102
DBMS_HYBRID_VECTOR

The DBMS_HYBRID_VECTOR package contains a JSON-based SEARCH API that lets you search by *keywords* and *vectors* against hybrid vector indexes. By integrating traditional keyword-based text search with vector-based similarity search, you can improve the overall search experience and provide users with more accurate information.

Related Topics

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

SEARCH

Use the DBMS_HYBRID_VECTOR.SEARCH PL/SQL function to run textual queries, vector similarity queries, or hybrid queries against hybrid vector indexes.

Purpose

To search by vectors and keywords. This function lets you perform the following tasks:

- Facilitate a combined (hybrid) query of textual documents and vectorized chunks:

  You can query a hybrid vector index in multiple vector and keyword search combinations called search modes, as described in Understand Hybrid Search. This API accepts a JSON specification for all query parameters.

- Fuse and reorder the search results:

  The search results of a hybrid query are fused into a unified result set as CLOB using the specified fusion set operator, and reordered by a combined score using the specified scoring algorithm.

- Run a default query for a simplified search experience:

  The minimum input parameters required are the hybrid_index_name and search_text. The same text string is used to query against a vectorized chunk index and a text document index.

Syntax

```
DBMS_HYBRID_VECTOR.SEARCH(
   json(
     '{  "hybrid_index_name"     :
"<hybrid_vector_index_name>",
         "search_text"           :  "<query string for keyword-and-semantic
search>",
         "search_fusion"         :  one of these values : "INTERSECT | UNION
| TEXT_ONLY | VECTOR_ONLY | MINUS_TEXT |
                                         MINUS_VECTOR | RERANK",
         "search_scorer"         :  one of these values : "RRF | RSF",
         "vector":
           {
            "search_text"             :  "<query string for semantic
search>",
            "search_vector"           :  "<vector_embedding>",
            "search_mode"             :  one of thse values : "DOCUMENT |
CHUNK",
            "aggregator"              :  one of these values : "COUNT | SUM |
MIN | MAX | AVG | MEDIAN | BONUSMAX | WINAVG |
                                         ADJBOOST | MAXAVGMED",
            "result_max"              :  <maximum number of vector results>,
            "score_weight"            :  <weight of vector score for RSF>,
            "rank_penalty"            :  <penalty of vector ranking for RRF>,
            "inpath"                  :  <an array of valid JSON paths>,
            "accuracy"                :  <target accuracy for semantic
search>,
            "index_probes"            :  <neighbor partitions for semantic
search>,
            "index_efsearch"          :  <efsearch for semantic search>,
            "filter_type"             :  one of these values : "IN_WO | IN_W
| PRE_WO | PRE_W | POST_WO | DEFAULT"
           },
         "text":
           {
            "contains"                :  "<query string for keyword
search>",
            "search_text"             :  "<alternative text to use to
construct a contains query automatically>",
            "json_textcontains"       :  <an array of valid JSON path and a
query string>,
            "score_weight"            :  <weight of text score for RSF>,
            "rank_penalty"            :  <penalty of text ranking for RRF>,
            "result_max:              :  <maximum number of document results>,
            "inpath"                  :  <array of valid JSON paths>
           },
         "filter_by":
           {
            "op"                      :  one of these values: "< | > | <= |
>= | = | != | ^- | <> | LIKE | LIKEC | LIKE2 | LIKE4 |
                                         REGEXP_LIKE | BETWEEN | EXISTS |
INSTR | INSTRC | INSTR2 | INSTR4 | STSTR | STSTR2 | STSTR4 |
                                         STSTRB | STSTRC | <ANY | >ANY |
<=ANY | >=ANY | =ANY | !=ANY | <SOME | >SOME |
                                         <=SOME | >=SOME | =SOME | !=SOME |
<ALL | >ALL | <=ALL | >=ALL | =ALL | !=ALL | IN |
                                         AND | OR | NOT | NOTOR |
NOTAND",
            "type"                    :  one of these values : "number |
string | date | timestamp",
            "col"                     :  "<base table column name>",
            "path"                    :  "<JSON path dot notation within a
base table JSON column>",
            "func"                    :  one of these values : "ABS | FLOOR |
LENGTH | CEILING | UPPER | LOWER | TO_BOOLEAN |
                                         TO_DATE | TO_DOUBLE |
TO_BINARYDOUBLE | TO_NUMBER | TO_CHAR | TO_TIMESTAMP",
            "args"                    :  <an array of arguments to the
operator>
           }
         "return":
           {
            "topN"                    :
<topN_value>,
            "values"                  :  one or more of these values : "rowid
| score | vector_score | text_score | vector_rank |
                                         text_rank | chunk_text | chunk_id |
paths",
            "format"                  :  one of these values : "JSON |
XML"
           }
     }'
  )
)
```

> **Note:**
>
> This API supports two constructs of search. One where you specify a single `search_text` field for both semantic search and keyword search (default setting). Another where you specify separate `search_text` and `contains` query fields using `vector` and `text` sub-elements for semantic search and keyword search, respectively. You cannot use both of these search constructs in one query.

hybrid_index_name

Specify the name of the hybrid vector index to use.

For information on how to create a hybrid vector index if not already created, see Manage Hybrid Vector Indexes.

search_text

Specify a search text string (your query input) for both semantic search and keyword search.

The same text string is used for a keyword query on document text index (by converting the `search_text` into a `CONTAINS ACCUM` operator syntax) and a semantic query on vectorized chunk index (by vectorizing or embedding the `search_text` for a `VECTOR_DISTANCE` search).

For example:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json('{ "hybrid_index_name" : "my_hybrid_idx",
            "search_text"       : "C, Python"
          }'))
FROM DUAL;
```

search_fusion

Specify a fusion sort operator to define what you want to retain from the combined set of keyword-and-semantic search results.

Note:

This search fusion operation is applicable only to non-pure hybrid search cases. Vector-only and text-only searches do not fuse any results.

Parameter	Description
INTERSECT	Returns only the rows that are common to both text search results and vector search results. Score condition: `text_score > 0 AND vector_score > 0`
UNION (default)	Combines all distinct rows from both text search results and vector search results. Score condition: `text_score > 0 OR vector_score > 0`
TEXT_ONLY	Returns all distinct rows from text search results plus the ones that are common to both text search results and vector search results. Thus, the fused results contain the text search results that appear in text search, including those that appear in both. Score condition: `text_score > 0`
VECTOR_ONLY	Returns all distinct rows from vector search results plus the ones that are common to both text search results and vector search results. Thus, the fused results contain the vector search results that appear in vector search, including those that appear in both. Score condition: `vector_score > 0`
MINUS_TEXT	Returns all distinct rows from vector search results minus the ones that are common to both text search results and vector search results. Thus, the fused results contain the vector search results that appear in vector search, excluding those that appear in both. Score condition: `text_score = 0`
MINUS_VECTOR	Returns all distinct rows from text search results minus the ones that are common to both text search results and vector search results. Thus, the fused results contain the text search results that appear in text search, excluding those that appear in both. Score condition: `vector_score = 0`
RERANK	Returns all distinct rows from text search ordered by the aggregated vector score of their respective vectors. There is no score condition for this field since the text search is followed by the use of the aggregated document vector scores.

For example:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json('{ "hybrid_index_name"     : "my_hybrid_idx",
            "search_fusion"         : "UNION",
            "vector":
                    { "search_text" : "leadership experience" },
             "text":
                    { "contains"    : "C and Python" }
          }'))
FROM DUAL;
```

search_scorer

Specify a method to evaluate the combined "fusion" search scores from both keyword and semantic search results.

- `RSF` (default) to use the Relative Score Fusion (RSF) algorithm
- `RRF` to use the Reciprocal Rank Fusion (RRF) algorithm

For a deeper understanding of how these algorithms work in hybrid search modes, see Understand Hybrid Search.

For example:

With a single search text string for hybrid search:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "search_text"       : "C, Python",
         "search_scorer"     : "rsf"
      }'))
FROM DUAL;
```

With separate vector and text search strings:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "search_scorer"     : "rsf",
         "vector":
          { "search_text"    : "leadership experience" },
         "text":
          { "contains"       : "C and Python" }
      }'))
FROM DUAL;
```

vector

Specify query parameters for semantic search against the vector index part of your hybrid vector index:

- `search_text`: Search text string (query text).

  This string is converted into a query vector (embedding), and is used in a `VECTOR_DISTANCE` query to search against the vectorized chunk index.

  For example:

  ```
  SELECT DBMS_HYBRID_VECTOR.SEARCH(
      json('{ "hybrid_index_name" : "my_hybrid_idx",
              "vector":
                      { "search_text" : "C, Python" }
            }'))
  FROM DUAL;
  ```

- `search_vector`: Vector embedding (query vector).

  This embedding is directly used in a `VECTOR_DISTANCE` query to search against the vectorized chunk index.

Note:

search_vector is an alternative to the above mentioned search_text when the semantic query is already available as a vector. The vector embedding that you pass here must be generated using the same embedding model used for semantic search by the specified hybrid vector index.

For example:

```
SELECT JSON_SERIALIZE(
         DBMS_HYBRID_VECTOR.SEARCH(
            json_object( 'hybrid_index_name' value 'my_hybrid_idx',
                 'vector' value json_object( 'search_vector' value
vector_serialize(
                                            vector_embedding(doc_model
                                                        using
                                                        'C, Python,
Database'
                                                        as data)
                                                    RETURNING CLOB)
                                             RETURNING JSON)
                  RETURNING JSON))
           RETURNING CLOB PRETTY)
FROM dual;
```

- search_mode: Document or chunk search mode in which you want to query the hybrid vector index:

Parameter	Description
DOCUMENT (default)	Returns document-level results. In document mode, the result of your search is a list of document IDs from the base table corresponding to the list of best documents identified.
CHUNK	Returns chunk-level results. In chunk mode, the result of your search is a list of chunk identifiers and associated document IDs from the base table corresponding to the list of best chunks identified, regardless of whether the chunks come from the same document or different documents. The content from these chunk texts can be used as input for LLMs to formulate responses.

For example, semantic search in chunk mode:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "vector":
          {
             "search_text"   : "leadership experience",
             "search_mode"   : "CHUNK"
          }
      }'))
FROM DUAL;
```

- aggregator: Aggregate function to apply for ranking the vector scores for each document in DOCUMENT SEARCH_MODE.

Parameter	Description
MAX (default)	Standard database aggregate function that selects the top chunk score as the result score.
AVG	Standard database aggregate function that sums the chunk scores and divides by the count.
MEDIAN	Standard database aggregate function that computes the middle value or an interpolated value of the sorted scores.
BONUSMAX	This function combines the maximum chunk score with the remainder multiplied by the average score of the other top scores.
WINAVG	This function computes the maximum average of the rolling window (of size windowSize) of chunk scores.
ADJBOOST	This function computes the average "boosted" chunk score. The chunk scores are boosted with the BOOSTFACTOR multiplied by average score of the surrounding chunk's scores (if they exist).
MAXAVGMED	This function computes a weighted sum of the MAX, AVGN, and MEDN values.

For example:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "vector":
          {
             "search_text"   : "leadership experience",
             "search_mode"   : "DOCUMENT",
             "aggregator"    : "AVG"
          }
      }'))
FROM DUAL;
```

- result_max: The maximum number of vector results in distance order to fetch (approx) from the vector index.

  Value : Any positive integer greater then 0 (zero)

  Default: If the field is not specified, by default, the maximum is computed based on topN.

  For Example:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "vector":
          {
             "search_text"   : "leadership experience",
             "search_mode"   : "DOCUMENT",
             "aggregator"    : "MAX",
             "score_weight"  : 5,
             "result_max"    : 100
          }
      }'))
FROM DUAL;
```

- score_weight: Relative weight (degree of importance or preference) to assign to the semantic VECTOR_DISTANCE query. This value is used when combining the results of RSF ranking.

Value: Any positive integer greater than `0` (zero)

Default: `10` (implies 10 times more importance to vector query than text query)

For example:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "vector":
          {
             "search_text"   : "leadership experience",
             "search_mode"   : "DOCUMENT",
             "aggregator"    : "MAX",
             "score_weight"  : 5
          }
      }'))
FROM DUAL;
```

- `rank_penalty`: Penalty (denominator in RRF, represented as `1/(rank+penalty)`) to assign to vector query. This can help in balancing the relevance score by reducing the importance of unnecessary or repetitive words in a document. This value is used when combining the results of RRF ranking.

  Value: `0` (zero) or any positive integer

  Default: `1`

  For example:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "search_scorer"     : "rrf",
         "vector":
          {
             "search_text"   : "leadership experience",
             "search_mode"   : "DOCUMENT",
             "aggregator"    : "MAX",
             "score_weight"  : 5,
             "rank_penalty"  : 2
          }
      }'))
FROM DUAL;
```

- `inpath`: Valid JSON paths

  `vector.inpath` uses the vectorizer paths as you have in the document. Providing this parameter will restrict the search to the paths specified in this field. Accepts an array of paths in valid JSON format - (`$.a.b.c.d`).

  The list of paths match against VECTORIZER index path lists to form a query constraint on the vector index search. Simple wild cards on the paths are supported such as $.main.*.

  For example:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "search_scorer"     : "rrf",
         "vector":
          {
             "search_text"   : "leadership experience",
             "search_mode"   : "DOCUMENT",
             "aggregator"    : "MAX",
             "score_weight"  : 5,
             "rank_penalty"  : 2,
             "inpath"   : ["$.person.*", "$.product.*"]
          }
      }'))
FROM DUAL;
```

- `accuracy`: Target accuracy to assign to the semantic `VECTOR_DISTANCE` query.

  Value: Any positive integer between `0` (zero) and `100`.

  Default: `0`(zero). The value `0` indicates that the internal default for vector_distance query will be assigned to the field.

  For example:

  ```
  SELECT DBMS_HYBRID_VECTOR.SEARCH(
      json(
        '{ "hybrid_index_name" : "my_hybrid_idx",
           "vector":
            {
               "search_text"   : "leadership experience",
               "search_mode"   : "DOCUMENT",
               "aggregator"    : "MAX",
               "score_weight"  : 5,
               "rank_penalty"  : 2,
               "inpath"        : ["$.person.*", "$.product.*"],
               "accuracy"      : 95
            }
        }'))
  FROM DUAL;
  ```

- `index_probes`: Number of probes to assign to the semantic `VECTOR_DISTANCE` query.

  Value: Any positive integer greater than `0` (zero).

  Default: `0`(zero). The value `0` indicates that the internal default number of probes will be assigned to the field.

  For example:

  ```
  SELECT DBMS_HYBRID_VECTOR.SEARCH(
      json(
        '{ "hybrid_index_name" : "my_hybrid_idx",
           "vector":
            {
               "search_text"   : "leadership experience",
               "search_mode"   : "DOCUMENT",
               "aggregator"    : "MAX",
               "score_weight"  : 5,
               "rank_penalty"  : 2,
               "inpath"        : ["$.person.*", "$.product.*"],
  ```

```
             "accuracy"      : 95,
             "index_probes"  : 3
          }
      }'))
FROM DUAL;
```

- index_efsearch: efs to assign to the semantic VECTOR_DISTANCE query.

  Value: Any positive integer greater than 0 (zero). The value 0 indicates that the internal default for vector_distance query will be assigned to the field.

  Default: 0(zero)

  For example:

  ```
  SELECT DBMS_HYBRID_VECTOR.SEARCH(
      json(
        '{ "hybrid_index_name"   : "my_hybrid_idx",
           "vector":
            {
               "search_text"      : "leadership experience",
               "search_mode"      : "DOCUMENT",
               "aggregator"       : "MAX",
               "score_weight"     : 5,
               "rank_penalty"     : 2,
               "inpath"           : ["$.person.*", "$.product.*"],
               "accuracy"         : 95,
               "index_probes"     : 3,
               "index_efsearch"   : 500,
            }
        }'))
  FROM DUAL;
  ```

- filter_type: Vector index hint filter type. For more information on optimizer plans, hints and filter types for vector indexes, please refer to Optimizer Plans for Vector Indexes and Vector Index Hints.

  Value: The filter_type field could take one of the following values:

  – PRE_W - Pre-filter with join back. This applies only to HNSW indexes.

  – PRE_WO - Pre-filter without join back. This applies to both HNSW and IVF indexes.

  – IN_W - In-filter with join back. This applies only to HNSW indexes.

  – IN_WO - In-filter without join back. This applies only to HNSW indexes.

  – POST_WO - Post-filter without join back. This applies only to IVF indexes.

  Default: No filter type hint.

  For example:

  ```
  SELECT DBMS_HYBRID_VECTOR.SEARCH(
      json(
        '{ "hybrid_index_name"   : "my_hybrid_idx",
           "vector":
            {
               "search_text"      : "leadership experience",
               "search_mode"      : "DOCUMENT",
               "aggregator"       : "MAX",
  ```

```
             "score_weight"    : 5,
             "rank_penalty"    : 2,
             "inpath"          : ["$.person.*", "$.product.*"],
             "accuracy"        : 95,
             "index_probes"    : 3,
             "index_efsearch"  : 500,
             "filter_type"     : "IN_WO"
          }
      }'))
FROM DUAL;
```

text

Specify query parameters for keyword search against the Oracle Text index part of your hybrid vector index:

- `contains`: Search text string (query text).

  This string is converted into an Oracle Text `CONTAINS` query operator syntax for keyword search.

  You can use `CONTAINS` query operators to specify query expressions for full-text search, such as OR (`|`), AND (`&`), STEM (`$`), MINUS (`-`), and so on. For a complete list of all such operators to use, see *Oracle Text Reference*.

  For example:

  With a text contains string for pure keyword search:

  ```
  SELECT DBMS_HYBRID_VECTOR.SEARCH(
      json('{ "hybrid_index_name" : "my_hybrid_idx",
              "text":
                     { "contains" : "C and Python" }
            }'))
  FROM DUAL;
  ```

  With separate search texts using `vector` and `text` sub-elements for hybrid search. One search text or a vector embedding to run a `VECTOR_DISTANCE` query for semantic search. A second search text to run a `CONTAINS` query for keyword search. This query conducts two separate keyword and semantic queries, where keyword scores and semantic scores are combined:

  ```
  SELECT DBMS_HYBRID_VECTOR.SEARCH(
      json('{ "hybrid_index_name" : "my_hybrid_idx",
              "vector":
                      { "search_text" : "leadership experience" },
               "text":
                      { "contains" : "C and Python" }
            }'))
  FROM DUAL;
  ```

- `search_text`: The alternative search text to use to construct a contains query automatically.

  ```
  SELECT DBMS_HYBRID_VECTOR.SEARCH(
      json('{ "hybrid_index_name" : "my_hybrid_idx",
              "text":
  ```

```
                   { "contains"    : "C and Python",
                     "search_text" : "data science skills"
                   }
          }'))
FROM DUAL;
```

- `json_textcontains`: An alternate JSON expression to use instead of `contains` AND `search_text`.

> **Note:**
>
> It is an error to specify `json_textcontains` **WITH** either `text.contains` or `text.search_text`.

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json('{ "hybrid_index_name" : "my_hybrid_idx",
            "text":
                   { "json_textcontains"    : ["$.person", "$C
and $Python"]
                   }
          }'))
FROM DUAL;
```

- `score_weight`: Relative weight (degree of importance or preference) to assign to the text `CONTAINS` query. This value is used when combining the results of RSF ranking.

  Value: Any positive integer greater than `0` (zero)

  Default: `1` (implies neutral weight)

  For example:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "text":
          {
             "contains"      : "C and Python",
             "score_weight"  : 1
          }
      }'))
FROM DUAL;
```

- `rank_penalty`: Penalty (denominator in RRF, represented as `1/(rank+penalty)`) to assign to keyword query.

  This can help in balancing the relevance score by reducing the importance of unnecessary or repetitive words in a document. This value is used when combining the results of RRF ranking.

  Value: `0` (zero) or any positive integer

  Default: `5`

For example:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "text":
          {
             "contains"       : "C and Python",
             "rank_penalty"   : 5
          }
      }'))
FROM DUAL;
```

- inpath: Valid JSON paths

  Providing this parameter will restrict the search to the paths specified in this field. Accepts an array of paths in valid JSON format - ($.a.b.c.d).

  For example:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "text":
          {
             "contains"       : "C and Python",
             "rank_penalty"   : 5,
             "inpath"    : ["$.person.*","$.product.*"]
          }
      }'))
FROM DUAL;
```

- result_max: The maximum number of document results (ordered by score) to retrieve from the document index. If not provided, the maximum is computed based on the topN.

  For example:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "text":
          {
             "contains"       : "C and Python",
             "rank_penalty"   : 5,
             "inpath"         : ["$.person.*","$.product.*"],
             "result_max"     : 100
          }
      }'))
FROM DUAL;
```

filter_by

To constrain the search results via standard relational logical constraints :

Parameter	Value

`op`	Logical comparison operator. Accepted values - One of these operators : • **Simple comparison operators** : `'<', '>', '<=', '>=', '=', '!=', '^=', '<>', 'LIKE', 'LIKEC', 'LIKE2', 'LIKE4', 'INSTR', 'INSTR2', 'INSTR4','INSTRB', 'INSTRC', 'STSTR', 'STSTR2', 'STSTR4','STSTRB', 'STSTRC', 'REGEXP_LIKE', 'BETWEEN', 'EXISTS'` **Note:** `STSTR` is the only non-standard operator in the list. It stands for “**START STRING**” and is analogous to `INSTR`, but the result must be equal to position 1. • **Group comparison operators** : – 18 combinations of `'<', '>', '<=', '>=', '=', '!='` with `ANY, SOME, ALL` – `IN` • **Logical operators** : `'AND', 'OR', 'NOT', 'NOTAND', 'NOTOR'` **Note:** `"NOTAND" and "NOTOR"` are short-hand for the following expressions, useful in reducing the JSON expression tree. `NOTOR` is NOT (arg1 OR arg2 ...). `NOTAND` is NOT (arg1 AND arg2 ...)
`col`	Base table column name.

	Note: • No `column` is required for logical operators. • Only one of `col` or `path` could be specified in the same element.
`path`	The JSON path dot notation within a base table JSON column. **Note:** • No `path` is required for logical operators. • Only one of `col` or `path` could be specified in the same element. • If the base table has a JSON column called `data`, then the syntax would be "`data.path`" where the path is case-sensitive matching the JSON data schema. For more details, see JSON dot notation.

type	The data type of the column. Accepted types include : number, date, timestamp and string.
func	For the comparison operators, an optional function can be applied to the column value before the comparison. These functions are the standard SQL functions. The one exception is "TO_DOUBLE" is provided as an alias to the full name "TO_BINARY_DOUBLE" Accepted values include : ABS, FLOOR, LENGTH, CEILING, UPPER, LOWER, TO_BOOLEAN, TO_DATE, TO_DOUBLE, TO_BINARY_DOUBLE, TO_NUMBER, TO_CHAR, TO_TIMESTAMP.
args	An array of arguments to the operator: • For simple comparison operators, the args contains a single literal value. It is an error to provide 0 or more than 1 arguments. • For group comparison operators, the args contains 1 or more literal values. It is an error to provide 0 arguments. • For logical operators, the args contains sub-elements of the same structure, forming an expression tree.

For example: Using simple comparison operators

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json('{ "hybrid_index_name" : "my_hybrid_idx",
            "filter_by":
                    { "op"   : "<",
                      "col"  : "price",
                      "type" : "number",
                      "func" : "ABS"
                      "args" : ["10"] }
          }'))
FROM DUAL;
```

For example: Using group comparison operators

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json('{ "hybrid_index_name" : "my_hybrid_idx",
            "filter_by":
                    { "op"   :  "IN",
                      "path" : "DATA.brand",
                      "type" " "string",
                      "args" : ["nike", "adidas"] }
```

```
          }'))
FROM DUAL;
```

For example: Using logical operators

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json('{ "hybrid_index_name" : "my_hybrid_idx",
            "filter_by":
                    { "op"   :  "AND",
                      "args" : [
                      {"op" : "IN", "col" : "brand", "type" : "string",
"args" : ["nike", "adidas"]},
                      {"op" : "<", "col" : "price", "type" : "number",
"args" : ["10"]}]
                    }
          }'))
FROM DUAL;
```

return

Specify which fields to appear in the result set:

Parameter	Description
`topN`	Maximum number of best-matched results to be returned Value: Any integer greater than `0` (zero) Default: `20`
`values`	Return attributes for the search results. Values for scores range between 100 (best) to 0 (worse). • `rowid`: Row ID associated with the source document. • `score`: Final score computed from keyword-and-semantic search scores. • `vector_score`: Semantic score from vector search results. • `text_score`: Keyword score from text search results. • `vector_rank`: Ranking of chunks retrieved from semantic or `VECTOR_DISTANCE` search. • `text_rank`: Ranking of documents retrieved from keyword or `CONTAINS` search. • `chunk_text`: Human-readable content from each chunk. • `chunk_id`: ID of each chunk text. • `paths`: Paths from which the result occurred. Default: All the above return attributes EXCEPT `paths` are shown by default. As there are no paths for non-JSON, you need to explicitly specify the `paths` field.
`format`	Format of the results as: • `JSON` (default) • `XML`

For example:

```
SELECT DBMS_HYBRID_VECTOR.SEARCH(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "search_text"       : "C, Python",
         "return":
          {
             "values"        : [ "rowid", "score", "paths" ],
             "topN"          : 3,
             "format"        : "JSON"
          }
      }'))
FROM DUAL;
```

Complete Example With All Query Parameters

The following example shows a hybrid search query that performs separate text and vector searches against `my_hybrid_idx`. This query specifies the `search_text` for vector search using the `vector_distance` function as `prioritize teamwork and leadership experience` and the keyword for text search using the `contains` operator as `C and Python`. The search mode is `DOCUMENT` to return the search results as topN documents.

```
SELECT JSON_SERIALIZE(
  DBMS_HYBRID_VECTOR.SEARCH(
    json(
      '{ "hybrid_index_name" : "my_hybrid_idx",
         "search_fusion"     : "INTERSECT",
         "search_scorer"     : "rsf",
         "vector":
          {
             "search_text"   : "prioritize teamwork and leadership
experience",
             "search_mode"   : "DOCUMENT",
             "score_weight"  : 10,
             "rank_penalty"  : 1,
             "aggregator"    : "MAX",
             "inpath"        : ["$.main.body", "$.main.summary"],
             "accuracy"      : 95
          },
         "text":
          {
             "contains"      : "C and Python",
             "score_weight"  : 1,
             "rank_penalty"  : 5,
             "inpath"        : ["$.main.body"]
          },
         "return":
          {
             "format"        : "JSON",
             "topN"          : 3,
             "values"        : [ "rowid", "score", "vector_score",
                                 "text_score", "vector_rank",
                                 "text_rank", "chunk_text", "chunk_id",
"paths" ]
          }
      }'
    )
  ) pretty)
FROM DUAL;
```

The top 3 rows are ordered by relevance, with higher scores indicating a better match. All the return attributes are shown by default:

```
[
  {
    "rowid"          : "AAAR9jAABAAAQeaAAA",
    "score"          : 58.64,
    "vector_score"   : 61,
    "text_score"     : 35,
    "vector_rank"    : 1,
    "text_rank"      : 2,
    "chunk_text"     : "Candidate 1: C Master. Optimizes low-level system
(i.e. Database)
                       performance with C. Strong leadership skills in
guiding teams to
                       deliver complex projects.",
    "chunk_id"       : "1",
    "paths"          : ["$.main.body","$.main.summary"]
  },
  {
    "rowid"          : "AAAR9jAABAAAQeaAAB",
    "score"          : 56.86,
    "vector_score"   : 55.75,
    "text_score"     : 68,
    "vector_rank"    : 3,
    "text_rank"      : 1,
    "chunk_text"     : "Candidate 3: Full-Stack Developer. Skilled in
Database, C, HTML,
                       JavaScript, and Python with experience in building
responsive web
                       applications. Thrives in collaborative team
environments.",
    "chunk_id"       : "1",
    "paths"          : ["$.main.body", "$.main.summary"]
  },
  {
    "rowid"          : "AAAR9jAABAAAQeaAAD",
    "score"          : 51.67,
    "vector_score"   : 56.64,
    "text_score"     : 2,
    "vector_rank"    : 2,
    "text_rank"      : 3,
    "chunk_text"     : "Candidate 2: Database Administrator (DBA). Maintains
and secures
                       enterprise database (Oracle, MySql, SQL Server).
Passionate about
                       data integrity and optimization. Strong mentor for
junior DBA(s).",
    "chunk_id"       : "1",
    "paths"          : ["$.main.body", "$.main.summary"]
  }
]
```

End-to-end example:

To see how to create a hybrid vector index and explore all types of queries against the index, see Query Hybrid Vector Indexes End-to-End Example.

Related Topics

- Perform Hybrid Search

GET_SQL

Use the DBMS_HYBRID_VECTOR.GET_SQL PL/SQL function to return the internal SQL query that is generated from the parameters.

When calling the DBMS_HYBRID_VECTOR Search function, the API is called using the JSON Document format. Using the GET_SQL procedure shows the SQL that the DBMS_HYBRID_VECTOR.SEARCH API has generated. The resulting SQL can be used to view the query execution plan to view the index chosen for the hybrid search operation. An example is shown below:

```
SET LINESIZE 200;
SET PAGESIZE 1000;
SET TAB OFF;
SET TRIMSPOOL ON;
DECLARE
    res CLOB;
BEGIN
    res := dbms_hybrid_vector.get_sql(JSON('{"hybrid_index_name" :
"trecvol2j_idx",
                                            "search_text" : "offers",
                                            "return" : { "values" :
[ "score" ] } }'));
    execute immediate 'EXPLAIN PLAN FOR '||res;
END;
/


SELECT PLAN_TABLE_OUTPUT FROM TABLE(DBMS_XPLAN.DISPLAY(NULL,NULL,'ADVANCED'));
```

SEARCHPIPELINE

Use the standard table function DBMS_HYBRID_VECTOR.SEARCHPIPELINE to return a pipeline of row records.

This pipeline function accepts valid JSON query input and returns a pipeline of row records. The syntax is as shown below:

```
FUNCTION SEARCHPIPELINE(qparams JSON)
RETURN results PIPELINED;
```

The results is of type RECORD. The results contains the following fields:

Field	Type
doc_rowid	varchar2(18)

score	number
vector_score	number
text_score	number
vector_rank	number
text_rank	number
chunk_text	varchar2(32767)
chunk_id	varchar2(4000)
paths	varchar2(4000)

The record members are the column names in the SELECT statement. These names are the same as the JSON field names that are returned in DBMS_HYBRID_VECTOR_SEARCH(), except that paths is a list of field IDs in the record, where as the JSON result maps the ids to their actual paths (in an array). Also note that the result record could not have a member named rowid nor a member with a rowid type.

Example 102-1

```
SELECT
   chartorowid(doc_rowid) as doc_rowid,
   score,
   vector_score,
   text_score,
   vector_rank,
   text_rank,
   chunk_text,
   chunk_id,
   paths
FROM dbms_hybrid_vector.searchpipeline(JSON('{"hybrid_index_name" : "idx",
                                              "search_text" : "teamwork" }'));
```

If you do not wish to use the table function DBMS_HYBRID_VECTOR.SEARCHPIPELINE(), the original SEARCH API can be wrapped in a JSON_TABLE specification. This is shown in the example below:

```
SELECT jt.*
FROM
   JSON_TABLE(
        dbms_hybrid_vector.search(
                      json_object('hybrid_index_name' value 'idx',
                      'search_text' value 'teamwork'
                      RETURNING JSON)
                   ),
                   '$[*]' COLUMNS idx for ORDINALITY,
                                  doc_rowid PATH '$.rowid',
                                  score NUMBER PATH '$.score',
                                  vector_score NUMBER PATH '$.vector_score',
                                  text_score NUMBER PATH '$.text_score',
                                  vector_rank NUMBER PATH '$.vector_rank',
                                  text_rank NUMBER PATH '$.text_rank',
                                  chunk_text PATH '$.chunk_text',
                                  chunk_id PATH '$.chunk_id',
                                  paths PATH '$.paths'
                    ) jt
ORDER by idx ASC
```