
```

INDEX_USER

Specify the user name of the HNSW vector index owner.

INDEX_NAME

Specify the name of the HNSW vector index for which you want to enable the Checkpoint feature.

The INDEX_NAME clause is optional. If you do not specify the index name, then this procedure enables the Checkpoint feature for all HNSW vector indexes under the given user.

Examples

- Using both the index name and index user:

  ```
  DBMS_VECTOR.ENABLE_CHECKPOINT('VECTOR_USER','VIDX1');
  ```

- Using only the index user:

  ```
  DBMS_VECTOR.ENABLE_CHECKPOINT('VECTOR_USER');
  ```

Related Topics

- *Oracle Database AI Vector Search User's Guide*
- DISABLE_CHECKPOINT
  Use the DISABLE_CHECKPOINT procedure to disable the Checkpoint feature for a given Hierarchical Navigable Small World (HNSW) index user and HNSW index name. This operation purges all older checkpoints for the HNSW index. It also disables the creation of future checkpoints as part of the HNSW graph refresh.

DROP_ONNX_MODEL Procedure

This procedure deletes the specified ONNX model.

Syntax

```
DBMS_VECTOR.DROP_ONNX_MODEL (model_name IN VARCHAR2,
                             force      IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 216-3 DROP_ONNX_MODEL Procedure Parameters

Parameter	Description
model_name	Name of the machine learning ONNX model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
force	Forces the machine learning ONNX model to be dropped even if it is invalid. An ONNX model may be invalid if a serious system error interrupted the model build process.

Usage Note

To drop an ONNX model, you must be the owner or you must have the DB_DEVELOPER_ROLE.

Example

You can use the following command to delete a valid ONNX model named doc_model that exists in your schema.

```
BEGIN
  DBMS_VECTOR.DROP_ONNX_MODEL(model_name => 'doc_model');
END;
/
```

GET_INDEX_STATUS

Use the `GET_INDEX_STATUS` procedure to query the status of a vector index creation.

Syntax

```
DBMS_VECTOR.GET_INDEX_STATUS ('USER_NAME','INDEX_NAME');
```

USER_NAME

Specify the user name of the vector index owner.

INDEX_NAME

Specify the name of the vector index. You can query the index creation status for both Hierarchical Navigable Small World (HNSW) indexes and Inverted File Flat (IVF) indexes.

Usage Notes

- You can use the `GET_INDEX_STATUS` procedure only during a vector index creation.
- The `Percentage` value is shown in the output only for Hierarchical Navigable Small World (HNSW) indexes (and not for Inverted File Flat (IVF) indexes).
- Along with the `DB_DEVELOPER_ROLE` privilege, you must have read access to the `VECSYS.VECTOR$INDEX$BUILD$` table.
- You can use the following query to view all auxiliary tables:

  ```
  select IDX_AUXILIARY_TABLES from vecsys.vector$index;
  ```

  - For HNSW indexes:

    `rowid_vid_map` stores the mapping between a row ID and vector ID. `shared_journal_change_log` stores the DML changes that are yet to be incorporated into an HNSW graph.
  - For IVF indexes:

    `centroids` stores the location for each centroid. `centroid_partitions` stores the best centroid for each vector.
- The possible values of `Stage` for HNSW vector indexes are:

Value	Description
`HNSW Index Initialization`	Initialization phase for the HNSW vector index creation
`HNSW Index Auxiliary Tables Creation`	Creation of the internal auxiliary tables for the HNSW Neighbor Graph vector index
`HNSW Index Graph Allocation`	Allocation of memory from the vector memory pool for the HNSW graph
`HNSW Index Loading Vectors`	Loading of the base table vectors into the vector pool memory
`HNSW Index Graph Construction`	Creation of the multi-layered HNSW graph with the previously loaded vectors
`HNSW Index Creation Completed`	HNSW vector index creation finished

- The possible values of `Stage` for IVF vector indexes are:

Value	Description
`IVF Index Initialization`	Initialization phase for the IVF vector index creation
`IVF Index Centroids Creation`	The K-means clustering phase that computes the cluster centroids on a sample of base table vectors
`IVF Index Centroid Partitions Creation`	Centroids assignment phase for the base table vectors
`IVF Index Creation Completed`	IVF vector index creation completed

Example

```
exec DBMS_VECTOR.GET_INDEX_STATUS('VECTOR_USER','VIDX_HNSW');

Index objn: 74745
Stage: HNSW Index Loading Vectors
Percentage: 80%
```

INDEX_ACCURACY_QUERY

Use the `DBMS_VECTOR.INDEX_ACCURACY_QUERY` function to verify the accuracy of a vector index for a given query vector, top-K, and target accuracy.

Syntax

```
DBMS_VECTOR.INDEX_ACCURACY_QUERY (
    OWNER_NAME          IN VARCHAR2,
    INDEX_NAME          IN VARCHAR2,
    QV                  IN VECTOR,
    TOP_K               IN NUMBER,
    TARGET_ACCURACY     IN NUMBER
) return VARCHAR2;

DBMS_VECTOR.INDEX_ACCURACY_QUERY (
    OWNER_NAME          IN VARCHAR2,
    INDEX_NAME          IN VARCHAR2,
    QV                  IN VECTOR,
    TOP_K               IN NUMBER,
    QUERY_PARAM         IN JSON
) return VARCHAR2;
```

Parameters

Table 216-4 INDEX_ACCURACY_QUERY (IN) Parameters of DBMS_VECTOR

Parameter	Description
`owner_name`	The name of the vector index owner.
`index_name`	The name of the vector index.
`qv`	Specifies the query vector.
`top_k`	The `top_k` value for accuracy computation.

Table 216-4 (Cont.) INDEX_ACCURACY_QUERY (IN) Parameters of DBMS_VECTOR

Parameter	Description
`target_accuracy`	The target accuracy value for the vector index.

For information about determining the accuracy of your vector indexes, see Index Accuracy Report in *Oracle Database AI Vector Search User's Guide*.

INDEX_ACCURACY_REPORT

Use the `DBMS_VECTOR.INDEX_ACCURACY_REPORT` function to capture from your past workloads, accuracy values achieved by approximate searches using a particular vector index for a certain period of time.

Syntax

```
DBMS_VECTOR.INDEX_ACCURACY_REPORT (
    OWNER_NAME         IN VARCHAR2,
    INDEX_NAME         IN VARCHAR2,
    START_TIME         IN TIMESTAMP WITH TIME ZONE,
    END_TIME           IN TIMESTAMP WITH TIME ZONE
) return NUMBER;
```

Parameters

Table 216-5 INDEX_ACCURACY_REPORT (IN) Parameters of DBMS_VECTOR

Parameter	Description
`owner_name`	The name of the vector index owner.
`index_name`	The name of the vector index.
`start_time`	Specifies from what time to capture query vectors to consider for the accuracy computation. A `NULL start_time` uses query vectors captured in the last 24 hours.
`end_time`	Specifies an end point up until which query vectors are considered for accuracy computation. A `NULL end_time` uses query vectors captured from `start_time` until the current time.

For information about determining the accuracy of your vector indexes, see Index Accuracy Report in *Oracle Database AI Vector Search User's Guide*.

INDEX_VECTOR_MEMORY_ADVISOR

Use the INDEX_VECTOR_MEMORY_ADVISOR procedure to determine the vector memory size needed for a particular vector index. This helps you evaluate the number of indexes that can fit for each simulated vector memory size.

Syntax

- Using the number and type of vector dimensions that you want to store in your vector index.

```
DBMS_VECTOR.INDEX_VECTOR_MEMORY_ADVISOR(
    INDEX_TYPE         IN     VARCHAR2,
    NUM_VECTORS        IN     NUMBER,
    DIM_COUNT          IN     NUMBER,
    DIM_TYPE           IN     VARCHAR2,
    PARAMETER_JSON     IN     CLOB,
    RESPONSE_JSON      OUT    CLOB);
```

- Using the table and vector column on which you want to create your vector index:

```
DBMS_VECTOR.INDEX_VECTOR_MEMORY_ADVISOR(
    TABLE_OWNER         IN     VARCHAR2,
    TABLE_NAME          IN     VARCHAR2,
    COLUMN_NAME         IN     VARCHAR2,
    INDEX_TYPE          IN     VARCHAR2,
    PARAMETER_JSON      IN     CLOB,
    RESPONSE_JSON       OUT    CLOB);
```

Table 216-6 Syntax Details: INDEX_VECTOR_MEMORY_ADVISOR

Parameter	Description
INDEX_TYPE	Type of vector index: • IVF for Inverted File Flat (IVF) vector indexes • HNSW for Hierarchical Navigable Small World (HNSW) vector indexes
NUM_VECTORS	Number of vectors that you plan to create the vector index with.
DIM_COUNT	Number of dimensions of a vector as a NUMBER.
DIM_TYPE	Type of dimensions of a vector. Possible values are: • FLOAT16 • FLOAT32 • FLOAT64 • INT8
TABLE_OWNER	Owner name of the table on which to create the vector index.
TABLE_NAME	Table name on which to create the vector index.
COLUMN_NAME	Name of the vector column on which to create the vector index.

Table 216-6 (Cont.) Syntax Details: INDEX_VECTOR_MEMORY_ADVISOR

Parameter	Description
`PARAMETER_JSON`	Input parameter in JSON format. You can specify only one of the following form: • `PARAMTER_JSON=>{"accuracy":value}` • `INDEX_TYPE=>IVF, parameter_json=>{"neighbor_partitions":value}` • `INDEX_TYPE=>HNSW, parameter_json=>{"neighbors":value}` **Note**: You cannot specify values for `accuracy` along with `neighbor_partitions` or `neighbors`.
`RESPONSE_JSON`	JSON-formatted response string.

Examples

- Using neighbors in the parameters list:

```
SET SERVEROUTPUT ON;
DECLARE
    response_json CLOB;
BEGIN
    DBMS_VECTOR.INDEX_VECTOR_MEMORY_ADVISOR(
        INDEX_TYPE=>'HNSW',
        NUM_VECTORS=>10000,
        DIM_COUNT=>768,
        DIM_TYPE=>'FLOAT32',
        PARAMETER_JSON=>'{"neighbors":32}',
        RESPONSE_JSON=>response_json);
END;
/
```

  Result:

```
Suggested vector memory pool size: 59918628 Bytes
```

- Using accuracy in the parameters list:

```
SET SERVEROUTPUT ON;
DECLARE
    response_json CLOB;
BEGIN
    DBMS_VECTOR.INDEX_VECTOR_MEMORY_ADVISOR(
        INDEX_TYPE=>'HNSW',
        NUM_VECTORS=>10000,
        DIM_COUNT=>768,
        DIM_TYPE=>'FLOAT32',
        PARAMETER_JSON=>'{"accuracy":90}',
        RESPONSE_JSON=>response_json);
END;
/
```

Result:

```
Suggested vector memory pool size: 53926765 Bytes
```

- Using the table and vector column on which you want to create the vector index:

```
SET SERVEROUTPUT ON;
DECLARE
    response_json CLOB;
BEGIN
    DBMS_VECTOR.INDEX_VECTOR_MEMORY_ADVISOR(
        'VECTOR_USER',
        'VECTAB',
        'DATA_VECTOR',
        'HNSW',
        RESPONSE_JSON=>response_json);
END;
/
```

Example result:

```
Using default accuracy: 90%
Suggested vector memory pool size: 76396251 Bytes
```

Related Topics

- *Oracle Database AI Vector Search User's Guide*

LOAD_ONNX_MODEL

This procedure enables you to load an ONNX model into the Database.

Syntax

```
DBMS_VECTOR.LOAD_ONNX_MODEL (
     directory  VARCHAR2,
     file_name   VARCHAR2,
     model_name  VARCHAR2,
     metadata   JSON);
```

```
DBMS_VECTOR.LOAD_ONNX_MODEL(
model_name  IN  VARCHAR2,
model_data  IN  BLOB,
metadata    IN  JSON);
```

Parameters

Table 216-7   LOAD_ONNX_MODEL Procedure Parameters

Parameter	Description
`directory`	The directory name of the data dump. For example, `DM_DUMP`.
`file_name`	A `VARCHAR2` type parameter that specifies the name of the ONNX model.

Table 216-7 (Cont.) LOAD_ONNX_MODEL Procedure Parameters

Parameter	Description
`model_name`	Name of the model in the form `[schema_name.]model_name`. If you do not specify a schema, then your own schema is used.
`model_data`	It is a `BLOB` holding the ONNX representation of the model. The `BLOB` contains the identical byte sequence as the one stored in an ONNX file.
`metadata`	A JSON description of the metadata describing the model. The metadata at minimum must describe the machine learning function supported by the model. The model's metadata parameters are described in JSON Metadata Parameters for ONNX Models.

Examples

The following examples illustrates a code snippet of using the `DBMS_VECTOR.LOAD_ONNX_MODEL` procedure. The complete step-by-step example is illustrated in Import ONNX Models and Generate Embeddings.

```
EXECUTE DBMS_VECTOR.LOAD_ONNX_MODEL(
    'DM_DUMP',
    'my_embedding_model.onnx',
    'doc_model',
    JSON('{"function" : "embedding", "embeddingOutput" : "embedding", "input": {"input":
["DATA"]}}'));
```

```
DBMS_VECTOR.LOAD_ONNX_MODEL('my_embedding_model.onnx',
                                             :blob_bind_variable,
                                              JSON('{"function" :
"embedding",
                                                     "embeddingOutput" :
"embedding" ,
                                                      "input":{"input":
["DATA"]}}'));
```

For a complete example to illustrate how you can define a `BLOB` variable and use it in the `LOAD_ONNX_MODEL` procedure, you can have the following:

```
CREATE OR REPLACE MY_LOAD_EMBEDDING_MODEL(embedding_model_name VARCHAR2,
onnx_blob BLOB) IS
BEGIN
DBMS_VECTOR.LOAD_ONNX_MODEL(embedding_model_name,
                            onnx_blob,
                            JSON('{"function" : "embedding",
                                   "embeddingOutput" : "embedding" ,
                                   "input":{"input": ["DATA"]}}'));
END;
/
```

Usage Notes

The name of the model follows the same restrictions as those used for other machine learning models, namely:

- The schema name, if provided, is limited to 128 characters.
- The model name is limited to 123 characters and must follow the rules of unquoted identifiers: they contain only alphanumeric characters, the underscore (_), dollar sign ($), and pound sign (#). The initial character must be alphabetic.
- The model size is limited to 1 gigabyte.
- The model must not depend on external initializers. To know more about initializers and other ONNX concepts, see https://onnx.ai/onnx/intro/concepts.html.
- There are default input and output names for input and output attributes for models that are prepared by the Python utility. You can load those models without the JSON parameters. For example:

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

> **See Also:**
>
> *Oracle Machine Learning for SQL User’s Guide* for examples of using ONNX models for machine learning tasks

JSON Metadata Parameters for ONNX Models

When importing models using the `IMPORT_ONNX_MODEL` (`DBMS_DATA_MINING`), `LOAD_ONNX_MODEL` (`DBMS_VECTOR`), or `LOAD_ONNX_MODEL_CLOUD` (`DBMS_VECTOR`) procedures, you supply metadata as JSON parameters.

Parameters

Field	Value Type	Description
`function`	String	Specify regression, classification, clustering, or embedding. This is a mandatory setting. **NOTE**: The only JSON parameter required when importing the model is the machine learning function.
`input`	NA	Describes the model input mapping. See "Input" in Usage Notes.
`regressionOutput`	String	The name of the regression model output that stores the regression results. The output is expected to be a tensor of supported shape of any supported regression output type. See "Output" in Usage Notes.
`classificationProbOutput`	String	The name of the classification model output storing probabilities. The output is expected to be a tensor value of type float (width 32/64) of supported shape. See "Automatic normalization of output probabilities" in Usage Notes.
`clusteringDistanceOutput`	String	The name of the clustering model output storing distances. The output is of type float (width 16/32/64) of supported shape.
`clusteringProbOutput`	String	The name of the clustering model output storing probabilities. The output is of type float (width 16/32/64) of supported shape.

Field	Value Type	Description
`classificationLabelOutput`	String	The name of the model output holding label information. You have the following metadata parameters to specify the labels for classification: • `labels`: specify the labels directly in the JSON metadata • `classificationLabelOutput`: specify the model output that provides labels If you do not specify any value for this parameter or the function of the model is not classification, you will receive an error. The user can specify to use labels from the model directly by setting `classificationLabelOutput` to the model output holding the label information. The tensor output holding the label information must be the same size as the number of classes and must be of integer or string type. If the tensor that holds the labels is of string type, the returned type of the `PREDICTION` operator is `VARCHAR2`. If the tensor that holds the labels is of integer type, the returned type of the `PREDICTION` operator is `NUMBER`.
`normalizeProb`	String	Describes automatic normalization of output probabilities. See "Automatic normalization of output probabilities" in Usage Notes.
`labels`	NA	The labels used for classification. If you want to use custom labels, specify the labels using the labels field in the JSON metadata. The field can be set to an array of length equal to the number of classes. The labels for the class i must be stored at index i of the label array. If an array of strings is used, the returned type of the `PREDICTION` operator is `VARCHAR2`. The size of the string labels specified by the user cannot exceed *4000* bytes. If an array of numbers is used, the returned type of the `PREDICTION` operator is `NUMBER`. If you do not specify labels or `classificationLabelOutput`, classes are identified by integers in the range 1 to N where N is the number of classes. In this case, the returned type of the `PREDICTION` operator is `NUMBER`.
`embeddingOutput`	String	The model output that holds the generated embeddings.
`suitableDistanceMetrics`	String	An array of names of suitable distance metrics for the model. The names must be the names of the distance metrics used for the Oracle `VECTOR_DISTANCE` operator. To know the supported distance metrics, see Vector Distance Metrics. This parameter is for informational purposes only.

Field	Value Type	Description
`normalization`	Boolean	A boolean value indicates if normalization is applied to the output vector. The value 1 means normalization is applied. Normalization is process of converting an embedding vector so that it's norm or length equals 1. A normalized vector maintains its direction but its length becomes 1. The resulting vector is often called a unit vector.
`maxSequenceLength`	Number	The maximum length of the token (input) sequence that is meaningful for the model. This parameter sets a limit on the number of tokens, words, or elements in each input sequence that the model will process. This ensures uniform input size for the model. For example, the value could be 128, or 512 to 4096 depending on the task for which the parameter is used. A machine translation model might have a `maxSequenceLength` of 512, accommodating sentences or paragraphs up to 512 tokens for translation tasks. This parameter is for informational purposes only.
`pooling`	String	Indicates the pooling function performed on the output vector. This parameter is for informational purposes only.
`modelDescription`	Object	A JSON object that allows users to add additional descriptions to the models complementing the existing ONNX metadata for model description. This parameter is for informational purposes only.
`languages`	String	A comma-separated list of language name or abbreviation, as described in "A.1 Languages" of *Oracle Database Globalization Support Guide*. If you import multi-lingual embedding model, specify the language or the language abbreviation as the metadata. This parameter is for informational purposes only.
`tokenizer`	String	Tokenizers help in transforming text into words. There are several tokenizers available, including: bert, gpt2, bpe, wordpiece, sentencepiece, and clip. This parameter is for informational purposes only.
`embeddingLayer`	String	An identifier for the embedding layer. An embedding layer, serving as a hidden layer in neural networks, transforms input data from high to lower dimensions, enhancing the network's understanding of input relationships and data processing efficiency. Embedding layer helps in processing and analyzing categorical or discrete data. It achieves this by transforming categories into continuous embeddings, capturing the essential semantic relationships and similarities between them. For example the last hidden state in some transformer, or a layer in a resnet network. This parameter is for informational purposes only.

Field	Value Type	Description
defaultOnNull	NA	Specify the replacement of missing values in the JSON using the defaultOnNull field. If defaultOnNull is not specified, the replacement of missing values is not performed. The defaultOnNull sets the missing values to NULL by default. You can override the default value of NULL by providing meaningful default values to substitute for NULL. The field must be a JSON object literal, whose fields are the input attribute names and whose values are the default values for the input. Note that the default value is of type string and must be a valid Oracle PL/SQL NVL value for the given datatype.

Note: The parameters are case-sensitive. A number of default conventions for output parameter names and default values allows to minimize the information that you may have to provide. The parameters such as suitableDistanceMetrics are informational only and you are not expected to provide this information while importing the model. The JSON descriptor may specify only one input attribute. If more are specified, you will receive an error. You will receive an error if the normalizeProb field is specified as the JSON metadata parameter.

Usage Notes

The name of the model follows the same restrictions as those used for other machine learning models, namely:

- **Input**

  When importing a model from an ONNX representation, you must specify the name of the attribute used for scoring and how it maps to actual ONNX inputs. A scoring operator uses these attribute names to identify the columns to be used. (For example, PREDICTION). Follow these conventions to specify the attribute names using the input field:

  not specified: When the field input is not specified, attribute names are mapped directly to model inputs by name. That is, if the attribute name is not specified in the JSON metadata, then the name of the input tensor is used as an attribute name. Each model input must have dimension [batch_size, value]. If you do not specify input in the JSON metatdata, the value must be 1. You don't have to specify extra metadata if the input of the model already conforms to the format. For an embedding model, a single input is provided that may be used in batches. Here, if the input parameter is not specified in the JSON metadata, the valid model will have [batch_size, 1].

  You must ensure that all attribute names, whether implied by the model or explicitly set by you through the input field, are valid Oracle Database identifiers for column names. Each attribute name within a model must be unique, ensuring no duplicates exist.

  You can explicitly specify attribute name for model that use input tensors that have a dimension larger than 1 (for example, (batch_size, 2)). In this case, you must specify a name for each of these values for them to be interpreted as independent attribute name. This can be done for regression, classification, clustering which are models whose scoring operation can take multiple input attributes.

- **Output**

  As models might have multiple outputs, you can specify which output is of interest for a specific machine learning technique. You have the following ways to specify model outputs:

  - Specify the output name of interest in the JSON during model import. If the specified name is not a valid model output (see the table with valid outputs for a given machine learning function), you will receive an error.
  - If the model produces an output that matches the expected output name for the given machine learning technique (for example, `classificationProbOutput`) and you didn't explicitly specify it, the output is automatically assumed.
  - If you do not specify any output name and the model has a single output, the system assumes that the single output corresponds to a default specific to the machine learning technique. For an embedding machine learning function, the default value is `embeddingOutput`.
    The system reports an error if you do not specify model outputs or if you supply outputs that the specified machine learning function does not support. The following table displays supported outputs for a specific machine learning function:

Machine learning function	Output
regression	`regressionOutput`
classification	`classificationProbOutput`
clustering	`clusteringDistanceOutput`
embedding	`embeddingOutput`

    If none of the mentioned model outputs are specified, or if you supply outputs that are not supported by the specified machine learning function, you will receive an error.

- **Automatic Normalization of Output Probabilities**

  Many users widely employ the softmax function to normalize the output of multi-class classification models, as it enables to easily interpret the results of these models. The **softmax function** is a mathematical function that converts a vector of real numbers into a probability distribution. It is also known as the softargmax, or normalized exponential function. This function is available to you to specify at the model import-time that a softmax normalization must be applied to the tensor holding output probabilities such as `classificationProbOutput` and `clusteringProbOutput`. Specify `normalizeProb` to define the normalization that must be applied for softmax normalization. The default setting is `none`, indicating that no normalization is applied. You can choose `softmax` to apply a softmax function to the probability output. Specifying any other value for this field will result in an error during import. Additionally, specifying this field for models other than classification and clustering will also lead to an error.

Example: Specifying JSON Metadata Parameters for Embedding Models

The following example illustrates a simple case of how you can specify JSON metadata parameters while importing an ONNX embedding model into the Database using the `DBMS_VECTOR.IMPORT_ONNX_MODEL` procedure.

```
DBMS_VECTOR.IMPORT_ONNX_MODEL('my_embedding_model.onnx', 'doc_model',
                 JSON('{"function" : "embedding",
                       "embeddingOutput" : "embedding" ,
                        "input":{"input": ["DATA"]}}'));
```

Example: Specifying Complete JSON Metadata Parameters for Embedding Models

The following example illustrates how to provide a complete JSON metadata parameters, with an exception of `embeddingLayer`, for importing embedding models.

```
DECLARE
  metadata JSON;
  mdtxt varchar2(4000);
BEGIN
  metadata := JSON(q'#
           {
             "function"                : "embedding",
             "embeddingOutput"         : "embedding",
             "input"                   : { "input" : ["txt"]},
             "maxSequenceLength"       : 512,
             "tokenizer"               : "bert",
             "suitableDistanceMetrics" : [ "DOT", "COSINE", "EUCLIDEAN"],
             "pooling"                 : "Mean Pooling",
             "normalization"           : true,
             "languages"               : ["US"],
             "modelDescription"        : {
                 "description" : "This model was tuned for semantic search:
Given a query/question, if can find relevant passages. It was trained on a
large and diverse set of (question, a
nswer) pairs.",
                 "url" : "https://example.co/sentence-transformers/
my_embedding_model"}
           }
           #');
  -- load the onnx model
    DBMS_VECTOR.IMPORT_ONNX_MODEL('my_embedding_model.onnx', 'doc_model',
metadata);
END;
/
```

> **See Also:**
>
> *Oracle Machine Learning for SQL User's Guide* for examples of using ONNX models for machine learning tasks

LOAD_ONNX_MODEL_CLOUD

This procedure enables you to load an ONNX model from object storage into the Database.

Syntax

```
DBMS_VECTOR.LOAD_ONNX_MODEL_CLOUD (
     model_name   IN  VARCHAR2,
     credential   IN  VARCHAR2,
     uri          IN  VARCHAR2,
     metadata     IN  JSON DEFAULT JSON('{"function" : "embedding", '||
                          '"embeddingOutput" : "embedding", "input": {"input":
["DATA"]}}')
);
```

Parameters

Table 216-8   LOAD_ONNX_MODEL_CLOUD Procedure Parameters

Parameter	Description
`model_name`	The name of the model in the form `[schema_name.]model_name`. If you do not specify a schema, then your own schema is used.
`credential`	The name of the credential to be used to access Object Store.
`uri`	The URI of the ONNX model.
`metadata`	A JSON description of the metadata describing the model. The metadata at minimum must describe the machine learning function supported by the model. The model's metadata parameters are described in JSON Metadata Parameters for ONNX Models.

Examples

The following example includes a code snippet of using the `DBMS_VECTOR.LOAD_ONNX_MODEL_CLOUD` procedure.

```
EXECUTE DBMS_VECTOR.LOAD_ONNX_MODEL_CLOUD(
    model_name => 'database',
    credential => 'MYCRED',
    uri => 'https://objectstorage.us-phoenix-1.oraclecloud.com/n/namespace-string/b/
bucketname/o/all-MiniLM-L6-v2.onnx',
    metadata => JSON('{"function" : "embedding", "embeddingOutput" : "embedding" , "input":
{"input": ["DATA"]}}')
);
```

Usage Notes

The name of the model follows the same restrictions as those used for other machine learning models, namely:

- The schema name, if provided, is limited to 128 characters.
- The model name is limited to 123 characters and must follow the rules of unquoted identifiers: they contain only alphanumeric characters, the underscore (_), dollar sign ($), and pound sign (#). The initial character must be alphabetic.
- The model size is limited to 1 gigabyte.

- The model must not depend on external initializers. To know more about initializers and other ONNX concepts, see https://onnx.ai/onnx/intro/concepts.html.
- There are default input and output names for input and output attributes for models that are prepared by the Python utility. You can load those models without the JSON parameters. For example:

  ```
  EXECUTE DBMS_VECTOR.LOAD_ONNX_MODEL_CLOUD(
      'database',
      'MYCRED',
      'https://objectstorage.us-phoenix-1.oraclecloud.com/n/namespace-
  string/b/bucketname/o/all-MiniLM-L6-v2.onnx'
  );
  ```

> **See Also:**
>
> *Oracle Machine Learning for SQL User's Guide* for examples of using ONNX models for machine learning tasks

QUERY

Use the `DBMS_VECTOR.QUERY` function to perform a similarity search operation which returns the top-k results as a JSON array.

Syntax

Query is overloaded and supports a version with `query_vector` passed in as a `VECTOR` type in addition to `CLOB`.

```
DBMS_VECTOR.QUERY (
    TAB_NAME              IN VARCHAR2,
    VEC_COL_NAME          IN VARCHAR2,
    QUERY_VECTOR          IN CLOB,
    TOP_K                 IN NUMBER,
    VEC_PROJ_COLS         IN JSON_ARRAY_T DEFAULT NULL,
    IDX_NAME              IN VARCHAR2 DEFAULT NULL,
    DISTANCE_METRIC       IN VARCHAR2 DEFAULT 'COSINE',
    USE_INDEX             IN BOOLEAN DEFAULT TRUE,
    ACCURACY              IN NUMBER DEFAULT '90',
    IDX_PARAMETERS        IN CLOB DEFAULT NULL
) return JSON_ARRAY_T;

DBMS_VECTOR.QUERY (
    TAB_NAME              IN VARCHAR2,
    VEC_COL_NAME          IN VARCHAR2,
    QUERY_VECTOR          IN VECTOR,
    TOP_K                 IN NUMBER,
    VEC_PROJ_COLS         IN JSON_ARRAY_T DEFAULT NULL,
    IDX_NAME              IN VARCHAR2 DEFAULT NULL,
    DISTANCE_METRIC       IN VARCHAR2 DEFAULT 'COSINE',
    USE_INDEX             IN BOOLEAN DEFAULT TRUE,
    ACCURACY              IN NUMBER DEFAULT '90',
```

```
    IDX_PARAMETERS       IN CLOB DEFAULT NULL
) return JSON_ARRAY_T;
```

Parameters

Specify the input parameters in JSON format.

Table 216-9 DBMS_VECTOR.QUERY Parameters

Parameter	Description
`tab_name`	Table name to query
`vec_col_name`	Vector column name
`query_vector`	Query vector passed in as `CLOB` or `VECTOR`.
`top_k`	Number of results to be returned.
`vec_proj_cols`	Columns to be projected as part of the result.
`idx_name`	Name of the index queried.
`distance_metric`	Distance computation metric. Defaults to `COSINE`. Can also be `MANHATTAN`, `HAMMING`, `DOT`, `EUCLIDEAN`, `L2_SQUARED`, `EUCLIDEAN_SQUARED`. .
`use_index`	Specifies whether the search is an approximate search or exact search. Defaults to TRUE (that is, approximate).
`accuracy`	Specifies the minimum desired query accuracy.
`idx_parameters`	Specifies values of `efsearch` and `neighbor partition probes` passed in, formatted as JSON

DATA

This function accepts the input data type as `VARCHAR2`, `NUMBER`, `JSON`, `BOOLEAN` or `CLOB`.

REBUILD_INDEX

Use the `DBMS_VECTOR.REBUILD_INDEX` function to rebuild a vector index.

Purpose

To rebuild a vector index such as Hierarchical Navigable Small World (HNSW) vector index or Inverted File Flat (IVF) vector index. In case only the `idx_name` is provided, it rebuilds the index using `get_ddl`. When all the parameters are provided, it performs a drop index followed by a call to `dbms_vector.create_index()`.

Syntax

```
DBMS_VECTOR.REBUILD_INDEX (
    idx_name                    IN VARCHAR2,
    table_name                  IN VARCHAR2 DEFAULT NULL,
    idx_vector_col              IN VARCHAR2 DEFAULT NULL,
    idx_include_cols            IN VARCHAR2 DEFAULT NULL,
    idx_partitioning_scheme     IN VARCHAR2 DEFAULT NULL,
    idx_organization            IN VARCHAR2 DEFAULT NULL,
    idx_distance_metric         IN VARCHAR2 DEFAULT 'COSINE',
    idx_accuracy                IN NUMBER DEFAULT 90,
    idx_parameters              IN CLOB DEFAULT NULL,
    idx_parallel_creation       IN NUMBER DEFAULT 1,
);
```

Parameters

Parameter	Description
`idx_name`	Name of the index to rebuild.
`table_name`	Table on which to create the index.
`idx_vector_col`	Vector column on which to create the index.
`idx_include_cols`	A comma-separated list of column names to be covered by the index.
`idx_partitioning_scheme`	Partitioning scheme for IVF indexes: • `GLOBAL` • `LOCAL` IVF indexes support both global and local indexes on partitioned tables. By default, these indexes are globally partitioned by centroid. You can choose to create a local IVF index, which provides a one-to-one relationship between the base table partitions or subpartitions and the index partitions. For detailed information on these partitioning schemes, see Inverted File Flat Vector Indexes Partitioning Schemes.
`idx_organization`	Index organization: • `NEIGHBOR PARTITIONS` • `INMEMORY NEIGHBOR GRAPH` For detailed information on these organization types, see Manage the Different Categories of Vector Indexes.
`idx_distance_metric`	Distance metric or mathematical function used to compute the distance between vectors: • `COSINE` (default) • `MANHATTAN` • `HAMMING` • `JACCARD` • `DOT` • `EUCLIDEAN` • `L2_SQUARED` • `EUCLIDEAN_SQUARED` For detailed information on each of these metrics, see Vector Distance Functions and Operators.

Parameter	Description
`idx_accuracy`	Target accuracy at which the approximate search should be performed when running an approximate search query. As explained in Understand Approximate Similarity Search Using Vector Indexes, you can specify non-default target accuracy values either by specifying a percentage value or by specifying internal parameters values, depending on the index type you are using. • For an HNSW approximate search: In the case of an HNSW approximate search, you can specify a target accuracy percentage value to influence the number of candidates considered to probe the search. This is automatically calculated by the algorithm. A value of 100 will tend to impose a similar result as an exact search, although the system may still use the index and will not perform an exact search. The optimizer may choose to still use an index as it may be faster to do so given the predicates in the query. Instead of specifying a target accuracy percentage value, you can specify the `EFSEARCH` parameter to impose a certain maximum number of candidates to be considered while probing the index. The higher that number, the higher the accuracy. For detailed information, see Understand Hierarchical Navigable Small World Indexes. • For an IVF approximate search: In the case of an IVF approximate search, you can specify a target accuracy percentage value to influence the number of partitions used to probe the search. This is automatically calculated by the algorithm. A value of 100 will tend to impose an exact search, although the system may still use the index and will not perform an exact search. The optimizer may choose to still use an index as it may be faster to do so given the predicates in the query. Instead of specifying a target accuracy percentage value, you can specify the `NEIGHBOR PARTITION PROBES` parameter to impose a certain maximum number of partitions to be probed by the search. The higher that number, the higher the accuracy. For detailed information, see Understand Inverted File Flat Vector Indexes.

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td><code>idx_parameters</code></td><td>
<p>Type of vector index and associated parameters.</p>
<p>Specify the indexing parameters in JSON format:</p>
<ul>
<li>For HNSW indexes:
<ul>
<li><code>type</code>: Type of vector index to create, that is, <code>HNSW</code></li>
<li><code>neighbors</code>: Maximum number of connections permitted per vector in the HNSW graph</li>
<li><code>efConstruction</code>: Maximum number of closest vector candidates considered at each step of the search during insertion</li>
</ul>
<p>For example:</p>
<pre>
{
    "type"   : "HNSW",
    "neighbors"   : 3,
    "efConstruction" : 4
}
</pre>
<p>For detailed information on these parameters, see Hierarchical Navigable Small World Index Syntax and Parameters.</p>
</li>
<li>For IVF indexes:
<ul>
<li><code>type</code>: Type of vector index to create, that is, <code>IVF</code></li>
<li><code>partitions</code>: Neighbor partition or cluster in which you want to divide your vector space</li>
</ul>
<p>For example:</p>
<pre>
{
    "type"   : "IVF",
    "partitions" : 5
}
</pre>
<p>For detailed information on these parameters, see Inverted File Flat Index Syntax and Parameters.</p>
</li>
</ul>
</td></tr>
<tr><td><code>idx_parallel_creation</code></td><td>Number of parallel threads used for index construction.</td></tr>
</table>

Examples

- Specify neighbors and efConstruction for HNSW indexes:

```
dbms_vector.rebuild_index(
    'v_hnsw_01',
    'vpt01',
    'EMBEDDING',
     NULL,
     NULL,
    'INMEMORY NEIGHBOR GRAPH',
    'EUCLIDEAN',
     95,
    '{"type" : "HNSW", "neighbors" : 3, "efConstruction" : 4}');
```

- Specify the number of partitions for IVF indexes:

```
dbms_vector.rebuild_index(
    'V_IVF_01',
    'vpt01',
    'EMBEDDING',
     NULL,
     NULL,
    'NEIGHBOR PARTITIONS',
    'EUCLIDEAN',
     95,
    '{"type" : "IVF", "partitions" : 5}');
```

RERANK

Use the `DBMS_VECTOR.RERANK` function to reassess and reorder an initial set of results to retrieve more relevant search output.

Purpose

To improve the relevance and quality of search results in both similarity search and Retrieval Augmented Generation (RAG) scenarios.

Reranking improves the quality of information ingested into an LLM by ensuring that the most relevant documents or chunks are prioritized. This helps to reduce hallucinations and improves the accuracy of generated outputs.

For this operation, Oracle AI Vector Search supports reranking models provided by Cohere and Vertex AI.

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

Syntax

```
DBMS_VECTOR.RERANK(
                QUERY      IN CLOB,
                DOCUMENTS  IN JSON,
                PARAMS     IN JSON default NULL
) return JSON;
```

This function accepts the input containing a query as CLOB and a list of documents in JSON format. It then processes this information to generate a JSON object containing a reranked list of documents, sorted by score.

For example, a reranked output includes:

```
{
    "index"   : "1",
    "score"   : "0.99",
    "content" : "Jupiter boasts an impressive system of 95 known moons."
}
```

Where,

- index specifies the position of the document in the list of input text.
- score specifies the relevance score.
- content specifies the input text corresponding to the index.

QUERY

Specify the search query (typically from an initial search) as CLOB.

DOCUMENTS

Specify a JSON array of strings (list of potentially relevant documents to rerank) in the following format:

```
{
  "documents": [
  "string1",
  "string2",
    ...
  ]
}
```

PARAMS

Specify the following list of parameters in JSON format. All these parameters are mandatory.

```
{
  "provider"          : "<service provider>",
  "credential_name"   : "<credential name>",
  "url"               : "<REST endpoint URL for reranking>",
  "model"             : "<reranking model name>",
  ...
}
```

Table 216-10 RERANK Parameter Details

Parameter	Description
provider	Supported REST provider to access for reranking: • cohere • vertexai

Table 216-10 (Cont.) RERANK Parameter Details

Parameter	Description
`credential_name`	Name of the credential in the form: *`schema.credential_name`* A credential name holds authentication credentials to enable access to your provider for making REST API calls. You need to first set up your credential by calling the `DBMS_VECTOR.CREATE_CREDENTIAL` helper function to create and store a credential, and then refer to the credential name here. See CREATE_CREDENTIAL.
`url`	URL of the third-party provider endpoint for each REST call, as listed in Supported Third-Party Provider Operations and Endpoints.
`model`	Name of the reranking model in the form: *`schema.model_name`* If the model name is not schema-qualified, then the schema of the procedure invoker is used.

Additional REST provider parameters:

Optionally, specify additional provider-specific parameters for reranking.

Important:

- The following examples are for illustration purposes. For accurate and up-to-date information on additional parameters to use, refer to your third-party provider's documentation.
- For a list of all supported REST endpoints, see Supported Third-Party Provider Operations and Endpoints.

Cohere example:

```
{
  "provider"        : "cohere",
  "credential_name" : "COHERE_CRED",
  "url"             : "https://api.cohere.example.com/rerank",
  "model"           : "rerank-english-v3.0",
  "return_documents": false,
  "top_n"           : 3
}
```

Vertex AI example:

```
{
  "provider"         : "vertexai",
  "credential_name"  : "VERTEXAI_CRED",
  "url"              : "https://googleapis.example.com/
default_ranking_config:rank",
  "model"            : "semantic-ranker-512@latest",
"ignoreRecordDetailsInResponse" : true,
"topN"                          : 3
}
```

Table 216-11   Additional REST Provider Parameter Details

Parameter	Description
`return_documents`	Whether to return search results with original documents or input text (`content`): • `false` (default, also recommended) to not return any input text (return only index and score) • `true` to return input text along with index and score **Note**: With Cohere as the provider, Oracle recommends that you keep this option disabled for better performance. You may choose to enable it for debugging purposes when you need to view the original text.
`ignoreRecordDetailsInResponse`	Whether to return search results with original record details or input text (`content`): • `false` (default) to return input text along with index and score • `true` (recommended) to not return any input text (return only index and score) **Note**: With Vertex AI as the provider, Oracle recommends that you keep this option enabled for better performance. You may choose to disable it for debugging purposes when you need to view the original text.
`top_n` or `topN`	The number of most relevant documents to return.

Examples

- Using Cohere:

```
declare
  params clob;
  reranked_output json;
begin
  params := '
{
  "provider": "cohere",
  "credential_name": "COHERE_CRED",
  "url": "https://api.cohere.com/v1/rerank",
  "model": "rerank-english-v3.0",
  "return_documents": true,
  "top_n": 3
}';

  reranked_output := dbms_vector.rerank(:query,
json(:initial_retrieval_docs), json(params));
  dbms_output.put_line(json_serialize(reranked_output));
end;
/
```

- Using Vertex AI:

```
declare
  params clob;
  reranked_output json;
begin
  params := '
{
  "provider": "vertexai",
  "credential_name": "VERTEXAI_CRED",
  "url": "https://discoveryengine.googleapis.com/v1/projects/1085581009881/
locations/global/rankingConfigs/default_ranking_config:rank",
  "model": "semantic-ranker-512@latest",
  "ignoreRecordDetailsInResponse": false,
  "topN": 3
}';

  reranked_output := dbms_vector.rerank(:query,
json(:initial_retrieval_docs), json(params));
  dbms_output.put_line(json_serialize(reranked_output));
end;
/
```

End-to-end example:

To run an end-to-end example scenario using this function, see Use Reranking for Better RAG Results.

UTL_TO_CHUNKS

Use the `DBMS_VECTOR.UTL_TO_CHUNKS` chainable utility function to split a large plain text document into smaller chunks of text.

Purpose

To perform a text-to-chunks transformation. This chainable utility function internally calls the `VECTOR_CHUNKS` SQL function for the operation.

To embed a large document, you may first need to split it into multiple appropriate-sized segments or chunks through a splitting process known as chunking (as explained in Understand the Stages of Data Transformations). A chunk can be words (to capture specific words or word pieces), sentences (to capture a specific meaning), or paragraphs (to capture broader themes). A single document may be split into multiple chunks, each transformed into a vector.

Syntax

```
DBMS_VECTOR.UTL_TO_CHUNKS (
    DATA          IN  CLOB | VARCHAR2,
    PARAMS        IN  JSON  default  NULL
) return VECTOR_ARRAY_T;
```

DATA

This function accepts the input data type as `CLOB` or `VARCHAR2`.

It returns an array of CLOBs, where each CLOB contains a chunk along with its metadata in JSON format, as follows:

```
{
    "chunk_id"     : NUMBER,
    "chunk_offset" : NUMBER,
    "chunk_length" : NUMBER,
    "chunk_data"   : "VARCHAR2(4000)"
}
```

For example:

```
{"chunk_id":1,"chunk_offset":1,"chunk_length":6,"chunk_data":"sample"}
```

Where,

- `chunk_id` specifies the chunk ID for each chunk.
- `chunk_offset` specifies the original position of each chunk in the source document, relative to the start of document which has a position of `1`.
- `chunk_length` specifies the character length of each chunk.
- `chunk_data` displays text pieces from each chunk.

PARAMS

Specify input parameters in JSON format:

```
{
    "by"           :     mode,
    "max"          :     max,
    "overlap"      :     overlap,
    "split"        :     split_condition,
    "custom_list"  :     [ split_chars1, ... ],
    "vocabulary"   :     vocabulary_name,
    "language"     :     nls_language,
    "normalize"    :     normalize_mode,
    "norm_options" :     [ normalize_option1, ... ],
    "extended"     :     boolean
}
```

For example:

```
JSON('
  { "by"           :    "vocabulary",
    "vocabulary"   :    "myvocab",
    "max"          :    "100",
    "overlap"      :    "0",
    "split"        :    "custom",
    "custom_list"  :    [ "<p>" , "<s>" ],
    "language"     :    "american",
    "normalize"    :    "options",
    "norm_options" :    [ "WHITESPACE" ]
  }')
```

Here is a complete description of these parameters:

Parameter	Description and Acceptable Values
`by`	Specify a mode for splitting your data, that is, to split by counting the number of characters, words, or vocabulary tokens. **Valid values**: • `characters` (or `chars`): Splits by counting the number of characters. • `words`: Splits by counting the number of words. Words are defined as sequences of alphabetic characters, sequences of digits, individual punctuation marks, or symbols. For segmented languages without whitespace word boundaries (such as Chinese, Japanese, or Thai), each native character is considered a word (that is, unigram). • `vocabulary`: Splits by counting the number of vocabulary tokens. Vocabulary tokens are words or word pieces, recognized by the vocabulary of the tokenizer that your embedding model uses. You can load your vocabulary file using the chunker helper API `DBMS_VECTOR_CHAIN.CREATE_VOCABULARY`. **Note**: For accurate results, ensure that the chosen model matches the vocabulary file used for chunking. If you are not using a vocabulary file, then ensure that the input length is defined within the token limits of your model. **Default value**: `words`
`max`	Specify a limit on the maximum size of each chunk. This setting splits the input text at a fixed point where the maximum limit occurs in the larger text. The units of `max` correspond to the `by` mode, that is, to split data when it reaches the maximum size limit of a certain number of characters, words, numbers, punctuation marks, or vocabulary tokens. **Valid values**: • `by characters`: `50` to `4000` characters • `by words`: `10` to `1000` words • `by vocabulary`: `10` to `1000` tokens **Default value**: `100`

Parameter	Description and Acceptable Values

`split [by]`	Specify where to split the input text when it reaches the maximum size limit. This helps to keep related data together by defining appropriate boundaries for chunks.
	Valid values:
	- `none`:
	Splits at the `max` limit of characters, words, or vocabulary tokens.
	- `newline`, `blankline`, and `space`:
	These are single-split character conditions that split at the last split character before the `max` value.
	Use `newline` to split at the end of a line of text. Use `blankline` to split at the end of a blank line (sequence of characters, such as two newlines). Use `space` to split at the end of a blank space.
	- `recursively`:
	This is a multiple-split character condition that breaks the input text using an ordered list of characters (or sequences).
	`recursively` is predefined as `BLANKLINE`, `newline`, `space`, `none` in this order:
	1. If the input text is more than the `max` value, then split by the first split character.
	2. If that fails, then split by the second split character.
	3. And so on.
	4. If no split characters exist, then split by `max` wherever it appears in the text.
	- `sentence`:
	This is an end-of-sentence split condition that breaks the input text at a sentence boundary.
	This condition automatically determines sentence boundaries by using knowledge of the input language's sentence punctuation and contextual rules. This language-specific condition relies mostly on end-of-sentence (EOS) punctuations and common abbreviations.
	Contextual rules are based on word information, so this condition is only valid when splitting the text by words or vocabulary (not by characters).
	Note: This condition obeys the `by word` and `max` settings, and thus may not determine accurate sentence boundaries in some cases. For example, when a sentence is larger than the `max` value, it splits the sentence at `max`. Similarly, it includes multiple sentences in the text only when they fit within the `max` limit.
	- `custom`:
	Splits based on a custom split characters list. You can provide custom sequences up to a limit of `16` split character strings, with a maximum length of `10` each.
	Specify an array of valid text literals using the `custom_list` parameter.
	`{`
	`    "split"        :  "custom",`
	`    "custom_list"  :  [ "split_chars1", ... ]`
	`}`
	For example:
	`{`
	`    "split"        :    "custom",`
	`    "custom_list"  :    [ "<p>" , "<s>" ]`
	`}`
	Note: You can omit sequences only for tab (`\t`), newline (`\n`), and linefeed (`\r`).
	Default value: `recursively`

Parameter	Description and Acceptable Values
`overlap`	Specify the amount (as a positive integer literal or zero) of the preceding text that the chunk should contain, if any. This helps in logically splitting up related text (such as a sentence) by including some amount of the preceding chunk text. The amount of overlap depends on how the maximum size of the chunk is measured (in characters, words, or vocabulary tokens). The overlap begins at the specified `split` condition (for example, at `newline`). **Valid value**: `5%` to `20%` of `max` **Default value**: `0`

`language`	Specify the language of your input data.
	This clause is important, especially when your text contains certain characters (for example, punctuations or abbreviations) that may be interpreted differently in another language.
	Valid values:
	• NLS-supported language name or its abbreviation, as listed in *Oracle Database Globalization Support Guide*.
	• Custom language name or its abbreviation, as listed in Supported Languages and Data File Locations. You use the `DBMS_VECTOR_CHAIN.CREATE_LANG_DATA` chunker helper API to load language-specific data (abbreviation tokens) into the database, for your specified language.
	Note: You must use escape characters with any language abbreviation that is also a SQL reserved word (for example, language abbreviations such as `IN`, `AS`, `OR`, `IS`).
	For example:
	`SELECT dbms_vector_chain.utl_to_chunks('this is an example',`
	`   JSON('{ "language" : "\"in\"" }'))`
	`from dual;`
	
	`SELECT dbms_vector_chain.utl_to_chunks('this is an example',`
	`   JSON_OBJECT('language' value '"in"' RETURNING JSON))`
	`from dual;`
	
	Default value: `NLS_LANGUAGE` from session

Parameter	Description and Acceptable Values

`normalize`	Automatically pre-processes or post-processes issues (such as multiple consecutive spaces and smart quotes) that may arise when documents are converted into text. Oracle recommends you to use a normalization mode to extract high-quality chunks.
	Valid values:
	• `none`:
	Applies no normalization.
	• `all`:
	Normalizes common multi-byte (unicode) punctuation to standard single-byte.
	• `options`:
	Specify an array of normalization options using the `norm_options` parameter.
	`{`
	`    "normalize"    :  "options",`
	`    "norm_options" :  [ "normalize_option1", ... ]`
	`}`
	– `punctuation`:
	Converts quotes, dashes, and other punctuation characters supported in the character set of the text to their common ASCII form. For example:
	* U+2013 (En Dash) maps to U+002D (Hyphen-Minus)
	* U+2018 (Left Single Quotation Mark) maps to U+0027 (Apostrophe)
	* U+2019 (Right Single Quotation Mark) maps to U+0027 (Apostrophe)
	* U+201B (Single High-Reversed-9 Quotation Mark) maps to U+0027 (Apostrophe)
	– `whitespace`:
	Minimizes whitespace by eliminating unnecessary characters.
	For example, retain blanklines, but remove any extra newlines and interspersed spaces or tabs: `" \n \n " => "\n\n"`
	– `widechar`:
	Normalizes wide, multi-byte digits and (a-z) letters to single-byte.
	These are multi-byte equivalents for `0-9` and `a-z A-Z`, which can show up in Chinese, Japanese, or Korean text.
	For example:
	`{`
	`    "normalize"    :  "options",`
	`    "norm_options" :  [ "whitespace" ]`
	`}`
	Default value: `none`

`extended`	Increases the output limit of a `VARCHAR2` string to `32767` bytes, without requiring you to set the `max_string_size` parameter to `extended`. **Default value**: `4000` or `32767` (when `max_string_size=extended`)

Example

```
SELECT D.id doc,
    JSON_VALUE(C.column_value, '$.chunk_id' RETURNING NUMBER) AS id,
    JSON_VALUE(C.column_value, '$.chunk_offset' RETURNING NUMBER) AS pos,
    JSON_VALUE(C.column_value, '$.chunk_length' RETURNING NUMBER) AS siz,
    JSON_VALUE(C.column_value, '$.chunk_data') AS txt
```

```
FROM docs D,
   dbms_vector.utl_to_chunks(D.text,
   JSON('{ "by"       : "words",
           "max"      : "100",
           "overlap"  : "0",
           "split"    : "recursively",
           "language" : "american",
           "normalize": "all" }')) C;
```

End-to-end examples:

To run end-to-end example scenarios using this function, see Perform Chunking With Embedding and Configure Chunking Parameters.

Related Topics

- VECTOR_CHUNKS

UTL_TO_EMBEDDING and UTL_TO_EMBEDDINGS

Use the `DBMS_VECTOR.UTL_TO_EMBEDDING` and `DBMS_VECTOR.UTL_TO_EMBEDDINGS` chainable utility functions to generate one or more vector embeddings from textual documents and images.

Purpose

To automatically generate one or more vector embeddings from textual documents and images.

- Text to Vector:

  You can perform a text-to-embedding transformation by accessing either Oracle Database or a third-party service provider:

  – Oracle Database as the service provider (default setting):

    This API calls an ONNX format embedding model that you load into the database.

  – Third-party embedding model:

    This API makes a REST API call to your chosen remote service provider (Cohere, Generative AI, Google AI, Hugging Face, OpenAI, or Vertex AI) or local service provider (Ollama).

- Image to Vector:

  You can also perform an image-to-embedding transformation. This API makes a REST call to your chosen image embedding model or multimodal embedding model by Vertex AI. Note that currently Vertex AI is the only supported service provider for this operation.

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

Syntax

- Text to Vector:

  ```
  DBMS_VECTOR.UTL_TO_EMBEDDING (
      DATA           IN CLOB,
      PARAMS         IN JSON default NULL
  ) return VECTOR;


  DBMS_VECTOR.UTL_TO_EMBEDDINGS (
      DATA           IN VECTOR_ARRAY_T,
      PARAMS         IN JSON default NULL
  ) return VECTOR_ARRAY_T;
  ```

- Image to Vector:

  ```
  DBMS_VECTOR.UTL_TO_EMBEDDING (
      DATA           IN BLOB,
      MODALITY       IN VARCHAR2,
      PARAMS         IN JSON default NULL
  ) return VECTOR;
  ```

DATA

- Text to Vector:

  `UTL_TO_EMBEDDING` accepts the input as `CLOB` containing textual data (text strings or small documents). It then converts the text to a single embedding (`VECTOR`).

  `UTL_TO_EMBEDDINGS` converts an array of chunks (`VECTOR_ARRAY_T`) to an array of embeddings (`VECTOR_ARRAY_T`).

  > **Note:**
  >
  > Although data is a `CLOB` or a `VECTOR_ARRAY_T` of `CLOB`, the maximum input is 4000 characters. If you have input that is greater, you can use `UTL_TO_CHUNKS` to split the data into smaller chunks before passing in.

- Image to Vector:

`UTL_TO_EMBEDDING` accepts the input as `BLOB` containing media data for media files such as images. It then converts the image input to a single embedding (`VECTOR`).

A generated embedding output includes:

```
{
    "embed_id"    :  NUMBER,
    "embed_data"  : "VARCHAR2(4000)",
    "embed_vector": "CLOB"
}
```

Where,

- `embed_id` displays the ID number of each embedding.
- `embed_data` displays the input text that is transformed into embeddings.
- `embed_vector` displays the generated vector representations.

MODALITY

For `BLOB` inputs, specify the type of content to vectorize. The only supported value is `image`.

PARAMS

Specify input parameters in JSON format, depending on the service provider that you want to use.

If using Oracle Database as the provider:

```
{
  "provider" : "database",
  "model"    : "<in-database ONNX embedding model filename>"
}
```

Table 216-12   Database Provider Parameter Details

Parameter	Description
`provider`	Specify `database` (default setting) to use Oracle Database as the provider. With this setting, you must load an ONNX format embedding model into the database.
`model`	User-specified name under which the imported ONNX embedding model is stored in Oracle Database. If you do not have an embedding model in ONNX format, then perform the steps listed in Convert Pretrained Models to ONNX Format.

If using a third-party provider:

Set the following parameters along with additional embedding parameters specific to your provider:

- For `UTL_TO_EMBEDDING`:

  ```
  {
    "provider"        : "<AI service provider>",
    "credential_name" : "<credential name>",
    "url"             : "<REST endpoint URL for embedding service>",
  ```

```
  "model"            : "<REST provider embedding model name>",
  "transfer_timeout": <maximum wait time for the request to complete>,
  "max_count": "<maximum calls to the AI service provider>",
  "<additional REST provider parameter>": "<REST provider parameter 
value>"
}
```

- For `UTL_TO_EMBEDDINGS`:

```
{
  "provider"         : "<AI service provider>",
  "credential_name" : "<credential name>",
  "url"              : "<REST endpoint URL for embedding service>",
  "model"            : "<REST provider embedding model name>",
  "transfer_timeout": <maximum wait time for the request to complete>,
  "batch_size"       : "<number of vectors to request at a time>",
  "max_count": "<maximum calls to the AI service provider>",
  "<additional REST provider parameter>": "<REST provider parameter 
value>"
}
```

Table 216-13 Third-Party Provider Parameter Details

Parameter	Description
`provider`	Third-party service provider that you want to access for this operation. A REST call is made to the specified provider to access its embedding model. For image input, specify `vertexai`. For text input, specify one of the following values: • `cohere` • `googleai` • `huggingface` • `ocigenai` • `openai` • `vertexai`
`credential_name`	Name of the credential in the form: *`schema.credential_name`* A credential name holds authentication credentials to enable access to your provider for making REST API calls. You need to first set up your credential by calling the `DBMS_VECTOR.CREATE_CREDENTIAL` helper function to create and store a credential, and then refer to the credential name here. See CREATE_CREDENTIAL.
`url`	URL of the third-party provider endpoint for each REST call, as listed in Supported Third-Party Provider Operations and Endpoints.

Table 216-13 (Cont.) Third-Party Provider Parameter Details

Parameter	Description
`model`	Name of the third-party embedding model in the form: *`schema.model_name`* If you do not specify a schema, then the schema of the procedure invoker is used. **Note**: • For Generative AI, all the supported third-party models are listed in Supported Third-Party Provider Operations and Endpoints. • For accurate results, ensure that the chosen text embedding model matches the vocabulary file used for chunking. If you are not using a vocabulary file, then ensure that the input length is defined within the token limits of your model. • To get image embeddings, you can use any image embedding model or multimodal embedding model supported by Vertex AI. Multimodal embedding is a technique that vectorizes data from different modalities such as text and images. When using a multimodal embedding model to generate embeddings, ensure that you use the same model to vectorize both types of content (text and images). By doing so, the resulting embeddings are compatible and situated in the same vector space, which allows for effective comparison between the two modalities during similarity searches.
`transfer_timeout`	Maximum time to wait for the request to complete. The default value is `60` seconds. You can increase this value for busy web servers.
`batch_size`	Maximum number of vectors to request at a time. For example, for a batch size of `50`, if `100` chunks are passed, then this API sends two requests with an array of `50` strings each. If `30` chunks are passed (which is lesser than the defined batch size), then the API sends those in a single request. For REST calls, it is more efficient to send a batch of inputs at a time rather than requesting a single input per call. Increasing the batch size can provide better performance, whereas reducing the batch size may reduce memory and data usage, especially if your provider has a rate limit. The default or maximum allowed value depends on the third-party provider settings.
`max_count`	Maximum number of times the API can be called for a given third-party provider. When set to an integer *n*, `max_count` stops the execution of the API for the given provider beyond *n* times. This prevents accidental calling of a third-party over some limit, for example to avoid surpassing the service amount that you have purchased.

Additional third-party provider parameters:

Optionally, specify additional provider-specific parameters.

Table 216-14 Additional REST Provider Parameter Details

Parameter	Description
`input_type`	Type of input to vectorize.

Let us look at some example configurations for all third-party providers:

Important:

- The following examples are for illustration purposes. For accurate and up-to-date information on the parameters to use, refer to your third-party provider's documentation.
- For a list of all supported REST endpoint URLs, see Supported Third-Party Provider Operations and Endpoints.
- The generated embedding results may be different between requests for the same input and configuration, depending on your embedding model or floating point precision. However, this does not affect your queries (and provides semantically correct results) because the vector distance will be similar.

Cohere example:

```
{
  "provider"       : "cohere",
  "credential_name": "COHERE_CRED",
  "url"            : "https://api.cohere.example.com/embed",
  "model"          : "embed-english-light-v2.0",
  "input_type"     : "search_query"
}
```

Generative AI example:

```
{
  "provider"       : "ocigenai",
  "credential_name": "OCI_CRED",
  "url"            : "https://generativeai.oci.example.com/embedText",
  "model"          : "cohere.embed-english-v3.0",
  "batch_size"     : 10
}
```

Google AI example:

```
{
  "provider"       : "googleai",
  "credential_name": "GOOGLEAI_CRED",
  "url"            : "https://googleapis.example.com/models/",
  "model"          : "embedding-001",
  "max_count"      : 500
}
```

Hugging Face example:

```
{
  "provider"       : "huggingface",
  "credential_name": "HF_CRED",
  "url"            : "https://api.huggingface.example.com/",
  "model"          : "sentence-transformers/all-MiniLM-L6-v2"
}
```

Ollama example:

```
{
  "provider"       : "ollama",
  "host"           : "local",
  "url"            : "http://localhost:11434/api/embeddings",
  "model"          : "phi3:mini"
}
```

OpenAI example:

```
{
  "provider"       : "openai",
  "credential_name": "OPENAI_CRED",
  "url"            : "https://api.openai.example.com/embeddings",
  "model"          : "text-embedding-3-small"
}
```

Vertex AI example:

```
{
  "provider"       : "vertexai",
  "credential_name": "VERTEXAI_CRED",
  "url"            : "https://googleapis.example.com/models/",
  "model"          : "textembedding-gecko:predict"
}
```

Examples

You can use `UTL_TO_EMBEDDING` in a `SELECT` clause and `UTL_TO_EMBEDDINGS` in a `FROM` clause, as follows:

UTL_TO_EMBEDDING:

- **Text to vector using Generative AI**:

  The following examples use `UTL_TO_EMBEDDING` to generate an embedding with `Hello world` as the input.

  Here, the cohere.embed-english-v3.0 model is used by accessing Generative AI as the provider. You can replace the `model` value with any other supported model that you want to use with Generative AI, as listed in Supported Third-Party Provider Operations and Endpoints.

  ```
  -- declare embedding parameters

  var params clob;

  begin
   :params := '
  {
    "provider": "ocigenai",
    "credential_name": "OCI_CRED",
    "url": "https://inference.generativeai.us-chicago-1.oci.oraclecloud.com/
  20231130/actions/embedText",
    "model": "cohere.embed-english-v3.0",
  ```

```
  "batch_size": 10
}';
end;
/

-- get text embedding: PL/SQL example

declare
  input clob;
  v vector;
begin
  input := 'Hello world';

  v := dbms_vector.utl_to_embedding(input, json(params));
  dbms_output.put_line(vector_serialize(v));
exception
  when OTHERS THEN
    DBMS_OUTPUT.PUT_LINE (SQLERRM);
    DBMS_OUTPUT.PUT_LINE (SQLCODE);
end;
/

-- get text embedding: select example

select dbms_vector.utl_to_embedding('Hello world', json(:params)) from
dual;
```

- **Image to vector using Vertex AI**:

  The following examples use `UTL_TO_EMBEDDING` to generate an embedding by accessing the Vertex AI's multimodal embedding model.

  Here, the input is `parrots.jpg`, `VEC_DUMP` is a local directory that stores the `parrots.jpg` file, and the modality is specified as `image`.

```
-- declare embedding parameters

var params clob;

begin
  :params := '
{
  "provider": "vertexai",
  "credential_name": "VERTEXAI_CRED",
  "url": "https://LOCATION-aiplatform.googleapis.com/v1/projects/PROJECT/
locations/LOCATION/publishers/google/models/",
  "model": "multimodalembedding:predict"
}';
end;
/

-- get image embedding: PL/SQL example

declare
  v vector;
  output clob;
begin
```

```
  v := dbms_vector.utl_to_embedding(
    to_blob(bfilename('VEC_DUMP', 'parrots.jpg')), 'image', json(:params));
  output := vector_serialize(v);
  dbms_output.put_line('vector data=' || dbms_lob.substr(output, 100) ||
'...');
end;
/

-- get image embedding: select example

select dbms_vector.utl_to_embedding(
  to_blob(bfilename('VEC_DUMP', 'parrots.jpg')), 'image', json(:params));
```

- **Text to vector using in-database embedding model**:

  The following example uses UTL_TO_EMBEDDING to generate a vector embedding by calling an ONNX format embedding model (doc_model) loaded into Oracle Database.

  Here, the provider is database, and the input is hello.

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

  select dbms_vector.utl_to_embedding('hello', json(:params)) from dual;
  ```

  For complete example, see Convert Text String to Embedding Within Oracle Database.

- **End-to-end examples**:

  To run various end-to-end example scenarios using UTL_TO_EMBEDDING, see Generate Embedding.

UTL_TO_EMBEDDINGS:

- **Text to vector using in-database embedding model**:

  The following example uses UTL_TO_EMBEDDINGS to generate an array of embeddings by calling an ONNX format embedding model (doc_model) loaded into Oracle Database.

  Here, the provider is database, and the input is a PDF document stored in the documentation_tab table. As you can see, you first use UTL_TO_CHUNKS to split the data into smaller chunks before passing in to UTL_TO_EMBEDDINGS.

  ```
  CREATE TABLE doc_chunks as
  (select dt.id doc_id, et.embed_id, et.embed_data,
  to_vector(et.embed_vector) embed_vector
   from
     documentation_tab dt,
     dbms_vector.utl_to_embeddings(
         dbms_vector.utl_to_chunks(dbms_vector.utl_to_text(dt.data),
  json('{"normalize":"all"}')),
         json('{"provider":"database", "model":"doc_model"}')) t,
     JSON_TABLE(t.column_value, '$[*]' COLUMNS (embed_id NUMBER PATH
  '$.embed_id', embed_data VARCHAR2(4000) PATH '$.embed_data', embed_vector
  CLOB PATH '$.embed_vector')) et
  );
  ```

For complete example, see SQL Quick Start Using a Vector Embedding Model Uploaded into the Database.

- **End-to-end examples**:

  To run various end-to-end example scenarios using `UTL_TO_EMBEDDINGS`, see Perform Chunking With Embedding.

UTL_TO_GENERATE_TEXT

Use the `DBMS_VECTOR.UTL_TO_GENERATE_TEXT` chainable utility function to generate a text response for a given prompt or an image, by accessing third-party text generation models.

Purpose

To communicate with Large Language Models (LLMs) through natural language conversations. You can generate a textual answer, description, or summary for prompts and images, given as input to LLM-powered chat interfaces.

- Prompt to Text:

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

  For this operation, this API makes a REST call to your chosen remote third-party provider (Cohere, Generative AI, Google AI, Hugging Face, OpenAI, or Vertex AI) or local third-party provider (Ollama).

- Image to Text:

  You can also prompt with a media file, such as an image, to extract text from pictures or photos. You supply a text question as the prompt (such as "`What is this image about?`" or "`How many birds are there in this painting?`") along with the image. The LLM responds with a textual analysis or description of the contents of the image.

  For this operation, this API makes a REST call to your chosen remote third-party provider (Google AI, Hugging Face, OpenAI, or Vertex AI) or local third-party provider (Ollama).

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

Syntax

This function accepts the input as CLOB containing text data (for textual prompts) or as BLOB containing media data (for media files such as images). It then processes this information to generate a new CLOB containing the generated text.

- Prompt to Text:

```
DBMS_VECTOR.UTL_TO_GENERATE_TEXT (
        DATA           IN CLOB,
        PARAMS         IN JSON default NULL
) return CLOB;
```

- Image to Text:

```
DBMS_VECTOR.UTL_TO_GENERATE_TEXT(
          TEXT_DATA      IN CLOB,
          MEDIA_DATA     IN BLOB,
          MEDIA_TYPE     IN VARCHAR2 default 'image/jpeg',
          PARAMS         IN JSON default NULL
) return CLOB;
```

DATA and TEXT_DATA

Specify the textual prompt as CLOB for the DATA or TEXT_DATA clause.

> **Note:**
>
> Hugging Face uses an image captioning model that does not require a prompt, when giving an image as input. If you input a prompt along with an image, then the prompt will be ignored.

MEDIA_DATA

Specify the BLOB file, such as an image or a visual PDF file.

MEDIA_TYPE

Specify the image format for the given image or visual PDF file (BLOB file) in one of the supported image data MIME types. For example:

- For PNG: image/png
- For JPEG: image/jpeg
- For PDF: application/pdf

> **Note:**
>
> For a complete list of the supported image formats, refer to your third-party provider's documentation.

PARAMS

Specify the following input parameters in JSON format, depending on the service provider that you want to access for text generation:

```
{
  "provider"         : "<AI service provider>",
  "credential_name"  : "<credential name>",
  "url"              : "<REST endpoint URL for text generation service>",
  "model"            : "<text generation model name>",
  "transfer_timeout" : <maximum wait time for the request to complete>,
  "max_count": "<maximum calls to the AI service provider>",
  "<additional REST provider parameter>": "<REST provider parameter value>"
}
```

Table 216-15 UTL_TO_GENERATE_TEXT Parameter Details

Parameter	Description
`provider`	Supported REST provider that you want to access to generate text. Specify one of the following values: For `CLOB` input: • `cohere` • `googleai` • `huggingface` • `ocigenai` • `openai` • `vertexai` For `BLOB` input: • `googleai` • `huggingface` • `openai` • `vertexai`
`credential_name`	Name of the credential in the form: *`schema.credential_name`* A credential name holds authentication credentials to enable access to your provider for making REST API calls. You need to first set up your credential by calling the `DBMS_VECTOR.CREATE_CREDENTIAL` helper function to create and store a credential, and then refer to the credential name here. See CREATE_CREDENTIAL.
`url`	URL of the third-party provider endpoint for each REST call, as listed in Supported Third-Party Provider Operations and Endpoints.
`model`	Name of the third-party text generation model in the form: *`schema.model_name`* If the model name is not schema-qualified, then the schema of the procedure invoker is used. **Note**: For Generative AI, all the supported third-party models are listed in Supported Third-Party Provider Operations and Endpoints.
`transfer_timeout`	Maximum time to wait for the request to complete. The default value is `60` seconds. You can increase this value for busy web servers.

Table 216-15 (Cont.) UTL_TO_GENERATE_TEXT Parameter Details

Parameter	Description
max_count	Maximum number of times the API can be called for a given third-party provider. When set to an integer *n*, max_count stops the execution of the API for the given provider beyond *n* times. This prevents accidental calling of a third-party over some limit, for example to avoid surpassing the service amount that you have purchased.

Additional third-party provider parameters:

Optionally, specify additional provider-specific parameters.

Table 216-16 Additional REST Provider Parameter Details

Parameter	Description
max_tokens	Maximum number of tokens in the output text.
temperature	Degree of randomness used when generating the output text, in the range of 0.0-5.0. To generate the same output for a prompt, use 0. To generate a random new text for that prompt, increase the temperature. **Note**: Start with the temperature set to 0. If you do not require random results, a recommended temperature value is between 0 and 1. A higher value is not recommended because a high temperature may produce creative text, which might also include hallucinations.
topP	Probability of tokens in the output, in the range of 0.0–1.0. A lower value provides less random responses and a higher value provides more random responses.
candidateCount	Number of response variations to return, in the range of 1-4.
maxOutputTokens	Maximum number of tokens to generate for each response.

Let us look at some example configurations for all third-party providers:

Important:

- The following examples are for illustration purposes. For accurate and up-to-date information on additional parameters to use, refer to your third-party provider's documentation.
- For a list of all supported REST endpoint URLs, see Supported Third-Party Provider Operations and Endpoints.

Cohere example:

```
{
  "provider"       : "cohere",
  "credential_name": "COHERE_CRED",
  "url"            : "https://api.cohere.example.com/chat",
  "model"          : "command"
}
```

Generative AI example:

> **Note:**
>
> For Generative AI, if you want to pass any additional REST provider-specific parameters, then you must enclose those in `chatRequest`.

```
{
  "provider"       : "ocigenai",
  "credential_name": "OCI_CRED",
  "url"            : "https://inference.generativeai.us-example.com/chat",
  "model"          : "cohere.command-r-16k",
  "chatRequest"    : {
                      "maxTokens"    : 256
                     }
}
```

Google AI example:

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

Hugging Face example:

```
{
  "provider"        : "huggingface",
  "credential_name" : "HF_CRED",
  "url"             : "https://api.huggingface.example.com/models/",
  "model"           : "gpt2"
}
```

Ollama example:

```
{
  "provider"       : "ollama",
  "host"           : "local",
  "url"            : "http://localhost:11434/api/generate",
  "model"          : "phi3:mini"
}
```

OpenAI example:

```
{
  "provider"        : "openai",
  "credential_name" : "OPENAI_CRED",
  "url"             : "https://api.openai.example.com",
  "model"           : "gpt-4o-mini",
  "max_tokens"      : 60,
  "temperature"     : 1.0
}
```

Vertex AI example:

```
{
  "provider"         : "vertexai",
  "credential_name"  : "VERTEXAI_CRED",
  "url"              : "https://googleapis.example.com/models/",
  "model"            : "gemini-1.0-pro:generateContent",
  "generation_config": {
                        "temperature"    : 0.9,
                        "topP"           : 1,
                        "candidateCount" : 1,
                        "maxOutputTokens": 256
                       }
}
```

Examples

- Prompt to Text:

  The following statements generate a text response by making a REST call to Generative AI. The prompt given here is "`What is Oracle Text?`".

  Here, the cohere.command-r-16k and meta.llama-3.1-70b-instruct models are used. You can replace the `model` value with any other supported model that you want to use with Generative AI, as listed in Supported Third-Party Provider Operations and Endpoints.

  Using the cohere.command-r-16k model:

```
-- select example

var params clob;
exec :params := '
{
  "provider"       : "ocigenai",
  "credential_name": "OCI_CRED",
  "url"            : "https://inference.generativeai.us-
chicago-1.oci.oraclecloud.com/20231130/actions/chat",
  "model"          : "cohere.command-r-16k",
  "chatRequest"    : {
                      "maxTokens": 256
                     }
}';

select dbms_vector.utl_to_generate_text(
 'What is Oracle Text?',
 json(:params)) from dual;

-- PL/SQL example

declare
  input clob;
  params clob;
  output clob;
begin
  input := 'What is Oracle Text?';

  params := '
{
  "provider"       : "ocigenai",
  "credential_name": "OCI_CRED",
  "url"            : "https://inference.generativeai.us-
chicago-1.oci.oraclecloud.com/20231130/actions/chat",
  "model"          : "cohere.command-r-16k",
  "chatRequest"    : {
                      "maxTokens": 256
                     }
}';

  output := dbms_vector.utl_to_generate_text(input, json(params));
  dbms_output.put_line(output);
  if output is not null then
    dbms_lob.freetemporary(output);
  end if;
exception
  when OTHERS THEN
    DBMS_OUTPUT.PUT_LINE (SQLERRM);
    DBMS_OUTPUT.PUT_LINE (SQLCODE);
end;
/
```

Using the meta.llama-3.1-70b-instruct model:

```
-- select example

var params clob;
exec :params := '
{
   "provider"       : "ocigenai",
   "credential_name": "OCI_CRED",
   "url"            : "https://inference.generativeai.us-
chicago-1.oci.oraclecloud.com/20231130/actions/chat",
   "model"          : "meta.llama-3.1-70b-instruct",
   "chatRequest"    : {
                       "topK" : 1
                      }
}';

select dbms_vector.utl_to_generate_text(
 'What is Oracle Text?',
 json(:params)) from dual;
```

```
-- PL/SQL example

declare
  input clob;
  params clob;
  output clob;
begin
  input := 'What is Oracle Text?';

  params := '
{
   "provider"       : "ocigenai",
   "credential_name": "OCI_CRED",
   "url"            : "https://inference.generativeai.us-
chicago-1.oci.oraclecloud.com/20231130/actions/chat",
   "model"          : "meta.llama-3.1-70b-instruct",
   "chatRequest"    : {
                       "topK" : 1
                      }
}';

  output := dbms_vector.utl_to_generate_text(input, json(params));
  dbms_output.put_line(output);
  if output is not null then
    dbms_lob.freetemporary(output);
  end if;
exception
  when OTHERS THEN
    DBMS_OUTPUT.PUT_LINE (SQLERRM);
    DBMS_OUTPUT.PUT_LINE (SQLCODE);
end;
/
```

End-to-end examples:

To run end-to-end example scenarios, see Generate Text Response.

- Image to Text:

  The following statements generate a text response by making a REST call to OpenAI. Here, the input is an image (`sample_image.jpeg`) along with the prompt "`Describe this image?`".

```
-- select example

var input clob;
var media_data blob;
var media_type clob;
var params clob;

begin
  :input := 'Describe this image';
  :media_data := load_blob_from_file('DEMO_DIR', 'sample_image.jpeg');
  :media_type := 'image/jpeg';
  :params := '
{
  "provider"       : "openai",
  "credential_name": "OPENAI_CRED",
  "url"            : "https://api.openai.com/v1/chat/completions",
  "model"          : "gpt-4o-mini",
  "max_tokens"     : 60
}';
end;
/

select dbms_vector.utl_to_generate_text(:input, :media_data, :media_type,
json(:params));

-- PL/SQL example

declare
  input clob;
  media_data blob;
  media_type varchar2(32);
  params clob;
  output clob;

begin
  input := 'Describe this image';
  media_data := load_blob_from_file('DEMO_DIR', 'image_file');
  media_type := 'image/jpeg';
  params := '
{
  "provider"       : "openai",
  "credential_name": "OPENAI_CRED",
  "url"            : "https://api.openai.com/v1/chat/completions",
  "model"          : "gpt-4o-mini",
  "max_tokens"     : 60
}';

  output := dbms_vector.utl_to_generate_text(
    input, media_data, media_type, json(params));
  dbms_output.put_line(output);

  if output is not null then
    dbms_lob.freetemporary(output);
  end if;
  if media_data is not null then
    dbms_lob.freetemporary(media_data);
  end if;
exception
  when OTHERS THEN
    DBMS_OUTPUT.PUT_LINE (SQLERRM);
    DBMS_OUTPUT.PUT_LINE (SQLCODE);
end;
/
```

End-to-end examples:

To run end-to-end example scenarios, see Describe Image Content.