217
DBMS_VECTOR_CHAIN

The DBMS_VECTOR_CHAIN package provides APIs to support advanced operations with Oracle AI Vector Search, such as chunking and embedding data along with text generation and summarization capabilities. It is more suitable for text processing with similarity search and hybrid search, using functionality that can be pipelined together for an end-to-end search.

These functions accept their respective input parameters in JSON format.

Related Topics

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

Summary of DBMS_VECTOR_CHAIN Subprograms

This table lists the DBMS_VECTOR_CHAIN subprograms and briefly describes them.

Table 217-1 ***DBMS_VECTOR_CHAIN Package Subprograms***

Subprogram	Description
Chainable Utility (UTL) Functions:	
These functions are a set of modular and flexible functions within vector utility PL/SQL packages. You can chain these together to automate end-to-end data transformation and similarity search operations.	
UTL_TO_TEXT	Extracts plain text data from documents
UTL_TO_CHUNKS	Splits data into smaller pieces or chunks
UTL_TO_EMBEDDING and UTL_TO_EMBEDDINGS	Converts data to one or more vector embeddings
UTL_TO_SUMMARY	Extracts a summary from documents
UTL_TO_GENERATE_TEXT	Generates text for a prompt (input string) or an image
Credential Helper Procedures:	
These procedures enable you to securely manage authentication credentials in the database. You require these credentials to enable access to third-party service providers for making REST calls.	
CREATE_CREDENTIAL	Creates a credential name
DROP_CREDENTIAL	Drops an existing credential name
Preference Helper Procedures:	
These procedures enable you to manage vectorizer preferences, to be used with the CREATE_HYBRID_VECTOR_INDEX and ALTER_INDEX SQL statements when creating or managing hybrid vector indexes.	
CREATE_PREFERENCE	Creates a vectorizer preference
DROP_PREFERENCE	Drops an existing vectorizer preference
Chunker Helper Procedures:	
These procedures enable you to configure vocabulary and language data (abbreviations), to be used with the VECTOR_CHUNKS SQL function or UTL_TO_CHUNKS PL/SQL function.	
CREATE_VOCABULARY	Loads your token vocabulary file into the database
DROP_VOCABULARY	Removes existing vocabulary data

Table 217-1 (Cont.) *DBMS_VECTOR_CHAIN Package Subprograms*

Subprogram	Description
CREATE_LANG_DATA	Loads your language data file into the database
DROP_LANG_DATA	Removes existing abbreviation data
Data Access Function:	
This function enables you to enhance search operations.	
RERANK	Reorders search results for more relevant output

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

CREATE_CREDENTIAL

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

Purpose

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

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

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

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

Syntax

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

CREDENTIAL_NAME

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

PARAMS

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

Generative AI requires the following authentication parameters:

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

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

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

Table 217-2 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 217-2 (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_chain.create_credential(
    credential_name   => 'OCI_CRED',
    params            => json(jo.to_string));
end;
/
```

- For Cohere:

```
declare
  jo json_object_t;
begin
  jo := json_object_t();
  jo.put('access_token', 'A1Aa0abA1AB1a1Abc123ab1A123ab123AbcA12a');
  dbms_vector_chain.create_credential(
```

```
        credential_name   => 'COHERE_CRED',
        params            => json(jo.to_string));
    end;
    /
```

End-to-end examples:

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

CREATE_LANG_DATA

Use the `DBMS_VECTOR_CHAIN.CREATE_LANG_DATA` chunker helper procedure to load your own language data file into the database.

Purpose

To create custom language data for your chosen language (specified using the `language` chunking parameter).

A language data file contains language-specific abbreviation tokens. You can supply this data to the chunker to help in accurately determining sentence boundaries of chunks, by using knowledge of the input language's end-of-sentence (EOS) punctuations, abbreviations, and contextual rules.

Usage Notes

- All supported languages are distributed with the default language-specific abbreviation dictionaries. You can create a language data based on the abbreviation tokens loaded in the `schema.table.column`, using a user-specified language data name (`PREFERENCE_NAME`).
- After loading your language data, you can use language-specific chunking by specifying the `language` chunking parameter with `VECTOR_CHUNKS` or `UTL_TO_CHUNKS`.
- You can query these data dictionary views to access existing language data:
  - `ALL_VECTOR_LANG` displays all available languages data.
  - `USER_VECTOR_LANG` displays languages data from the schema of the current user.
  - `ALL_VECTOR_ABBREV_TOKENS` displays abbreviation tokens from all available language data.
  - `USER_VECTOR_ABBREV_TOKENS` displays abbreviation tokens from the language data owned by the current user.

Syntax

```
DBMS_VECTOR_CHAIN.CREATE_LANG_DATA (
    PARAMS       IN JSON default NULL
);
```

PARAMS

Specify the input parameters in JSON format:

```
{
    table_name,
    column_name,
```

```
    language,
    preference_name
}
```

Table 217-3 Parameter Details

Parameter	Description	Required	Default Value
table_name	Name of the table (along with the optional table owner) in which you want to load the language data	Yes	No value
column_name	Column name in the language data table in which you want to load the language data	Yes	No value
language	Any supported language name, as listed in Supported Languages and Data File Locations	Yes	No value
preference_name	User-specified preference name for this language data	Yes	No value

Example

```
declare
    params CLOB := '{"table_name"      : "eos_data_1",
                     "column_name"     : "token",
                     "language"        : "indonesian",
                     "preference_name" : "my_lang_1"}';
begin
    DBMS_VECTOR_CHAIN.CREATE_LANG_DATA(
        JSON (params));
end;
/
```

End-to-end example:

To run an end-to-end example scenario using this procedure, see Create and Use Custom Language Data.

Related Topics

- VECTOR_CHUNKS
- UTL_TO_CHUNKS
- Text Processing Views

CREATE_PREFERENCE

Use the DBMS_VECTOR_CHAIN.CREATE_PREFERENCE helper procedure to create a vectorizer preference, to be used when creating or updating hybrid vector indexes.

Purpose

To create a vectorizer preference.

This allows you to customize vector search parameters of a hybrid vector indexing pipeline. The goal of a vectorizer preference is to provide you with a straightforward way to configure

how to chunk or embed your documents, without requiring a deep understanding of various chunking or embedding strategies.

Usage Notes

A **vectorizer** preference is a JSON object that collectively holds user-specified values related to the following chunking, embedding, or vector index creation parameters:

- Chunking (`UTL_TO_CHUNKS` and `VECTOR_CHUNKS`)
- Embedding (`UTL_TO_EMBEDDING`, `UTL_TO_EMBEDDINGS`, and `VECTOR_EMBEDDING`)
- Vector index creation (`distance`, `accuracy`, and `vector_idxtype`)

All vector index preferences follow the same JSON syntax as defined for their corresponding `DBMS_VECTOR` and `DBMS_VECTOR_CHAIN` APIs.

After creating a vectorizer preference, you can use the `VECTORIZER` parameter to pass this preference name in the `paramstring` of the `PARAMETERS` clause for `CREATE_HYBRID_VECTOR_INDEX` and `ALTER_INDEX` SQL statements.

Creating a preference is optional. If you do not specify any optional preference, then the index is created with system defaults.

Syntax

```
DBMS_VECTOR_CHAIN.CREATE_PREFERENCE (
    PREF_NAME      IN VARCHAR2,
    PREF_TYPE      IN VARCHAR2,
    PARAMS         IN JSON default NULL
);
```

PREF_NAME

Specify the name of the vectorizer preference to create.

PREF_TYPE

Type of preference. The only supported preference type is:

```
DBMS_VECTOR_CHAIN.VECTORIZER
```

PARAMS

Specify vector search-specific parameters in JSON format:

- Embedding Parameter
- Chunking Parameters
- Vector Index Parameters
- Paths Parameter

Embedding Parameter:

```
{ "model" : <embedding_model_for_vector_generation> }
```

For example:

```
{ "model" : MY_INDB_MODEL }
```

`model` specifies the name under which your ONNX embedding model is stored in the database.

If you do not have an in-database embedding model in ONNX format, then perform the steps listed in *Oracle Database AI Vector Search User's Guide*.

Chunking Parameters:

```
{
    "by"           :  mode,
    "max"          :  max,
    "overlap"      :  overlap,
    "split"        :  split_condition,
    "vocabulary"   :  vocabulary_name,
    "language"     :  nls_language,
    "normalize"    :  normalize_mode,
    "extended"     :  boolean
}
```

For example:

```
JSON(
 '{ "by"           :    "vocabulary",
    "max"          :    "100",
    "overlap"      :    "0",
    "split"        :    "none",
    "vocabulary"   :    "myvocab",
    "language"     :    "american",
    "normalize"    :    "all"
  }')
```

If you specify `split` as `custom` and `normalize` as `options`, then you must additionally specify the `custom_list` and `norm_options` parameters, respectively:

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

The following table describes all the chunking parameters:

Parameter	Description and Acceptable Values
by	Specify a mode for splitting your data, that is, to split by counting the number of characters, words, or vocabulary tokens. **Valid values**: • characters (or chars): Splits by counting the number of characters. • words: Splits by counting the number of words. Words are defined as sequences of alphabetic characters, sequences of digits, individual punctuation marks, or symbols. For segmented languages without 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)

Vector Index Parameters:

```
{
  "distance"        :  <vector_distance>,
  "accuracy"        :  <vector_accuracy>,
```

```
  "vector_idxtype"   :  <vector_idxtype>
}
```

For example:

```
{
  "distance"         :  COSINE,
  "accuracy"         :  95,
  "vector_idxtype"   :  HNSW
}
```

Parameter	Description
distance	Distance metric or mathematical function used to compute the distance between vectors: • COSINE • MANHATTAN • DOT • EUCLIDEAN • L2_SQUARED • EUCLIDEAN_SQUARED **Note**: Currently, the HAMMING and JACCARD vector distance metrics are not supported with hybrid vector indexes. For detailed information on each of these metrics, see Vector Distance Functions and Operators. **Default value**: COSINE

Parameter	Description
accuracy	Target accuracy at which the approximate search should be performed when running an approximate search query using vector indexes. As explained in Understand Approximate Similarity Search Using Vector Indexes, you can specify non-default target accuracy values either by specifying a percentage value or by specifying internal parameters values, depending on the index type you are using. • For a Hierarchical Navigable Small World (HNSW) approximate search: In the case of an HNSW approximate search, you can specify a target accuracy percentage value to influence the number of candidates considered to probe the search. This is automatically calculated by the algorithm. A value of 100 will tend to impose a similar result as an exact search, although the system may still use the index and will not perform an exact search. The optimizer may choose to still use an index as it may be faster to do so given the predicates in the query. Instead of specifying a target accuracy percentage value, you can specify the EFSEARCH parameter to impose a certain maximum number of candidates to be considered while probing the index. The higher that number, the higher the accuracy. For detailed information, see Understand Hierarchical Navigable Small World Indexes. • For an Inverted File Flat (IVF) approximate search: In the case of an IVF approximate search, you can specify a target accuracy percentage value to influence the number of partitions used to probe the search. This is automatically calculated by the algorithm. A value of 100 will tend to impose an exact search, although the system may still use the index and will not perform an exact search. The optimizer may choose to still use an index as it may be faster to do so given the predicates in the query. Instead of specifying a target accuracy percentage value, you can specify the NEIGHBOR PARTITION PROBES parameter to impose a certain maximum number of partitions to be probed by the search. The higher that number, the higher the accuracy. For detailed information, see Understand Inverted File Flat Vector Indexes. Valid range for both HNSW and IVF vector indexes is: `> 0 and <= 100` **Default value**: None
vector_idxtype	Type of vector index to create: • HNSW for an HNSW vector index • IVF for an IVF vector index For detailed information on each of these index types, see Manage the Different Categories of Vector Indexes. **Default value**: IVF

Paths Parameter:

This field allows specification of an array of path objects. There can be many path objects and each path object must specify a type and a path_list.

> **Note:**
>
> If the user does not specify the `paths` field, the whole document would be considered.

```
"paths":[
        {"type"      : "<path_type>",
         "path_list" : ["<path_list_array>"]
        }
        ]
```

Let us consider a sample JSON document:

```
 {
      "person":
       {
         "bio": "James is a data scientist who specializes in natural
language .. ",
         "profile":
          {
              "text" : "James is a data scientist with expertise in Python
programming...",
              "embedding" :
[1.60541728E-001,5.76677322E-002,4.0473938E-003,1.2037459E-001,-5.98970801E-00
4, ..]
          },
         "avatar": "https://example.com/images/James.jpg"
       },
      "product":
      {
        "description": "A social media analytics tool.", "It helps brands
track...",
        "image": "https://example.com/images/data_tool.jpg",
        "embedding" :
[1.60541728E-001,5.76677322E-002,4.0473938E-003,1.2037459E-001,-5.98970801E-00
4, ..]
      }
 }
```

And a `path_list` corresponding to the above JSON is provided here:

```
"paths": [
         {"type"       : "VECTOR",
          "path_list" : ["$.person.profile.embedding", "$.product.embedding"]
         },
         {"type"       : "STRING",
          "path_list" : ["$.person.bio", "$.product.description"]
         }
         ]
```

The following table describes the details of `paths` parameter:

Parameter	Accepted Values
type	The possible values for this field are: • STRING - for fields that need to be converted to vectors. • VECTOR - for fields that are already vectors.
path_list	Accepts an array of paths with at least one path in valid JSON format - ($.a.b.c.d). **Note:** For the VECTOR option, Oracle currently accepts one vector array per path.

Example

```
begin
  DBMS_VECTOR_CHAIN.CREATE_PREFERENCE(
    'my_vec_spec',
     DBMS_VECTOR_CHAIN.VECTORIZER,
     json('{ "vector_idxtype" :  "hnsw",
             "model"          :  "my_doc_model",
             "by"             :  "words",
             "max"            :  100,
             "overlap"        :  10,
             "split"          :  "recursively,
             "language"       :  "english",
             "paths":          : [
                                  {
                                   "type"      : "VECTOR",
                                   "path_list" : ["$.person.profile.embedding"]
                                   }
                                 ]
              }'));
end;
/

CREATE HYBRID VECTOR INDEX my_hybrid_idx on
    doc_table(text_column)
    parameters('VECTORIZER my_vec_spec');
```

Related Topics

- CREATE HYBRID VECTOR INDEX

CREATE_VOCABULARY

Use the DBMS_VECTOR_CHAIN.CREATE_VOCABULARY chunker helper procedure to load your own token vocabulary file into the database.

Purpose

To create custom token vocabulary that is recognized by the tokenizer used by your vector embedding model.

A vocabulary contains a set of tokens (words and word pieces) that are collected during a model's statistical training process. You can supply this data to the chunker to help in accurately selecting the text size that approximates the maximum input limit imposed by the tokenizer of your embedding model.

Usage Notes

- Usually, the supported vocabulary files (containing recognized tokens) are included as part of a model's distribution. Oracle recommends to use the vocabulary files associated with your model.

  If a vocabulary file is not available, then you may download one of the following files depending on the tokenizer type:

  - **WordPiece**:

    Vocabulary file (`vocab.txt`) for the "bert-base-uncased" (English) or "bert-base-multilingual-cased" model

  - **Byte-Pair Encoding (BPE)**:

    Vocabulary file (`vocab.json`) for the "GPT2" model

    Use the following python script to extract the file:

    ```
    import json
    import sys

    with open(sys.argv[1], encoding="utf-8") as f:
      d = json.load(f)
      for term in d:
        print(term)
    ```

  - **SentencePiece**:

    Vocabulary file (`tokenizer.json`) for the "xlm-roberta-base" model

    Use the following python script to extract the file:

    ```
    import json
    import sys

    with open(sys.argv[1], encoding="utf-8") as f:
      d = json.load(f)
      for entry in d["model"]["vocab"]:
        print(entry[0])
    ```

  Ensure to save your vocabulary files in `UTF-8` encoding.

- You can create a vocabulary based on the tokens loaded in the `schema.table.column`, using a user-specified vocabulary name (`vocabulary_name`).

  After loading your vocabulary data, you can use the `by vocabulary` chunking mode (with `VECTOR_CHUNKS` or `UTL_TO_CHUNKS`) to split input data by counting the number of tokens.

- You can query these data dictionary views to access existing vocabulary data:

  - `ALL_VECTOR_VOCAB` displays all available vocabularies.
  - `USER_VECTOR_VOCAB` displays vocabularies from the schema of the current user.
  - `ALL_VECTOR_VOCAB_TOKENS` displays a list of tokens from all available vocabularies.

- `USER_VECTOR_VOCAB_TOKENS` displays a list of tokens from the vocabularies owned by the current user.

Syntax

```
DBMS_VECTOR_CHAIN.CREATE_VOCABULARY(
    PARAMS      IN JSON default NULL
);
```

PARAMS

Specify the input parameters in JSON format:

```
{
    table_name,
    column_name,
    vocabulary_name,
    format,
    cased
}
```

Table 217-4 Parameter Details

Parameter	Description	Required	Default Value
`table_name`	Name of the table (along with the optional table owner) in which you want to load the vocabulary file	Yes	No value
`column_name`	Column name in the vocabulary table in which you want to load the vocabulary file	Yes	No value
`vocabulary_name`	User-specified name of the vocabulary, along with the optional owner name (if other than the current owner)	Yes	No value
`format`	• `xlm` for SentencePiece tokenization • `bert` for WordPiece tokenization • `gpt2` for BPE tokenization	Yes	No value
`cased`	Character-casing of the vocabulary, that is, vocabulary to be treated as cased or uncased	No	`false`

Example

```
DECLARE
  params clob := '{"table_name"       : "doc_vocabtab",
                   "column_name"      : "token",
                   "vocabulary_name"  : "doc_vocab",
                   "format"           : "bert",
                   "cased"            : false}';

BEGIN
  dbms_vector_chain.create_vocabulary(json(params));
END;
/
```

End-to-end example:

To run an end-to-end example scenario using this procedure, see Create and Use Custom Vocabulary.

Related Topics

- VECTOR_CHUNKS
- UTL_TO_CHUNKS
- Text Processing Views

DROP_CREDENTIAL

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

Syntax

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

CREDENTIAL_NAME

Specify the credential name that you want to drop.

Examples

- For Generative AI:

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

- For Cohere:

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

DROP_LANG_DATA

Use the `DBMS_VECTOR_CHAIN.DROP_LANG_DATA` chunker helper procedure to remove abbreviation data from the data dictionary.

Syntax

```
DBMS_VECTOR_CHAIN.DROP_LANG_DATA(
    PREF_NAME     IN VARCHAR2
);
```

LANG

Specify the name of the language data that you want to drop for a given language.

Example

```
DBMS_VECTOR_CHAIN.DROP_LANG_DATA('indonesian');
```

DROP_PREFERENCE

Use the `DBMS_VECTOR_CHAIN.DROP_PREFERENCE` preference helper procedure to remove an existing Vectorizer preference.

Syntax

```
DBMS_VECTOR_CHAIN.DROP_PREFERENCE (PREF_NAME);
```

PREF_NAME

Name of the Vectorizer preference to drop.

Example

```
DBMS_VECTOR_CHAIN.DROP_PREFERENCE ('scott_vectorizer');
```

DROP_VOCABULARY

Use the `DBMS_VECTOR_CHAIN.DROP_VOCABULARY` chunker helper procedure to remove vocabulary data from the data dictionary.

Syntax

```
DBMS_VECTOR_CHAIN.DROP_VOCABULARY(
    VOCABULARY_NAME    IN VARCHAR2
);
```

VOCAB_NAME

Specify the name of the vocabulary that you want to drop, in the form:

vocabulary_name

or

owner.vocabulary_name

Example

```
DBMS_VECTOR_CHAIN.DROP_VOCABULARY('MY_VOCAB_1');
```

RERANK

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

Purpose

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

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

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

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

Syntax

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

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

For example, a reranked output includes:

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

Where,

- `index` specifies the position of the document in the list of input text.

- `score` specifies the relevance score.
- `content` specifies the input text corresponding to the index.

QUERY

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

DOCUMENTS

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

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

PARAMS

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

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

Table 217-5 RERANK Parameter Details

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

Additional REST provider parameters:

Optionally, specify additional provider-specific parameters for reranking.

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

Cohere example:

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

Vertex AI example:

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

Table 217-6 Additional REST Provider Parameter Details

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

Table 217-6 (Cont.) Additional REST Provider Parameter Details

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

Examples

- Using Cohere:

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

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

- Using Vertex AI:

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

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

End-to-end example:

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

UTL_TO_CHUNKS

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

Purpose

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

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

Syntax

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

DATA

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

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

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

For example:

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

Where,

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

PARAMS

Specify input parameters in JSON format.

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

For example:

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

Here is a complete description of these parameters:

Parameter	Description and Acceptable Values
by	Specify a mode for splitting your data, that is, to split by counting the number of characters, words, or vocabulary tokens. **Valid values**: • characters (or chars): Splits by counting the number of characters. • words: Splits by counting the number of words. Words are defined as sequences of alphabetic characters, sequences of digits, individual punctuation marks, or symbols. For segmented languages without 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_chain.utl_to_chunks(D.text,
   JSON('{ "by"       : "words",
           "max"      : "100",
           "overlap"  : "0",
           "split"    : "recursively",
           "language" : "american",
           "normalize": "all" }')) C;
```

End-to-end examples:

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

Related Topics

- VECTOR_CHUNKS

UTL_TO_EMBEDDING and UTL_TO_EMBEDDINGS

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

Purpose

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

- Text to Vector:

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

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

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

  - Third-party embedding model:

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

- Image to Vector:

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

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

Syntax

- Text to Vector:

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


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

- Image to Vector:

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

DATA

- Text to Vector:

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

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

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

- Image to Vector:

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

A generated embedding output includes:

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

Where,

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

MODALITY

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

PARAMS

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

If using Oracle Database as the provider:

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

Table 217-7 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 217-8 Third-Party Provider Parameter Details

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

Table 217-8 (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 217-9 Additional REST Provider Parameter Details

Parameter	Description
`input_type`	Type of input to vectorize.

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

Important:

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

Cohere example:

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

Generative AI example:

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

Google AI example:

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

Hugging Face example:

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

Ollama example:

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

OpenAI example:

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

Vertex AI example:

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

Examples

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

UTL_TO_EMBEDDING:

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

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

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

```
-- declare embedding parameters

var params clob;

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

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

-- get text embedding: PL/SQL example

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

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

-- get text embedding: select example

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

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

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

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

```
-- declare embedding parameters

var params clob;

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

-- get image embedding: PL/SQL example

declare
  v vector;
  output clob;
begin
```

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

-- get image embedding: select example

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

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

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

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

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

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

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

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

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

UTL_TO_EMBEDDINGS:

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

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

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

```
CREATE TABLE doc_chunks as
(select dt.id doc_id, et.embed_id, et.embed_data,
to_vector(et.embed_vector) embed_vector
 from
   documentation_tab dt,
   dbms_vector_chain.utl_to_embeddings(

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

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

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

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

UTL_TO_GENERATE_TEXT

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

Purpose

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

- Prompt to Text:

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

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

- Image to Text:

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

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

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

Syntax

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

- Prompt to Text:

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

- Image to Text:

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

DATA and TEXT_DATA

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

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

MEDIA_DATA

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

MEDIA_TYPE

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

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

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

PARAMS

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

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

Table 217-10 UTL_TO_GENERATE_TEXT Parameter Details

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

Table 217-10 (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 217-11 Additional REST Provider Parameter Details

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

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

Important:

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

Cohere example:

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

```
  "model"           : "command"
}
```

Generative AI example:

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

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

Google AI example:

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

Hugging Face example:

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

Ollama example:

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

OpenAI example:

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

Vertex AI example:

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

Examples

- Prompt to Text:

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

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

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

  ```
  -- select example

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

  select dbms_vector_chain.utl_to_generate_text(
   'What is Oracle Text?',
  ```

```
 json(:params)) from dual;

-- PL/SQL example

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

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

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

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

```
-- select example

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

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

```
-- PL/SQL example

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

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

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

End-to-end examples:

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

- Image to Text:

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

```
-- select example

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

begin
  :input := 'Describe this image';
  :media_data := load_blob_from_file('DEMO_DIR', 'sample_image.jpeg');
  :media_type := 'image/jpeg';
  :params := '
{
```

```
  "provider"       : "openai",
  "credential_name": "OPENAI_CRED",
  "url"            : "https://api.openai.com/v1/chat/completions",
  "model"          : "gpt-4o-mini",
  "max_tokens"     : 60
}';
end;
/

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

-- PL/SQL example

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

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

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

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

End-to-end examples:

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

UTL_TO_TEXT

Use the `DBMS_VECTOR_CHAIN.UTL_TO_TEXT` chainable utility function to convert an input document (for example, PDF, DOC, JSON, XML, or HTML) to plain text.

Purpose

To perform a file-to-text transformation by using the Oracle Text component (`CONTEXT`) of Oracle Database.

Syntax

```
DBMS_VECTOR_CHAIN.UTL_TO_TEXT (
    DATA           IN CLOB | BLOB,
    PARAMS         IN JSON default NULL
) return CLOB;
```

DATA

This function accepts the input data type as `CLOB` or `BLOB`. It can read documents from a remote location or from files stored locally in the database tables.

It returns a plain text version of the document as `CLOB`.

Oracle Text supports around 150 file types. For a complete list of all the supported document formats, see *Oracle Text Reference*.

PARAMS

Specify the following input parameter in JSON format:

```
{
    "plaintext" : "true or false",
    "charset"   : "UTF8"
}
```

Table 217-12 Parameter Details

Parameter	Description
`plaintext`	Plain text output. The default value for this parameter is `true`, that is, by default the output format is plain text. If you do not want to return the document as plain text, then set this parameter to `false`.
`charset`	Character set encoding. Currently, only `UTF8` is supported.

Example

```
select DBMS_VECTOR_CHAIN.UTL_TO_TEXT (
    t.blobdata,
     json('{
            "plaintext": "true",
```

```
            "charset"  : "UTF8"
           }')
) from tab t;
```

End-to-end example:

To run an end-to-end example scenario using this function, see Convert File to Text to Chunks to Embeddings Within Oracle Database.

UTL_TO_SUMMARY

Use the `DBMS_VECTOR_CHAIN.UTL_TO_SUMMARY` chainable utility function to generate a summary for textual documents.

A summary is a short and concise extract with key features of a document that best represents what the document is about as a whole. A summary can be free-form paragraphs or bullet points based on the format that you specify.

Purpose

To perform a text-to-summary transformation by accessing either Oracle Database or a third-party service provider:

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

  Uses the in-house implementation with Oracle Database, where Oracle Text is internally used to extract a summary (gist) from your document using the Oracle Text PL/SQL procedure `CTX_DOC.GIST`.

- Third-party summarization model:

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

> **Note:**
>
> Currently, `UTL_TO_SUMMARY` does not work for Generative AI because the model and summary endpoint supported for Generative AI have been retired. It will be available in a subsequent release.

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

Syntax

```
DBMS_VECTOR_CHAIN.UTL_TO_SUMMARY (
    DATA          IN CLOB,
    PARAMS        IN JSON default NULL
) return CLOB;
```

DATA

This function accepts the input data type in plain text as `CLOB`.

It returns a summary of the input document also as `CLOB`.

PARAMS

Specify summary parameters in JSON format, depending on the service provider that you want to use for document summarization.

If using Oracle Database as the provider:

```
{
  "provider"     : "database",
  "glevel"       : "<summary format>",
  "numParagraphs": <number in the range 1-16>,
  "maxPercent"   : <number in the range 1-100>,
  "num_themes"   : <number in the range 1-50>,
  "language"     : "<name of the language>"
}
```

Table 217-13 Database Provider Parameter Details

Parameter	Description
`provider`	Specify `database` (default setting) to access Oracle Database as the provider. Leverages the document `gist` or summary generated by Oracle Text.
`glevel`	Format to display the summary: • `SENTENCE` | `S`: As a list of sentences • `PARAGRAPH` | `P`: In a free-form paragraph
`numParagraphs`	Maximum number of document paragraphs (or sentences) selected for the summary. The default value is `16`. The `numParagraphs` parameter is used only when this parameter yields a smaller summary size than the summary size yielded by the `maxPercent` parameter, because the function always returns the smallest size summary.
`maxPercent`	Maximum number of document paragraphs (or sentences) selected for the summary, as a percentage of the total paragraphs (or sentences) in the document. The default value is `10`. The `maxPercent` parameter is used only when this parameter yields a smaller summary size than the summary size yielded by the `numParagraphs` parameter, because the function always returns the smallest size summary.

Table 217-13 (Cont.) Database Provider Parameter Details

Parameter	Description
num_themes	Number of theme summaries to produce. For example, if you specify 10, then this function returns the top 10 theme summaries. If you specify 0 or NULL, then this function returns all themes in a document. The default value is 50. If the document contains more than 50 themes, only the top 50 themes show conceptual hierarchy.
language	Language name of your summary text, as listed in Supported Languages and Data File Locations.

For example:

```
{
  "provider"          : "database",
  "glevel"            : "sentence",
  "numParagraphs"     : 1
}
```

If using a third-party provider:

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

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

Table 217-14 Third-Party Provider Parameter Details

Parameter	Description
provider	Third-party service provider that you want to access to get the summary. A REST call is made to the specified provider to access its text summarization model. Specify one of the following values: • cohere • googleai • huggingface • ocigenai • openai • vertexai

Table 217-14 (Cont.) Third-Party Provider 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_CHAIN.CREATE_CREDENTIAL` helper function to create and store a credential, and then refer to the credential name here. See CREATE_CREDENTIAL.
`url`	URL of the third-party provider endpoint for each REST call, as listed in Supported Third-Party Provider Operations and Endpoints.
`model`	Name of the third-party text summarization model in the form: *`schema.model_name`* If the model name is not schema-qualified, then the schema of the procedure invoker is used. **Note**: For Generative AI, you must specify *`schema.model_name`*. All the third-party models supported for Generative AI are listed in Supported Third-Party Provider Operations and Endpoints.
`transfer_timeout`	Maximum time to wait for the request to complete. The default value is `60` seconds. You can increase this value for busy web servers.
`max_count`	Maximum number of times the API can be called for a given third-party provider. When set to an integer *n*, `max_count` stops the execution of the API for the given provider beyond *n* times. This prevents accidental calling of a third-party over some limit, for example to avoid surpassing the service amount that you have purchased.

Additional third-party provider parameters:

Optionally, specify additional provider-specific parameters.

Table 217-15 Additional REST Provider Parameter Details

Parameter	Description
`length`	Approximate length of the summary text: • `SHORT`: Roughly up to 2 sentences • `MEDIUM`: Between 3 and 5 sentences • `LONG`: 6 or more sentences • `AUTO`: The model chooses a length based on the input size **Note**: For Generative AI, you must enter this value in uppercase.
`format`	Format to display the summary: • `PARAGRAPH`: In a free-form paragraph • `BULLETS`: In bullet points **Note**: For Generative AI, you must enter this value in uppercase.

Table 217-15 (Cont.) Additional REST Provider Parameter Details

Parameter	Description
`temperature`	Degree of randomness used when generating output text, in the range of `0.0-5.0`. To generate the same output for a prompt, use `0`. To generate a random new text for that prompt, increase the temperature. Default temperature is `1` and the maximum temperature is `5`. **Note**: To summarize a text, start with the temperature set to `0`. If you do not require random results, a recommended temperature value is `0.2` for Generative AI and between `0` and `1` for Cohere. Use a higher value if for example you plan to perform a selection of the various summaries afterward. Do not use a high temperature for summarization because a high temperature encourages the model to produce creative text, which might also include hallucinations.
`extractiveness`	How much to reuse the input in the summary: • `LOW`: Summaries with low extractiveness tend to paraphrase. • `HIGH`: Summaries with high extractiveness lean toward reusing sentences verbatim. **Note**: For Generative AI, you must enter this value in uppercase.
`max_tokens`	Maximum number of tokens in the output text.
`topP`	Probability of tokens in the output, in the range of `0.0-1.0`. A lower value provides less random responses and a higher value provides more random responses.
`candidateCount`	Number of response variations to return, in the range of `1-4`.
`maxOutputTokens`	Maximum number of tokens to generate for each response.

> **Note:**
>
> When specifying the `length`, `format`, and `extractiveness` parameters for Generative AI, ensure to enter the values in uppercase letters.

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

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

Cohere example:

```
{
  "provider"          : "cohere",
  "credential_name"   : "COHERE_CRED",
```

```
  "url"               : "https://api.cohere.example.com/summarize",
  "model"             : "command",
  "length"            : "medium",
  "format"            : "paragraph",
  "temperature"       : 1.0
}
```

Generative AI example:

```
{
 "provider"           : "ocigenai",
 "credential_name"    : "OCI_CRED",
 "url"                : "https://generativeai.oci.example.com/summarizeText",
 "model"              : "cohere.command-r-16k",
 "length"             : "MEDIUM",
 "format"             : "PARAGRAPH"
}
```

Google AI example:

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

Hugging Face example:

```
{
  "provider"          : "huggingface",
  "credential_name"   : "HF_CRED",
  "url"               : "https://api.huggingface.example.co/models/",
  "model"             : "facebook/bart-large-cnn"
}
```

Ollama example:

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

OpenAI example:

```
{
  "provider"          : "openai",
  "credential_name"   : "OPENAI_CRED",
  "url"               : "https://api.openai.example.com",
  "model"             : "gpt-4o-mini",
  "max_tokens"        : 256,
  "temperature"       : 1.0
}
```

Vertex AI example:

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

Examples

- **Generate summary using Oracle Database**:

  This statement specifies `database` as the provider. Here, the Oracle Text PL/SQL procedure `CTX_DOC.GIST` is internally called to generate a summary of an extract on "Transactions".

```
-- select example

set serveroutput on

var params clob;
begin
  :params := '
{
  "provider": "database",
  "glevel": "sentence",
  "numParagraphs": 1
}';
end;
/

select dbms_vector_chain.utl_to_summary(
 'A transaction is a logical, atomic unit of work that contains one or
more SQL statements.  An RDBMS must be able to group SQL statements so
that they are either all committed, which means they are applied to the
database, or all rolled back, which means they are undone.  An
illustration of the need for transactions is a funds transfer from a
```

```
savings account to a checking account. The transfer consists of the
following separate operations:
    1. Decrease the savings account.
    2. Increase the checking account.
    3. Record the transaction in the transaction journal.
    Oracle Database guarantees that all three operations succeed or fail
as a unit. For example, if a hardware failure prevents a statement in the
transaction from executing, then the other statements must be rolled back.
    Transactions set Oracle Database apart from a file system. If you
perform an atomic operation that updates several files, and if the system
fails halfway through, then the files will not be consistent. In contrast,
a transaction moves an Oracle database from one consistent state to
another. The basic principle of a transaction is "all or nothing": an
atomic operation succeeds or fails as a whole.',
 json(:params)) from dual;

-- PL/SQL example

declare
  input clob;
  params clob;
  output clob;
begin
  input := 'A transaction is a logical, atomic unit of work that contains
one or more SQL statements.  An RDBMS must be able to group SQL statements
so that they are either all committed, which means they are applied to the
database, or all rolled back, which means they are undone.  An
illustration of the need for transactions is a funds transfer from a
savings account to a checking account. The transfer consists of the
following separate operations:
    1. Decrease the savings account.
    2. Increase the checking account.
    3. Record the transaction in the transaction journal.
    Oracle Database guarantees that all three operations succeed or fail
as a unit. For example, if a hardware failure prevents a statement in the
transaction from executing, then the other statements must be rolled back.
    Transactions set Oracle Database apart from a file system. If you
perform an atomic operation that updates several files, and if the system
fails halfway through, then the files will not be consistent. In contrast,
a transaction moves an Oracle database from one consistent state to
another. The basic principle of a transaction is "all or nothing": an
atomic operation succeeds or fails as a whole.';

  params := '
  {
   "provider": "database",
   "glevel": "sentence",
   "numParagraphs": 1
  }';

  output := dbms_vector_chain.utl_to_summary(input, json(params));
  dbms_output.put_line(output);
  if output is not null then
    dbms_lob.freetemporary(output);
  end if;
exception
```

```
  when OTHERS THEN
    DBMS_OUTPUT.PUT_LINE (SQLERRM);
    DBMS_OUTPUT.PUT_LINE (SQLCODE);
end;
/
```

- **Generate summary using Generative AI**:

  These statements generate a summary of an extract on "Transactions" by accessing Generative AI as the provider.

  Here, the cohere.command-r-16k model is used for the summarization operation. You can replace the `model` value with any other supported model that you want to use with Generative AI, as listed in Supported Third-Party Provider Operations and Endpoints.

```
-- select example

var params clob;
begin
  :params := '
{
 "provider": "ocigenai",
 "credential_name": "OCI_CRED",
 "url": "https://inference.generativeai.us-chicago-1.oci.oraclecloud.com/
20231130/actions/chat",
 "model": "cohere.command-r-16k",
 "temperature": "0.0",
 "extractiveness": "LOW"
}';
end;
/

select dbms_vector_chain.utl_to_summary(
 'A transaction is a logical, atomic unit of work that contains one or
more SQL statements.  An RDBMS must be able to group SQL statements so
that they are either all committed, which means they are applied to the
database, or all rolled back, which means they are undone.  An
illustration of the need for transactions is a funds transfer from a
savings account to a checking account. The transfer consists of the
following separate operations:
    1. Decrease the savings account.
    2. Increase the checking account.
    3. Record the transaction in the transaction journal.
    Oracle Database guarantees that all three operations succeed or fail
as a unit. For example, if a hardware failure prevents a statement in the
transaction from executing, then the other statements must be rolled back.
    Transactions set Oracle Database apart from a file system. If you
perform an atomic operation that updates several files, and if the system
fails halfway through, then the files will not be consistent. In contrast,
a transaction moves an Oracle database from one consistent state to
another. The basic principle of a transaction is "all or nothing": an
atomic operation succeeds or fails as a whole.',
 json(:params)) from dual;

-- PL/SQL example

declare
```

```
  input clob;
  params clob;
  output clob;
begin
  input := 'A transaction is a logical, atomic unit of work that contains
one or more SQL statements.  An RDBMS must be able to group SQL statements
so that they are either all committed, which means they are applied to the
database, or all rolled back, which means they are undone.  An
illustration of the need for transactions is a funds transfer from a
savings account to a checking account. The transfer consists of the
following separate operations:
    1. Decrease the savings account.
    2. Increase the checking account.
    3. Record the transaction in the transaction journal.
    Oracle Database guarantees that all three operations succeed or fail
as a unit. For example, if a hardware failure prevents a statement in the
transaction from executing, then the other statements must be rolled back.
    Transactions set Oracle Database apart from a file system. If you
perform an atomic operation that updates several files, and if the system
fails halfway through, then the files will not be consistent. In contrast,
a transaction moves an Oracle database from one consistent state to
another. The basic principle of a transaction is "all or nothing": an
atomic operation succeeds or fails as a whole.';

  params := '
  {
   "provider": "ocigenai",
   "credential_name": "OCI_CRED",
   "url": "https://inference.generativeai.us-chicago-1.oci.oraclecloud.com/
20231130/actions/chat",
   "model": "cohere.command-r-16k",
   "length": "MEDIUM",
   "format": "PARAGRAPH",
   "temperature": 1.0
  }';

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

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

  To run end-to-end example scenarios using this function, see Generate Summary.

Supported Languages and Data File Locations

These are the supported languages for which language data files are distributed by default in the specified directories.

Language Name	Abbreviation	Data File
AFRIKAANS	af	ctx/data/eos/dreosaf.txt
AMERICAN	us	ctx/data/eos/dreosus.txt
ARABIC	ar	ctx/data/eos/dreosar.txt
BASQUE	eu	ctx/data/eos/dreoseu.txt
BELARUSIAN	be	ctx/data/eos/dreosbe.txt
BRAZILIAN PORTUGUESE	ptb	ctx/data/eos/dreosptb.txt
BULGARIAN	bg	ctx/data/eos/dreosbg.txt
CANADIAN FRENCH	frc	ctx/data/eos/dreosfrc.txt
CATALAN	ca	ctx/data/eos/dreosca.txt
CROATIAN	hr	ctx/data/eos/dreoshr.txt
CYRILLIC SERBIAN	csr	ctx/data/eos/dreoscsr.txt
CZECH	cs	ctx/data/eos/dreoscs.txt
DANISH	dk	ctx/data/eos/dreosdk.txt
DARI	prs	ctx/data/eos/dreosprs.txt
DUTCH	nl	ctx/data/eos/dreosnl.txt
EGYPTIAN	eg	ctx/data/eos/dreoseg.txt
ENGLISH	gb	ctx/data/eos/dreosgb.txt
ESTONIAN	et	ctx/data/eos/dreoset.txt
FINNISH	sf	ctx/data/eos/dreossf.txt
FRENCH	f	ctx/data/eos/dreosf.txt
GALICIAN	ga	ctx/data/eos/dreosga.txt
GERMAN	d	ctx/data/eos/dreosd.txt
GERMAN DIN	din	ctx/data/eos/dreosdin.txt
GREEK	el	ctx/data/eos/dreosel.txt
HEBREW	iw	ctx/data/eos/dreosiw.txt
HINDI	hi	ctx/data/eos/dreoshi.txt
HUNGARIAN	hu	ctx/data/eos/dreoshu.txt
ICELANDIC	is	ctx/data/eos/dreosis.txt
INDONESIAN	in	ctx/data/eos/dreosin.txt
ITALIAN	i	ctx/data/eos/dreosi.txt
JAPANESE	ja	ctx/data/eos/dreosja.txt
KOREAN	ko	ctx/data/eos/dreosko.txt
LATIN AMERICAN SPANISH	esa	ctx/data/eos/dreosesa.txt
LATIN BOSNIAN	lbs	ctx/data/eos/dreoslbs.txt
LATIN SERBIAN	lsr	ctx/data/eos/dreoslsr.txt
LATVIAN	lv	ctx/data/eos/dreoslv.txt
LITHUANIAN	lt	ctx/data/eos/dreoslt.txt

Language Name	Abbreviation	Data File
MACEDONIAN	mk	ctx/data/eos/dreosmk.txt
MALAY	ms	ctx/data/eos/dreosms.txt
MEXICAN SPANISH	esm	ctx/data/eos/dreosesm.txt
NORWEGIAN	n	ctx/data/eos/dreosn.txt
NYNORSK	nn	ctx/data/eos/dreosnn.txt
PERSIAN	fa	ctx/data/eos/dreosfa.txt
POLISH	pl	ctx/data/eos/dreospl.txt
PORTUGUESE	pt	ctx/data/eos/dreospt.txt
ROMANIAN	ro	ctx/data/eos/dreosro.txt
RUSSIAN	ru	ctx/data/eos/dreosru.txt
SIMPLIFIED CHINESE	zhs	ctx/data/eos/dreoszhs.txt
SLOVAK	sk	ctx/data/eos/dreossk.txt
SLOVENIAN	sl	ctx/data/eos/dreossl.txt
SPANISH	e	ctx/data/eos/dreose.txt
SWEDISH	s	ctx/data/eos/dreoss.txt
THAI	th	ctx/data/eos/dreosth.txt
TRADITIONAL CHINESE	zht	ctx/data/eos/dreoszht.txt
TURKISH	tr	ctx/data/eos/dreostr.txt
UKRAINIAN	uk	ctx/data/eos/dreosuk.txt
URDU	ur	ctx/data/eos/dreosur.txt

Related Topics

- CREATE_LANG_DATA
  Use the `DBMS_VECTOR_CHAIN.CREATE_LANG_DATA` chunker helper procedure to load your own language data file into the database.