PUBLIC

SAP HANA Platform 2.0 SPS 01
Document Version: 1.0 – 2017-04-12

SAP HANA Search Developer Guide

Content

1 SAP HANA Search Developer Guide

With the SAP HANA platform, your users can search tables and views much like they would when searching for information on the Internet. In SAP HANA, you can either query data using OData service definitions, directly with SQL queries, or via the built-in procedure `sys.esh_search()`. You build your UIs using SAPUI5.

Prerequisites

Before enabling search, note the following prerequisites:

- Your SAP HANA database must contain column-oriented tables.
- A valid data type must be assigned to each column. The data types of the columns determine how you can query your data.

Context

In SAP HANA, you can search on single or multiple columns of almost any visible data type. In addition to standard string search, SAP HANA also supports full-text search.

During a full-text search, the SAP HANA search engine examines both structured text, such as author and date attributes, and unstructured text, such as body text. Unlike a string search, the sequence of words and characters used for a text search is not critical for finding matches. A full-text index enables this functionality by analyzing and preprocessing the available text semantically. This includes normalization, tokenization, word stemming, and part of speech tagging.

Procedure

To enable search in SAP HANA, proceed as follows:

1. Define the persistency and indexing.
2. Create your search models.
3. Access your data using full-text search.
4. Create your search UI.

Related Information

Defining the Persistency and Indexing [page 6]

Creating Search Models [page 27]
Accessing Data Using Full-Text Search [page 83]
Creating Search UIs With SAPUI5 [page 296]

2 Defining the Persistency and Indexing

The first task when developing a search application is to define the persistency of your data before you can index the data for a search.

Context

There are two ways to define database tables, views and indexes, and search-related settings.

- **Programming model A**
  You can use Core Data Services (CDS) with its annotations syntax.
- **Programming model B**
  You can use the SQL syntax.

> **Note**
>
> We recommend using programming model A: CDS search annotations.

Procedure

1. Define the persistency for your data.

   Create database tables, which store the data that you want to search for. Create views to model your data. Views are also needed for the OData service definitions in the CDS programming model.
2. Define indexes and choose the columns which you want to search in.

   Use a powerful set of properties to set up all search related functions you want to use, for example the search mode FUZZY or text analysis.

Related Information

Creating Full-Text Indexes Using CDS Annotations [page 7]
Creating Full-Text Indexes Using SQL [page 10]

2.1 Creating Full-Text Indexes Using CDS Annotations

CDS uses a SQL-like syntax to define full-text indexes and fuzzy search indexes in the `technical configuration` part of an entity definition.

Procedure

1. You use the `technical configuration` part of the entity definition (in an `.hdbcds` or `.hdbdd` file) to create full-text indexes and fuzzy search indexes.

   CDS entities are used to define database tables. The elements of an entity describe the columns of the table. For more information, see *SAP HANA Developer Guide For SAP HANA XS Advanced Model* and *SAP HANA Core Data Services CDS Reference*.
2. Add a `technical configuration` part to the entity definition.
3. Define the full-text index or fuzzy search index inside the **technical configuration** part. You use syntax similar to SQL syntax to define the indexes.

Example

Sample Code

Example for the Technical Configuration for Full-Text Indexes:

After the entity block, you define the indexes in the `technical configuration` block for each column that you want to search in.

```
Entity award
{
  key id:               Integer;
  title:                String(500);
  abstract:             LargeString;
  institutionName:      String(100);
  institutionCountry:   String(100);
  institutionState:     String(10);
  institutionZip:       String(20);
  institutionCity:      String(100);
  institutionStreet:    String(100);
  programOfficer:       String(100);
}
technical configuration
{
  fulltext index fti_title     on (title)           fast preprocess off
                                                    language detection ('en')
                                                    fuzzy search index on;
  fulltext index fti_abstract  on (abstract)        fast preprocess off
                                                    language detection ('en')
                                                    fuzzy search index on;
  fulltext index fti_institutionName on (institutionName);
  fulltext index fti_programofficer  on (programOfficer);
  fuzzy search index                 on (institutionCity);
};
```

Sample Code

Example for the Technical Configuration for Complex Types:

Example of the `technical configuration` block with index definitions for complex types.

```
type contentType
{
  title:                String(500);
  abstract:             LargeString;
};

type institutionType
{
  institutionName:      String(100);
  institutionCountry:   String(100);
  institutionState:     String(10);
  institutionZip:       String(20);
  institutionCity:      String(100);
  institutionStreet:    String(100);
};

Entity award
{
  key id:               Integer;
  content:              contentType;
  institution:          institutionType;
  programOfficer:       String(100);
}
technical configuration
{
  fulltext index fti_title           on (content.title)     fast preprocess
off
                                                             language
detection ('en')
                                                             fuzzy search
index on;
  fulltext index fti_abstract        on (content.abstract)  fast preprocess
off
                                                             language
detection ('en')
                                                             fuzzy search
index on;
  fulltext index fti_institutionName on (institution.institutionName);
  fulltext index fti_programofficer  on (programOfficer);
  fuzzy search index                 on (institution.institutionCity);
};
```

Sample Code

Example for the `SearchIndex` Annotation for Full-Text Indexes (Repository/XS Classic):

```
entity myEntity
{
  key id: String(100);
  author: String(100);
  abstract: LargeString;
  content: LargeString;
  languagecode: String(2)
}
technical configuration
{
  fulltext index fti_abstract  on (abstract) fast preprocess off
                                             language column 'languagecode'
```

```
                                                    fuzzy search index on;
  fulltext index fti_content   on (content);
  fuzzy search index           on (author);
}
;
```

Sample Code

Example for the `SearchIndexes` Annotation for Structured Types (Repository/XS Classic):

```
type languageType
{
  code: String(2);
  name: String(100);
};

type documentType
{
  author:   String(100);
  abstract: LargeString;
  content:  LargeString;
  language: languageType;
};
entity documents
{
  key id: Integer;
  document: documentType;
}
technical configuration
{
  fulltext index fti_abstract  on (document.abstract) fast preprocess off
                                                     language column
'document.language.code'
                                                     fuzzy search index on;
  fulltext index fti_content   on (document.content);
  fuzzy search index           on (document.author);
}
;
```

Related Information

Full-Text Index Parameters [page 20]

2.2 Creating Full-Text Indexes Using SQL

When you create a TEXT or SHORTTEXT column in a table, SAP HANA automatically creates a corresponding full-text index. For columns of other data types, however, you have to manually create and define any required full-text indexes.

Context

A full-text index is an additional data structure that is created to enable text search features on a specific column in a table. Conceptually, full-text indexes support searching on columns in the same way that indexes support searching through books.

To create a full-text index, proceed as follows:

Procedure

1. Determine the columns for which you require an index.
2. Use the `CREATE FULLTEXT INDEX` statement to create an index with the specified index name.

```
CREATE FULLTEXT INDEX <index_name> ON <tableref> '(' <column_name> ')'
[<fulltext_parameter_list>]
```

3. Specify any of the following additional parameters for the full-text index:

```
LANGUAGE COLUMN <column_name>
LANGUAGE DETECTION '(' <string_literal_list> ')'
MIME TYPE COLUMN <column_name>
FUZZY SEARCH INDEX <on_off>
PHRASE INDEX RATIO <on_off>
CONFIGURATION <string_literal>
SEARCH ONLY <on_off>
FAST PREPROCESS <on_off>
TEXT MINING <on_off>
TEXT MINING CONFIGURATION <string_literal>
TEXT MINING CONFIGURATION OVERLAY <string_literal>
TEXT ANALYSIS <on_off>
MIME TYPE <specified mime type, e.g. application/pdf>
TOKEN SEPARATORS <\/;,.:-_()[]<>!?*@+{}="&>
To set the synchronization, specify one of the folowing parameters:
|SYNC
|ASYNC
|ASYNC FLUSH [QUEUE] EVERY n MINUTES
|ASYNC FLUSH [QUEUE] AFTER n DOCUMENTS
|ASYNC FLUSH [QUEUE] EVERY n MINUTES OR AFTER m DOCUMENTS
```

If you do not specify any parameters, the default values are used.

> **Note**
>
> For a complete list of available parameters, see Full-Text Index Parameters [page 20].

The system creates a separate, hidden full-text index column for each source column that you have specified. You can now create queries to search these columns.

Results

You can check the parameters of an existing full-text index using the `SYS.FULLTEXT_INDEXES` monitoring view.

Example

You want to create a full-text index i1 for table A, column C, with the following characteristics:

- Synchronous processing
- Fuzzy search index disabled
- Languages for language detection: English, German, and Korean

To create the index, you use the following syntax:

```
CREATE FULLTEXT INDEX i1 ON A(C) FUZZY SEARCH INDEX OFF
                                           SYNC
                                           LANGUAGE DETECTION ('EN','DE','KR')
```

Related Information

Full-Text Index Types [page 11]
Synchronization [page 15]
Full-Text Index Parameters [page 20]

2.2.1 Full-Text Index Types

SAP HANA automatically creates full-text indexes for columns of type TEXT and SHORTTEXT(n). For other column types, you must manually create any required full-text indexes.

Table 1:

Characteristic	TEXT/BINTEXT	SHORTTEXT (n)	Manually Created
SQL data type exposed to user	NCLOB	NVARCHAR(n)	Original data type
Data returned by SELECT	Original text data (returns normalized data if the SEARCH_ONLY parameter is ON)	Original data	Original data
SQL insertion mode	SYNC or ASYNC	SYNC	SYNC or ASYNC

Characteristic	TEXT/BINTEXT	SHORTTEXT (n)	Manually Created
Drop index	Yes (via drop column)	Yes (via drop column)	Yes
Text search via CONTAINS	Yes	Yes	Yes
SQL string search	Not possible	Possible	Depends on underlying data type
Change parameters of full-text index	Partially	Partially	All (with rebuild)
Rebuild index	No	No	Yes
Base type can be changed	No	No	No (no dependency between base-column and index available)

2.2.1.1 TEXT or BINTEXT Indexes

In a SAP HANA database, when you create a table that contains large text or binary documents, you can define the columns with the TEXT or BINTEXT data type. This data type allows you to query textual data and present content excerpts in search hit lists. You can also reconstruct the document and display its original textual content.

> **Note**
>
> Embedded document filters can automatically extract text and metadata from most popular document formats, including Adobe PDF, Microsoft Office, OpenDocument, email, HTML, XML, and plain text.
>
> Archive file formats, which may contain nested sub-files, are not supported. This includes ZIP and RAR files, as well as personal email folders like PST files.

When you create a TEXT/BINTEXT column and insert content, SAP HANA extracts and processes the text from the original document and then automatically generates a full-text index for the column. To create this full-text index, SAP HANA replaces the original data in the column with the processed text. This text is then returned with the data type NCLOB. The original data is no longer available.

If you insert new entries in the TEXT/BINTEXT column, the full-text index is automatically updated.

Limitations

The TEXT/BINTEXT data types have the following search-relevant limitations:

- SQL string searches are not supported.
- The SQL functions `CONCAT` and `JOIN` are not supported.
- TEXT columns cannot be converted to other data types.
- The LIKE predicate is not supported.

For TEXT/BINTEXT columns, SAP HANA does not support the following SQL expressions:

- HAVING
- WHERE with strings or non-alphanumeric characters
- ORDER BY
- GROUP BY
- Aggregate expressions (COUNT, MIN, MAX, etc.)
- JOIN ON

Changes to TEXT/BINTEXT Indexes

TEXT/BINTEXT full-text indexes are automatically generated and you do not specify names for them; therefore, you cannot directly manipulate them. However, when you create, alter, or drop a table column, the same change is automatically applied to the full-text index for that column.

By using the `ALTER TABLE` statement to affect changes on the index, you can alter the following parameters:

- PHRASE INDEX RATIO
- FUZZY SEARCH INDEX
- LANGUAGE DETECTION

Example

Sample Code

```
CREATE COLUMN TABLE <tablename>
(
    k int primary key,
    content TEXT
    FAST PREPROCESS OFF
    PHRASE INDEX RATIO 0.77
)
```

2.2.1.2 SHORTTEXT(n) Indexes

If the tables in your SAP HANA database contain columns with text strings that are relatively short in length, you can define those columns with the SHORTTEXT(n) data type. The SHORTTEXT(n) data type enables both SQL string search and full-text search capabilities.

SAP HANA preprocesses the text in the column and stores that preprocessed text as a full-text index in a hidden column attached to the original column. When queried, the text in the full-text index returns with the NVARCHAR data type. The original text is still available; however, search queries are performed, by default, on the text in the index.

When you create a column table and define a column with the data type SHORTTEXT(n), as in the following example, a full-text index is automatically generated. Whenever new entries are then inserted in the column, the full-text index is automatically and synchronously updated.

Example

```
CREATE COLUMN TABLE <tablename>
(
        k int primary key,
        content SHORTTEXT(100)
                                        FAST PREPROCESS OFF
                                        SEARCH ONLY ON
)
```

Changes to SHORTTEXT(n) Indexes

SHORTTEXT(n) full-text indexes are automatically generated and you do not specify names for them; therefore, you cannot directly manipulate them. However, when you create, alter, or drop a table column, the same change is automatically applied to the index for that column.

When using the ALTER TABLE statement to affect changes on the index, you can only alter the following parameters:

- PHRASE INDEX RATIO
- FUZZY SEARCH INDEX

Note

You cannot change the length of the original text and you cannot convert SHORTTEXT(n) to another data type.

2.2.1.3 Manually Created Indexes

If the tables in your SAP HANA database contain extensive columns that are frequently queried but do not have automatically generated full-text indexes, meaning they are not TEXT or SHORTTEXT(n) type columns, you can improve search performance by manually creating full-text indexes.

To manually create a full-text index, the column must have one of the following SQL data types:

- VARCHAR
- NVARCHAR
- ALPHANUM
- CLOB
- NCLOB
- BLOB

When you manually create an index, the system attaches a hidden column to the specified column. This hidden column contains textual data that SAP HANA Preprocessor has extracted from the text in the source column. The original text in the source column remains unchanged. Search queries are then performed on the hidden column; however, they always return the original text. Depending on the data type that is assigned to a source column, string search may be possible.

You can manually create an index directly after creating a table or you can create the index later. Once you create an index for a column, the system automatically processes any text that is inserted into this column and adds the processed text to the index. Processing for manually created indexes can be performed synchronously or asynchronously.

You can specify different parameters when you create a full-text index. If parameter changes are required later, you can change the values for the existing index directly or re-create the index with the parameters that you want to change.

Related Information

Creating Full-Text Indexes Using SQL [page 10]
Altering Full-Text Index Parameters [page 19]
Synchronization [page 15]
Full-Text Index Parameters [page 20]

2.2.2 Synchronization

Full-text indexes in a SAP HANA database must be created and updated in synchronization with the corresponding columns. This synchronization can be either synchronous or asynchronous.

Synchronous

In SAP HANA, indexes of the type TEXT and SHORTTEXT(n) are synchronous. Text preprocessing is automatically performed when a column is created or new text is inserted and the full-text index is then automatically updated. The system cannot transport any data to the proper database tables until text preprocessing is complete.

Asynchronous

If you manually create a full-text index, you can specify whether the index is synchronous or asynchronous. By default, manually created indexes are asynchronous. Text preprocessing is not initially performed when the table or column is created or whenever new text is inserted. In this case, inserting the results of the text preprocessing and writing the original data do not occur at the same time. Therefore, the full-text information may not be immediately available for searching.

To handle asynchronous processing of text, SAP HANA uses queues.

2.2.2.1 Queues

The queue is a mechanism used to enable a full-text index to operate asynchronously. This means that when you insert new entries into the column, the text is not made available in the column until it is preprocessed.

When you insert new entries, the queue sends the text to the preprocessor for analysis. It returns a serialized instance of a DAF (document analysis format) object, which is then processed further by the HANA column store. The result is stored in the full-text index.

The SAP HANA queue manager automatically creates a queue when you create an asynchronous full-text index or when the index server is started and the queue manager finds the information that a specific queue is needed. The queues are always created on the server on which the table is stored.

Every entry in the queue has one of the following processing states:

- New
- Preprocessing
- Preprocessed
- Indexing
- Error

If the original column entry is modified or deleted during text processing, the queue is notified and, if necessary, the entry is preprocessed again.

> **Note**
>
> The content of the queue is not made persistent at any stage. If the HANA index server process fails, the queue data is lost and the queue manager automatically restarts the process for those entries that were not already processed. Any incomplete text preprocessing is restarted from the beginning.

Flush Scheduling

When you create an asynchronous full-text index, you can specify when documents are removed from the queue after they are preprocessed and inserted into the full-text index; this is called flushing. You can schedule flushing based on either time or the number of documents. To do this, when you create the full-text index, define one of the following clauses with the `ASYNC` parameter:

- `FLUSH EVERY (n) MINUTES`
- `FLUSH AFTER (n) DOCUMENTS`
- `FLUSH EVERY (n) MINUTES OR AFTER (m) DOCUMENTS`

> **Note**
>
> You cannot specify negatives values for minutes or documents.

To determine when the queue of an existing full-text index is flushed, see the `FLUSH_EVERY_MINUTES` and `FLUSH_AFTER_ROWS` attributes in the view `FULLTEXT_INDEXES`.

Related Information

Manipulating Queue Processing [page 17]

2.2.2.2 Encoding of the Synchronization Modes

The synchronization mode of an index is encoded in two attributes in the SYS.FULLTEXT_INDEXES view.

If you want to know which synchronization mode is set for an index, you can check and encode the values of the `FLUSH_EVERY_MINUTES` and `FLUSH_AFTER_DOCUMENTS` attributes in the `SYS.FULLTEXT_INDEXES` view.

Table 2:

Sync Mode Clause	Value of Attribute FLUSH_EVERY_MINUTES	Value of Attribute FLUSH_AFTER_DOCUMENTS
SYNC	0	-1
ASYNC	-1	-1
ASYNC FLUSH EVERY n MINUTES (n>0)	n	-1
ASYNC FLUSH AFTER n DOCUMENTS (n>0)	-1	n
ASYNC FLUSH EVERY n MINUTES OR AFTER m DOCUMENTS	n	m

2.2.2.3 Manipulating Queue Processing

By default, queues are active initially and run automatically based on the parameters you specify when creating the full-text index. However, if necessary, you can manually manipulate the processing of an existing queue.

Context

To manipulate the processing of a queue, the following commands are available:

- `FLUSH`
  Updates the full-text index with the documents in the queue which have already been processed and removes them from the queue.
- `SUSPEND`
  Suspends the full-text index processing queue
- `ACTIVATE`
  Activates the full-text index processing queue if it has been suspended

To manipulate the processing of a queue:

Procedure

1. Identify which queue process you want to manipulate by using the monitoring view `M_FULLTEXT_QUEUES`. For information about the specific content of the view, see *SAP HANA System Tables and Monitoring Views*.
2. Use the `ALTER FULLTEXT INDEX` statement to flush, suspend, or reactivate the queue.

   Use the following syntax:

   ```
   ALTER FULLTEXT INDEX <index name> FLUSH|SUSPEND|ACTIVATE QUEUE
   ```

Related Information

SAP HANA System Tables and Monitoring Views: M_FULLTEXT_QUEUES

2.2.3 Text Analysis

Text analysis is a feature enabled with the full-text index to discover and classify entities in your documents.

Text analysis provides a vast number of possible entity types and analysis rules for many industries in many languages. You do not have to deal with this complexity when analyzing your individual set of documents. The language modules included with the software contain system dictionaries and provide an extensive set of predefined entity types. The extraction process can extract entities using these lists of specific entities. It can also discover new entities using linguistic models. Extraction classifies each extracted entity by entity type and presents this metadata in a standardized format. You can also customize the text analysis process and even define your own entity types.

Individual text analysis options are grouped into text analysis configurations, which are stored in SAP HANA in an XML format. The system includes a number of predefined configurations. You can use any of these, or create your own custom text analysis configurations. To use your own text analysis extraction dictionaries and extraction rules, you need to create a custom text analysis configuration.

2.2.4 Dropping Full-Text Indexes

If you want to delete a full-text index that you manually created, for example, because it is referenced only rarely or preprocessing is too time-consuming, you can drop the full-text index. For TEXT or SHORTTEXT full-text indexes, you cannot drop the full-text index; instead, you must delete the related column in the table.

Context

You also need to drop full-text indexes when adding or removing index parameters. As parameters cannot be added to or removed from an existing full-text index, if you want to change parameters, you must first drop the full-text index and then create a new index with the new parameters.

To drop a full-text index, you use the DROP FULLTEXT INDEX statement:

```
DROP FULLTEXT INDEX <index_name>
```

> **Note**
>
> Before you can drop a full-text index, you must remove the relationship between the source table and any existing $TA tables (for text analysis). To do so, use the following statement:
>
> ```
> ALTER TABLE SCHEMA <$TA_table> DROP <name_constraint>
> ```
>
> The name constraint must be the same as originally used when adding the constraint. For more information, see *Text Analysis*.

Related Information

Altering Full-Text Index Parameters [page 19]

2.2.5 Altering Full-Text Index Parameters

You can alter a full-text index after it is created. Altering an index includes changing the values of the parameters and altering the parameters by replacing the index. It can also be used to invoke text mining initialization.

Procedure

- To alter the parameters of a full-text index, use the `ALTER FULLTEXT INDEX` statement.

You can only use this statement to alter the following parameters:

- Fuzzy search index
- Phrase index ratio
- Text mining on/off (sets flag only, doesn't invoke text mining initialization)
- Text mining configuration (also invokes text mining initialization)
- Text mining configuration overlay (also invokes text mining initialization)

Example syntax:

```
ALTER FULLTEXT INDEX <index_name>
PHRASE INDEX RATIO <parameter value>
FUZZY SEARCH INDEX <on_off>
ALTER FULLTEXT INDEX <index_name>
TEXT MINING CONFIGURATION <config_name>
```

- To alter any other parameter, you must replace the existing full-text index as follows:
  a. Delete the existing full-text index by using the `DROP FULLTEXT INDEX` statement.
  b. Create a new index using the new parameter values.

Related Information

Creating Full-Text Indexes Using SQL [page 10]
Synchronization [page 15]
Queues [page 16]
Full-Text Index Parameters [page 20]

2.2.6 Full-Text Index Parameters

The content and behavior of a full-text index is configured by the use of both default and user-specified parameters. To view the configuration of a full-text index, you use the `SYS.FULLTEXT_INDEXES` view.

Note

Note that the parameters inside the `SYS.FULLTEXT_INDEXES` view are written with underscores. When using the parameters in SQL no underscores are to be used.

Table 3:
In SAP HANA, full-text indexes are configured using the following parameters:

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td><code>LANGUAGE COLUMN</code></td><td>
Specifies the language used for analyzing the document. If no language is specified, the language is detected automatically. The detected language is stored in text attribute <code>TEXT_AE</code> and can be queried via LANGUAGE(columnName).
Example values: 'EN', 'KO', 'DE'
With this option, you can refer to a column of the same database table where the language for the document is stored.
The column is read by the queue. Language columns in base tables should be of type NVARCHAR/ VARCHAR(m).
The language is specified using either ISO 639-1 (2 character) codes or ABAP (1 character) codes. Valid ISO 639-1 (2 character) language codes are listed in the <code>SYS.M_TEXT_ANALYSIS_LANGUAGES</code> view.
Note
The valid ISO 639-1 language codes can be displayed using the following SQL statement:
<code>SELECT * FROM "SYS"."M_TEXT_ANALYSIS_LANGUAGES"</code>
Note
Specify the "Neutral" language (two-letter code "UD") to perform only basic analysis on the document, such as tokenization, case normalization, and umlaut conversion. You may want to do this when a document is in a language not listed in the <code>SYS.M_TEXT_ANALYSIS_LANGUAGES</code> view (for example, Irish Gaelic). No entities will be extracted unless you have custom rules or dictionaries written in the neutral language. See the SAP HANA Text Analysis Developer Guide for additional information.
</td></tr>
<tr><td><code>MIME TYPE COLUMN</code></td><td>
This column holds a mimetype indicator that is used for preprocessing. If this is empty or invalid, auto detection is used.
Valid MIME types are listed in the <code>SYS.M_TEXT_ANALYSIS_MIME_TYPES</code> view.
Example values: 'text/plain','text/html','application/pdf'
</td></tr>
<tr><td><code>MIME TYPE</code></td><td>
Specifies the default MIME type used for preprocessing (see <code>MIME TYPE COLUMN</code>). If both <code>MIME TYPE</code> and <code>MIME TYPE COLUMN</code> are set, and if the corresponding cell for the <code>MIME TYPE COLUMN</code> is null, the value of parameter <code>MIME TYPE</code> is used for preprocessing.
Example: You can specify <code>MIME TYPE 'application/pdf'</code> if the table contains only pdf content.
</td></tr>
</table>

Parameter	Description
LANGUAGE DETECTION	Specifies the set of languages to be used for automatic language detection. If reliable language detection cannot be performed, the first language in the list will be used as the default language. This option is used to limit the languages for text analysis. Valid languages are listed in the SYS.M_TEXT_ANALYSIS_LANGUAGES view, and can be displayed using the following SQL statement: `SELECT * FROM "SYS"."M_TEXT_ANALYSIS_LANGUAGES".` Example: **LANGUAGE DETECTION ('EN','DE','JA')** The language is specified in the ISO 639-1 (2 characters) and not in the ABAP format (1 character with conversion exit) **Note** Only the LANGUAGE COLUMN parameter accepts SAP/ABAP codes. The LANGUAGE DETECTION parameter only accepts ISO 639-1 codes. **Note** In addition, you may specify the "neutral" language code ("UD") as the first language in the LANGUAGE DETECTION list. This causes "neutral" language processing to be used when the actual language cannot be determined automatically.
FAST PREPROCESS	Specifies whether fast preprocessing should be performed. Default: FAST PREPROCESS ON With fast preprocessing, the language detection just returns the default language, which is 'EN'. Linguistic analysis is skipped, and only a simple tokenizer is used. This simple tokenizer does not work for languages which do not use spaces as word separators (like Japanese). It cannot handle binary documents either. **Note** The parameter combination FAST PREPROCESS ON + [LANGUAGE COLUMN or LANGUAGE DETECTION or TEXT ANALYSIS ON] is not allowed.
FUZZY SEARCH INDEX	If this option is enabled, a special index is created for fuzzy search. This index accelerates fuzzy search, but uses additional memory.
SEARCH ONLY	If set to ON, the original content is not stored in the text attribute. You can get a reconstructed version of the document, but it may deviate from the original content. It is not possible to show the document in its original formatting when using the highlight function or to retrieve the HTML-converted data from the text attribute. The document will use less memory however. If the text attribute is created via a manually created full-text index, the source attribute that contains the original data is not affected by this setting.

Parameter	Description
`SYNC`	Insert/Update calls do not return until all documents have been preprocessed and inserted into the fulltext index.
`ASYNC`	Insert/Update calls m return before all documents are preprocessed and inserted into the index. However, preprocessing and insertion into index starts immediately upon insert/update call.
`ASYNC FLUSH EVERY n MINUTES`	Like ASYNC, but insertion into fulltext index takes place every n minutes instead of immediately upon insert/update call. Preprocessing still starts immediately.
`ASYNC FLUSH AFTER m DOCUMENTS`	Like ASYNC, but insertion into fulltext index takes place after m new documents have inserted/updated instead of immediately upon insert/update call. Preprocessing still starts immediately.
`ASYNC FLUSH EVERY n MINUTES OR AFTER m DOCUMENTS`	Like ASYNC, but insertion into fulltext index takes place every n minutes or after m new documents have been inserted/updated instead of immediately upon insert/update call. Preprocessing still starts immediately. **Note** Note that `DELETE` commands do not reduce the **document count**, e.g. if you use `FLUSH AFTER 5 DOCUMENTS`, insert four documents, delete one and afterwards insert one more document, the fulltext index gets updated as in total five documents where inserted.

Parameter	Description
CONFIGURATION	Specifies the text analysis options to be used. The options are bundled into a single file with the .hdbtextconfig file extension, referred to as a text analysis configuration file. You can create your own custom configurations with the SAP HANA repository, the HANA deployment infrastructure or the TEXT_CONFIGURATION_CREATE stored procedure. Or you can use one of the following predefined text analysis configurations delivered by SAP: • LINGANALYSIS_BASIC • LINGANALYSIS_STEMS • LINGANALYSIS_FULL • EXTRACTION_CORE • EXTRACTION_CORE_VOICEOFCUSTOMER • EXTRACTION_CORE_ENTERPRISE • EXTRACTION_CORE_PUBLIC_SECTOR • GRAMMATICAL_ROLE_ANALYSIS Specify the package path to the configuration file, the configuration file name, and file extension as follows: **[<package-path>::]<configuration-name>[.hdbtextconfig]** The file extension defaults to .hdbtextconfig. Omit the package path and file extension when using a predefined configuration. For example: **CONFIGURATION 'EXTRACTION_CORE'** If a custom configuration was created in a package, such as when using the SAP HANA repository, you must specify the full package path when using that configuration. For example: **CONFIGURATION 'acme.com.ta.config::MY_TA_CONFIGURATION'** This parameter requires the setting TEXT ANALYSIS = ON.
PHRASE INDEX RATIO	Stores information about the occurrence of words and the proximity of words to one another. If a phrase index is present, phrase searches are sped up (e.g. SELECT * FROM T WHERE CONTAINS(COLUMN1, '"cats and dogs"')). The float value is between 0.0 and 1.0. 1.0 means that the internal phrase index can use 100% of the memory size of the fulltext index.
TEXT ANALYSIS	This parameter enables the creation of the $TA table. Text analysis may occur even when TEXT ANALYSIS OFF is specified, depending on the other indexing parameters that are specified. If set to ON, the FAST PREPROCESS parameter is automatically set to OFF. Text analysis can extract entities such as persons, products, or places from documents, and thus enriches the set of structured information in SAP HANA. You can use these additional attributes, which are stored in a new table, when creating models (views) for analytics and search scenarios to enable improved analytics and search in SAP HANA.

Parameter	Description
`TOKEN SEPARATORS`	A set of characters used for token separation. Only ASCII characters are considered. Default values are `\/;,.:-_()[]<>!?*@+{}="&` If for example "`-`" is removed from the default token separators set, a chemical formula like "CH_3-CH_2-CH_3" won't get split into "CH_3", "CH_2" and "CH_3". It will remain one token.
`TEXT MINING`	If set to ON, text mining capabilities will be enabled on the indexed column. Text mining initialization will be called for the document data in that column when the table is loaded or when that data has changed.
`TEXT MINING CONFIGURATION`	States the name of the configuration file for text mining created with the SAP HANA repository, with the SAP HANA deployment infrastructure, or with the `TEXT_CONFIGURATION_CREATE` stored procedure. Default is `DEFAULT.textminingconfig`. Custom configuration files for text mining can be specified by name.
`TEXT MINING CONFIGURATION OVERLAY`	This parameter specifies literal text mining configuration data that should override the text mining configuration file. The format is the same as in the configuration file, typically a small subset of parameters. This allows parameter experimentation without requiring creation of configuration files for each case.

2.2.6.1 Memory Consumption of Full-Text Index Parameters

In SAP HANA, certain full-text index parameters can have a significant impact on memory consumption based on how they are defined.

Table 4:
The following full-text index parameters can have a significant impact on memory consumption:

Parameter	Memory Impact Details
`PHRASE INDEX RATIO`	If the value is greater than 0.0, there is additional memory overhead. The maximum memory consumption is the memory consumption of the full-text index multiplied by the parameter value.
`FUZZY SEARCH INDEX`	To increase response times for fuzzy search, when enabled, this parameter creates additional in-memory structures. For text-type columns, fuzzy search indexes require approximately 10% of the memory size of the column.

Parameter	Memory Impact Details
TEXT ANALYSIS	If this parameter is set to ON, an additional table is created for storing structured data extracted from the source text for text analysis. The amount of extracted data depends on the data in the source column, the text analysis rules, and the structure of the results. In certain cases, the memory consumption of the extracted data could exceed the memory consumption of the source data.

3 Creating Search Models

To search for data you have to model it first.

Context

Depending on which persistency you have chosen, you have various options for modelling your data:

- Using CDS search annotations
  If you use the CDS search approach, you define the search models either in the `hdbdd` file (XS Classic) or `hdbcds` file (XS Advanced) using search-specific annotations.
- Using the built-in procedure `sys.esh_config()`
  If you use the built-in procedure `sys.esh_config()`, you define the search models in a metadata document that contains the columns of the join view, SQL view, or table function that are relevant for search.
- Using attribute views
  If you use the SQL search approach, you create an attribute view in the SAP HANA modeler using the tables that you want to enable for the search. Create joins and add the attributes that you want to use for searching and displaying. You can also join additional attributes derived from the text analysis.

Related Information

Modeling with CDS Annotations (XS Advanced) [page 28]
Modeling with Built-In Procedure sys.esh_config() [page 61]
Modeling With Attribute Views [page 81]

3.1 Transformation Rules for CDS Views

CDS views can be transformed to join views if they follow certain rules.

To enable a search using the `CONTAINS()` predicate, you need to create a column view of type join (based on the view defined in the CDS model) at search runtime. This is automatically performed by the SQL interpreter when a `SELECT` statement on the CDS view is executed.

The search only returns correct results if this transformation of the CDS view to the column view is possible. Otherwise the `CONTAINS()` predicate used in the search query results in an SQL error (or, in some cases, may not return correct results).

CDS views can be transformed to join views if they follow the rules below:

- The CDS view is only created on top of the following database objects:
  - Column tables created via CDS entities
  - CDS views that are used as a wrapper for existing column tables created with a `CREATE COLUMN TABLE` statement.
  - CDS views created on top of the above database objects
  - Other database objects are not allowed. This includes, for example, column views created with the `CREATE COLUMN VIEW` statement, attribute views, and calculation views.
- The CDS view uses inner joins and left outer joins only.
- The CDS view does not use calculated columns or joins over calculated columns.
- Only equi-joins between tables are allowed.

3.2 Modeling with CDS Annotations (XS Advanced)

To define a search model on XS Advanced, you need to use the CDS annotations.

SAP HANA supports the following annotations, which are important elements of the CDS documents used to define CDS-compliant catalog objects (these catalog objects are used to create search models):

- `Search`
- `EnterpriseSearch`
- `EnterpriseSearchHana`
- `Hierarchy`
- `Semantics`

You can use further configuration options by defining values on the metadata of these annotations directly, or on the higher view, entity, or element level.

For details on programming with XS Advanced see the *SAP HANA Developer Guide for SAP HANA XS Advanced Model*.

Related Information

Annotation Search [page 29]
Annotation EnterpriseSearch [page 34]
Annotation EnterpriseSearchHana [page 42]
Annotation Hierarchy [page 46]

3.2.1 Annotation Search

This annotation marks a view as searchable. You define the fuzziness threshold as well as the specifics of term mappings at element level.

Syntax

```
namespace sap.common;

annotation Search
{
  /* annotation at view level */
  searchable:            Boolean;

  /* annotations at element level */
  defaultSearchElement:  Boolean;

  ranking:               String(6) enum {
                           HIGH='HIGH';
                           MEDIUM='MEDIUM';
                           LOW='LOW';
  };

  fuzzinessThreshold:    Decimal(3,2);

  termMappingDictionary: String(128);
  termMappingListID:     array of String(32);
};
```

Annotations on View Level

Table 5:

Annotation	Possible Values	Description
@Search.searchable	true, false	Defines if a CDS view or entity is generally relevant for search scenarios. The annotation offers a general switch and a means to quickly detect whether a view is relevant or not. Set to **true** to enable **@Search annotations**. At least one column has to be defined as `defaultSearchElement`.

Annotations on Element Level

Table 6:

Annotation	Possible Values	Description
@Search.defaultSearchElement	true, false	Defines a column as full-text search column. **Note** At least one column has to be defined as default full-text search column. Search in views without default full-text search elements is not supported.
@Search.fuzzinessThreshold	0..1	Defines the threshold for a fuzzy search. Default value is 1. This means that an exact search is performed.
@Search.ranking	#HIGH #MEDIUM #LOW	Defines the ranking weight for a column ('HIGH' = 1.0, 'MEDIUM' = 0.7, 'LOW' = 0.5)
@Search.termMappingDictionary	string	Defines the name of the term mapping table (format: `schemaname.tablename`). passed to the search option '`termMappingTable`' of the `CONTAINS()` predicate. **Note** This annotation cannot be used if the `@EnterpriseSearch.searchOptions` annotation is used. In this case the term mapping definition has to be added to the search options string.

Annotation	Possible Values	Description
@Search.termMappingListID	array of strings	Defines the names of the term mapping list IDs, passed to the search option '`termMappingListId`' of the `CONTAINS()` predicate. **Note** This annotation cannot be used if the `@EnterpriseSearch.searchOptions` annotation is used. In this case the term mapping definition has to be added to the search options string.

Related Information

FUZZY Search [page 149]
Term Mappings [page 218]

3.2.1.1 Metadata on View- and Entity Level

On view and entity level you can specify additional metadata.

3.2.1.1.1 searchable

Defines if a CDS view or entity is generally relevant for search scenarios.

This annotation must be set if *SearchDetails* annotations are being defined for elements of the corresponding CDS view or entity.

The annotation offers a general switch and a way of quickly detecting whether or not a view is relevant.

3.2.1.2 Metadata on Element Level

On element-level via the `Search` annotation those metadata can be specified which is relevant for all search scenarios.

3.2.1.2.1 defaultSearchElement

This annotation specifies that the element is to be considered in a full-text search (for example a `SELECT`…) where no columns are specified.

Such a search must not operate on all elements – for performance reasons, and because not all elements (e.g. internal keys) do qualify for this kind of access.

3.2.1.2.2 ranking

This annotation defines the ranking weight of a column that is used to calculate the overall score.

There are complementary aspects to ranking, where e.g. absolute values influence the ranking, without regard to the actual search terms.

Only one of the annotations `@EnterpriseSearchHana.weight` and `@Search.ranking` can be used for a column.

> **Note**
>
> This annotation should be used only if the attribute is annotated with usage mode `FREESTYLE_SEARCH`.

The `ranking` attribute can have the following values:

Table 7:

Value	Description
`HIGH`	The attribute is of high relevancy; this holds usually for ID and their descriptions. '`HIGH`' = 1.0
`MEDIUM`	The attribute is of medium relevancy; this holds usually for other, important attributes. '`MEDIUM`' = 0.7
`LOW`	Although the attribute is relevant for freestyle search, a hit in this attribute has no real significance for a result item's ranking. '`LOW`' = 0.5

> **Note**
>
> Beginning with metadata calls to API version 5 ('`/v5/$metadata`), the annotation `@EnterpriseSearchHana.weight` is always returned instead of `@Search.ranking`. '`HIGH`', '`MEDIUM`', and '`LOW`' are returned as their numeric values.

3.2.1.2.3 fuzzinessThreshold

This annotation specifies the least level of fuzziness (with regard to some comparison criteria passed at runtime) the element has to have to be considered in a fuzzy search at all.

Related Information

Fuzzy Score [page 158]

3.2.1.2.4 termMappingDictionary

This annotation specifies the table/entity that holds the term mappings to be considered in the context of a fuzzy search on this view.

Term mappings are usually synonyms, but other semantic classes (like homonyms) and degrees of identity.

Related Information

Term Mappings [page 218]

3.2.1.2.5 termMappingListID

This annotation specifies one or multiple lists within the term mapping dictionary mentioned before.

The list is implemented as a column of the term mapping table, with the list ID as content of this column. This concept has the aim to enable overarching term mapping dictionaries while being able to separate domain-specific content at the same time.

Related Information

[Term Mappings [page 218]](#)

3.2.2 Annotation EnterpriseSearch

You use the annotation `EnterpriseSearch` to define response attributes in your model.

Syntax

```
namespace sap.common;

using sap.common::CDSTypes.elementRef;

annotation EnterpriseSearch
{
  /* annotation at view level */
  enabled: Boolean;

  /* annotation at element level */
  key:       Boolean;

  /* annotation at view level */
  fieldGroupForSearchQuery: array of {
    name : String(128);
    elements : array of elementRef;
  };

  /* annotation at element level */
  defaultValueSuggestElement : Boolean;

  /* annotation at element level */
  searchOptions : String(5000);

  /* annotation at element level */
  filteringFacet {
    default:                    Boolean;
    numberOfValues:             Integer;
    caseInsensitiveAggregation: Boolean;
  };

  /* annotation at element level */
  presentationMode: array of String(20) enum {
                      DETAIL='DETAIL';
                      HIDDEN='HIDDEN';
                      IMAGE='IMAGE';
                      SUMMARY='SUMMARY';
                      THUMBNAIL='THUMBNAIL';
                      TITLE='TITLE';
                      NONE='NONE';
  };

  /* annotation at element level */
  snippets {
    enabled:  Boolean;
  };
```

```
  /* annotation at element level */
  highlighted {
    enabled:  Boolean;
  };
};
```

Annotations on View Level

Table 8:

Annotation	Possible Values	Description
@EnterpriseSearch.enabled	true, false	Has to be true. The view has to be defined as `@Search.searchable` and needs a valid anchor key definition.
@EnterpriseSearch.fieldGroupForSearchQuery	-	A field group defines a set of view columns that belong together. The field group is used in the query language to limit the search for a search term to the columns of the field group. For example: the field group 'name' that contains the columns 'firstName', 'lastName', 'companyName'.
@EnterpriseSearch.fieldGroupForSearchQuery.name	string	Defines the name of a field group. The name is used in the query language as a reference to the field group.
@EnterpriseSearch.fieldGroupForSearchQuery.elements	array of strings	Contains a list of column names. This defines the view columns that belong to the field group.

Annotations on Element Level

Table 9:

Annotation	Possible Values	Description
@EnterpriseSearch.key	true, false	Defines a column that is part of the anchor key. All anchor key columns have to be part of the same database table and this has to be the first table that is used in the view definition. **Note** At least one column of the view has to be marked as the anchor key. Search in views without an anchor key is not supported.
@EnterpriseSearch.searchOptions	string	Defines the search options string that is passed to the `CONTAINS()` predicate when searching in the column. **Note** Search options `emptyScore`, `emptyMatchesNull`, and `returnAll` are not allowed.
@EnterpriseSearch.usageMode	#AUTO_FACET #SUGGESTION	**Note** This annotation is deprecated. It is replaced by the annotations `@EnterpriseSearch.filteringFacet` and `@EnterpriseSearch.defaultValueSuggestElement`. AUTO_FACET: defines a column as a facet column. SUGGESTION: defines a column that is used to calculate suggestions.

Annotation	Possible Values	Description
@EnterpriseSearch.filteringFacet.de-fault	true, false	Defines a column as a facet column. If set to `true`, a facet is automatically returned for this column in the search response if facets are enabled (with custom query option `facets=all`, for example). If set to `false`, a facet for this column is returned only if the facet is requested explicitly (by adding the column name to custom query option facets, as, for example, `facets=AMOUNT,INSTITUTIONCOUNTRY`). Beginning with Enterprise Search version 5, columns of CDS type hana.ST_POINT can also be defined as facet columns.
@EnterpriseSearch.filteringFacet.num-berOfValues	integer > 0	Defines the number of values that are returned in the facet. If this parameter is not given, 10 values are returned by default.
@EnterpriseSearch.filteringFacet.ca-seInsensitiveAggregation	true, false	If set to `true`, a case-insensitive aggregation of facet values is done. If not given or set to `false`, the default aggregation is case-sensitive.
@EnterpriseSearch.defaultValueSug-gestElement	true, false	Defines a column that is used to calculate suggestions, if set to `true`.

Annotation	Possible Values	Description
@EnterpriseSearch.presentationMode	#DETAIL #HIDDEN, #IMAGE #SUMMARY #THUMBNAIL #TITLE #NONE	Only columns with a presentation mode other than 'NONE' will be returned in the search result. If presentation modes are not defined, all columns of the view will be returned. **Note** Only the following types of columns are allowed as response columns: • columns from the anchor table • columns from other tables that are joined as a 1:1 join • columns from other tables that are joined as a 1:n join and that are either marked as a multi-value column or that belong to a subobject. Setting a presentation mode for other columns of a table that is joined as a 1:n join results in wrong search results.
@EnterpriseSearch.snippets.enabled	true, false	Enables snippet calculation for a column. The column has to support text search. This means it needs a fulltext index or has to be of SQL type TEXT, BINTEXT or SHORTTEXT. **Note** This annotation must not be used for columns that are part of a 1:n join.
@EnterpriseSearch.highlighted.enabled	true, false	Enables highlighting for a column. The column has to support text search. This means it needs a fulltext index or has to be of SQL type TEXT, BINTEXT or SHORTTEXT.

3.2.2.1 Metadata on View Level and Entity Level

On view level and entity level, you can specify additional metadata.

3.2.2.1.1 @EnterpriseSearch.enabled

Defines on the view level whether the view is suitable for integration into the Search application, and is therefore a precondition for all @EnterpriseSearch annotations on element level.

The distinction is necessary because usually there are normalized views for an object's nodes (which are still likely to be searchable via @Search), and also dedicated, extremely denormalized search views. This annotation makes the denormalized views explicit.

3.2.2.2 Metadata on Element Level

Metadata, which is relevant for all search scenarios, can be specified on element level via the `EnterpriseSearch` annotation.

3.2.2.2.1 snippets

The annotation specifies that if a hit is returned for one of the search terms, the element does not receive the actual content, but only an excerpt with the search terms being marked.

> **Note**
>
> Note that if a fuzzy search is performed, the marked term is not necessarily identical to the search term.
>
> Note also that for a *-only search, the snippet tokens will be taken from the beginning of the text.

Search queries using the SNIPPETS function return the data type `NVARCHAR`.

3.2.2.2.2 highlighted

The annotation specifies that if a hit is returned for one of the search terms, the element does not receive the actual content, but the original content with marked search terms.

> **Note**
>
> Note that when a fuzzy search is performed, the marked term is not necessarily identical to the search term.

Note also that a *-only search will return no marked terms.

Given the definition of the fragment length at the `snippets` annotation, this is the same as highlighted, with an infinite fragment length.

3.2.2.2.3 presentation mode

The presentation mode specifies how a generic UI can present a result item in the search result in a manner which makes it easier to understand.

Only columns with a presentation mode other than `NONE` will be returned in the search result.

If you have not specified a presentation mode, all columns of the view will be returned.

> **Note**
>
> All response attributes, for example attributes with a presentation mode set, have to have cardinality 0..1, with regard to the item (this means for example that item data cannot be included in the header's response).
>
> Only columns from the anchor table and from other tables that are joined as a :1 join (1:1 or n:1) are allowed as response columns. Setting a presentation mode for columns of a table that is joined as a 1:n join results in incorrect search results.

The `presentation mode` attribute can have the following values:

Table 10:

Value Name	Description
`DETAIL`	The element will be displayed in a more detailed view, which is meant to provide the data a user needs to decide whether s/he wants to navigate to the object, respectively the most important data s/he needs in order not to have to navigate to the object's user interface.
`HIDDEN`	The element will be sent to the client, but it is not relevant to be displayed on the user interface. This presentation mode cannot be selected if any of the other ones are.
`IMAGE`	The element contains a graphic that will be displayed in a more detailed view. It is usually larger than `thumbnail`. This presentation mode is valid only for attributes with an appropriate data type.
`SUMMARY`	The element will be displayed in a summary view that adds a few important attributes to the title.

Value Name	Description
`THUMBNAIL`	The element contains a graphic to be displayed as a thumbnail in the result list. This presentation mode is valid only for attributes with an appropriate data type.
`TITLE`	The element will be part of the item title in the search result.
`NONE`	The element will not be part of the response.

3.2.2.2.4 usage mode

The usage mode defines how an attribute can be used correctly in a request.

> **Note**
>
> This annotation is deprecated. It is replaced by the annotations `@EnterpriseSearch.filteringFacet` and `@EnterpriseSearch.defaultValueSuggestElement`.

The `usage mode` attribute can have the following values:

Table 11:

Value Name	Description
`AUTO_FACET`	The attribute is relevant for filtering via facets. Attributes have to be medium selective: IDs are highly selective and therefore unusable in filtering facets (each filter value yields 1 hit). Attributes that split the result at a ratio of 100:1 cannot be used either. A search application will automatically provide filtering facets for these attributes.
`SUGGESTION`	The attribute is relevant for functions like type-ahead or auto completion. A search application will provide suggestion results only for the content of the attributes annotated with this usage mode. **Note** For performance reasons, this feature has to be limited to a very small number of attributes.

3.2.2.2.5 key

Defines on the view level which are the semantic key attributes, i.e. how the search result item instance is defined.

Because of the massive denormalization common to the `EnterpriseSearch` views, this semantic key tends to be a real subset of the view's technical key. The latter defines the Cartesian product across all involved tables; `EnterpriseSearch.key` defines the key of the entity to be found. All elements annotated as `key` have to stem from the same data base table.

3.2.3 Annotation EnterpriseSearchHana

The `EnterpriseSearchHana` annotation is not part of the CDS core annotations. It extends the search annotations defined by `EnterpriseSearch` and adds annotations, which are only available for SAP HANA CDS and which are used by the search runtime (`sys.esh_search()`).

Syntax

```
namespace sap.common;

using sap.common::CDSTypes.elementRef;

annotation EnterpriseSearchHana
{
  /* annotation at view level */
  identifier: String(127);

  /* annotation at element level */
  arbitraryCardinality: Boolean;

  /* annotation at view level */
  layoutStructuredObject: array of {
    layerId:              String(20);
    arbitraryCardinality: Boolean;
    defaultExpand: String(20) enum {
      WHY_FOUND = 'WHY_FOUND';
      "ALL"     = 'ALL';
    };
    elementList: array of {
      element: elementRef;
      key:     Boolean;
    };
  };

  /* annotation at element level */
  weight: Decimal(4,3);

  /* annotation at view or element level */
  uiResource {
    label {
      bundle: String(1000);
      key:    String(100);
    };
  };
```

```
};
```

Annotations on View Level

Table 12:

Annotation	Values	Description
@EnterpriseSearchHana.layoutStructuredObject	-	Defines a subobject of the anchor object. This means that the columns of the subobject come from a 1:n join and are returned in the search result as an array.
@EnterpriseSearchHana.layoutStructuredObject.layerId	string	Defines the name of the subobject.
@EnterpriseSearchHana.layoutStructuredObject.arbitraryCardinality	true	Specifies that the subobject comes from a 1:n join and that there may be multiple subobject instances for a given anchor object.
@EnterpriseSearchHana.layoutStructuredObject.defaultExpand	`#ALL` `#WHY_FOUND`	Specifies which subobjects are returned in the search result. `#ALL`: all subobjects of an anchor object are returned `#WHY_FOUND`: only the subobject of the anchor object are returned that contain at least one of the search terms. If the annotation is not given, the default value is `#ALL`.
@EnterpriseSearchHana.layoutStructuredObject.elementList	-	Specifies the view columns that belong to the subobject.
@EnterpriseSearchHana.layoutStructuredObject.elementList.element	string	Contains the name of a view column that belongs to the subobject

Annotation	Values	Description
@EnterpriseSearchHana.layoutStructuredObject.elementList.key	true, false	Specifies if a column is part of the subobject's key. Used to identify the distinct subobjects of an anchor object. If there are no key columns defined for a subobject, all columns are used to identify the distinct subobjects. **Note** All key columns have to be of a SQL type that can be used in a group by clause.
@EnterpriseSearchHana.layoutStructuredObject.layerId	string	Defines the name of the subobject.
@EnterpriseSearchHana.identifier	string	Defines the OData identifier of the view. The value has to be a valid OData identifier. If this annotation is not given, the OData identifier is calculated based on the view name. See the metadata service call for details on OData identifiers.
@EnterpriseSearchHana.uiResource.label.bundle	string	Defines the name of a resource bundle. Can be used by the search UI to load a view label. Info **Note** `uiResource` annotations do not change the behavior of `sys.esh_search()`. They are only returned in the metadata of the search configuration, so a UI may use these values to get texts in the end user's language.

Annotation	Values	Description
@EnterpriseSearchHana.uiResource.label.key	string	Defines the name of a key in a resource bundle. Can be used by the search UI to load a view label.

Annotations on Element Level

Table 13:

Annotation	Values	Description
@EnterpriseSearchHana.arbitraryCardinality	true, false	Defines a column as a multi-value column. This means that the column comes from an 1:n join and that it is returned in the search result as an array. **Note** This annotation is only allowed for columns with an SQL type that can be used in a group by clause.
@EnterpriseSearchHana.weight	DECIMAL(4,3) > 0	Defines the ranking weight of a column that is used to calculate the overall score. Only one of the annotations `@EnterpriseSearchHana.weight` and `@Search.ranking` can be used for a column. **Note** For metadata calls up to API version 4 ('`/$metadata`' or '`/v4/$metadata`), the annotation `@Search.ranking` is returned and the value given with `@EnterpriseSearchHana.weight` is rounded to '`HIGH`, '`MEDIUM`', or '`LOW`'.

Annotation	Values	Description
@EnterpriseSearchHana.uiRe-source.label.bundle	string	Defines the name of a resource bundle. Can be used by the search UI to load la-bels for view elements. **Note** `uiResource` annotations do not change the behavior of `sys.esh_search()`. They are only returned in the meta-data of the search configuration, so a UI may use these values to get texts in the end user's language. To get the annotation, API version 5 or higher of the metadata call has to be used ('/v5/$metadata').
@EnterpriseSearchHana.uiRe-source.label.key	string	Defines the name of a key in a resource bundle. Can be used by the search UI to load la-bels for view elements.

3.2.4 Annotation Hierarchy

The `Hierarchy` annotation contains a subset of the hierarchy annotations defined as CDS core annotations. It contains only the annotations that are used by `sys.esh_search()`.

Syntax

```
namespace sap.common;
using sap.common::CDSTypes.elementRef;
annotation Hierarchy {
  /* annotation at view level */
  leveled : array of {
    name : String(127);
    levels : array of {
      element : elementRef;
    };
  };
};
```

Annotations on View Level

Table 14: Metadata of Annotation Hierarchy

@Hierarchy.leveled	-	Defines a leveled hierarchy that is used to define leveled facets. Examples of leveled hierarchies: • year, month, date • country, region, city
@Hierarchy.leveled.name	string	Defines the name of the hierarchy.
@Hierarchy.leveled.levels	-	Defines which elements of the view specify the levels of the hierarchy. The sequence in which the elements are specified also defines the level on which the elements appear in the hierarchy.
@Hierarchy.leveled.levels.element	string	Contains the name of an element that defines the specific hierarchy level. **Note** The elements of the leveled hierarchy have to be annotated with `@EnterpriseSearch.filteringFacet.default: true`. Otherwise they will not be returned in the search response.

3.2.5 Annotation Semantics

The `Semantics` annotation contains a subset of the semantic annotations defined as CDS core annotations. It only contains the annotations that are currently used by `esh_search()`.

Syntax

```
namespace sap.common;
using sap.common::CDSTypes.elementRef;
annotation Semantics {
  /* annotations at element level */
  languageReference : elementRef;
```

```
  businessDate {
    at : Boolean;
    "from" : Boolean;
    to : Boolean;
    createdAt : Boolean;
    lastChangedAt : Boolean;
  };
  systemDate {
    createdAt : Boolean;
    lastChangedAt : Boolean;
  };
  contact {
    birthDate : Boolean;
  };
};
```

Annotations on Element Level

Table 15:

Annotation	Values	Description
@Semantics.languageReference	string	Defines the name of a column that contains the language code of the annotated column's content. If a column contains text in multiple languages, this annotation can be used to define the name of the column that contains the corresponding language code. If the language code column is a 1-character column (String(1)), 1-character SAP language codes are used. If the language code column is a 2-character column (String(2)), 2-character ISO language codes are used. **Note** Key and client columns cannot be defined as language dependent columns.

Annotation	Values	Description
@Semantics.businessDate.at @Semantics.businessDate.from @Semantics.businessDate.to @Semantics.businessDate.createdAt @Semantics.businessDate.lastChangedAt @Semantics.systemDate.createdAt @Semantics.systemDate.lastChangedAt @Semantics.contact.birthDate	true, false	Defines a column that contains a date or time stamp. The annotations are valid for some CDS types only: • LocalDate, UTCDateTime, UTCTimestamp (SQL types `DATE`, `SECONDDATE`, `TIMESTAMP`): These types are always treated as a date or time stamp. The annotation is allowed, but does not change the behavior. • String(8) - SQL type `NVARCHAR(8)`: The column is treated as a date column. Dates stored in the column have to be of format `YYYYMMDD`. • Decimal(15,0) - SQL type `DECIMAL(15)`: The column is treated as a time stamp (similar to CDS type UTCDateTime). Time stamps stored in the column have to be of format `YYYYMMDDHHMMSS`. Only years between 1000 and 9999 are supported. • Decimal(21,7) - SQL type `DECIMAL(21,7)`: The column is treated as a time stamp (similar to CDS type UTCTimestamp). Time stamps stored in the column have to be of format `YYYYMMDDHHMMSS.NNNNNNN`. Only years between 1000 and 9999 are supported. The annotation changes the behavior of String and Decimal columns so that esh_search() treats them like other date or time stamp columns: • Facets are returned as date intervals. Only valid dates or date patterns are searched in these columns (except for Decimal columns that

Annotation	Values	Description
		also accept decimal numbers as search terms). String columns accept the initial value of ABAP DATS columns ('00000000') as a search term. Decimal columns accept the initial value 0 as a search term.

3.2.6 CDSTypes

The `CDSTypes.hdbcds` file contains a type definition for CDS type `ElementRef`. It will be used until SAP HANA CDS supports `ElementRef` as a built-in type.

Syntax

Code Syntax

```
namespace sap.common;
context CDSTypes
{
  type elementRef: String(127);
};
```

3.2.7 Creating a Term Mapping Table via CDS

If you use programming model A (CDS) you can use term mappings as well as with the SQL approach.

Context

You need to create a table for the term mappings in your `hdbdd` file:

Procedure

1. Inside the CDS file create a term mapping table as shown in the example below:

Sample Code

```
@Catalog.tableType: #COLUMN
entity myTermMappings
{
  key MAPPING_ID: String(32);
  LIST_ID: String(32) NOT NULL;
  LANGUAGE_CODE: String(2);
  TERM_1: String(200) NOT NULL;
  TERM_2: String(200) NOT NULL;
  WEIGHT: Decimal(3,2) NOT NULL;
};
```

2. To use the term mapping table on a certain column of a CDS view add the following search annotation to this column:

```
@Search: { termMappingDictionary: 'myTermMappings', termMappingListID: ['L1'] }
```

Related Information

Term Mappings [page 218]

3.2.8 Example: Defining Search Configurations

See the example code, which contains tables, view definitions, and models.

The example below shows the configuration of a search view that joins several tables of an awards database.

Anchor object and anchor key: The anchor object of the view is the award that is identified by the anchor key column called 'id'.

1:n joins: The data contains 1:n information that is joined to the award objects. The 1:n objects are defined as subobjects 'investigator' and 'programElement'. To see the difference in search results, the program reference is not defined as a subobject. Instead, 'programReferenceText' is defined as multi-value element.

Field groups: Some of the view elements are grouped as field groups 'award', 'investigator', and 'organization'. The 'award' group, for example, can be used to search for text in the award title, abstract, or id.

Leveled facets: Hierarchies are defined to return leveled facets 'instLocation' and 'organization'.

Snippets and highlighted text: Instead of the full abstract a text snippet shall be returned only. The award title shall be returned with highlighting if any of the search terms is found in the title. So the 'abstract' and 'title' columns re annotated with snippets and highlighted annotations and do not have a presentation mode annotation.

Sample Code

Search and EnterpriseSearch Annotations

```
namespace search;
```

```
using sap.common::Hierarchy;
using sap.common::Search;
using sap.common::EnterpriseSearch;
using sap.common::EnterpriseSearchHana;

using awards::award;
using awards::award_investigator;
using awards::award_programElement;
using awards::award_programReference;
using awards::investigator;
using awards::organization;
using awards::programElement;
using awards::programReference;
using awards::state;

@Search.searchable: true
@EnterpriseSearch.enabled: true

@EnterpriseSearch.fieldGroupForSearchQuery: [ {
    name: 'award', elements: [ 'id', 'title', 'abstract' ]
  }, {
    name: 'investigator', elements: [ 'invFirstName', 'invLastName',
'invEmail' ]
  }, {
    name: 'organization', elements: [ 'orgDirectorate', 'orgDivision' ]
  } ]

@EnterpriseSearchHana.layoutStructuredObject: [ {
    layerId: 'investigator', arbitraryCardinality: true, defaultExpand: #ALL,
    elementList: [ { element: 'invFirstName' }, { element: 'invLastName' },
{ element: 'invEmail' },
                   { element: 'invRole' }, { element: 'invStartDate' },
{ element: 'invEndDate' } ]
  }, {
    layerId: 'programElement', arbitraryCardinality: true, defaultExpand: #ALL,
    elementList: [ { element: 'programElementCode' }, { element:
'programElementText' } ]
  } ]

@Hierarchy.leveled: [ {
  name:   'instLocation',
  levels: [ { element : 'instCountry' },
            { element : 'instState' },
            { element : 'instCity' } ]
  }, {
  name:   'organization',
  levels: [ { element : 'orgDirectorate' },
            { element : 'orgDivision' } ]
  } ]

@EnterpriseSearchHana.identifier: 'awards'
view awardssearch
as select from award a
left outer join organization o on a.organizationCode = o.code              /*
1:1   */
left outer join state s on a.institutionState = s.code                     /*
1:1   */
left outer join award_investigator ai on a.id = ai.awardId                 /*
1:n   */
left outer join investigator i on ai.investigatorId = i.id                 /*
1:n:1 */
left outer join award_programElement ape on a.id = ape.awardId             /*
1:n   */
left outer join programElement pe on ape.programElementCode = pe.code      /*
1:n:1 */
left outer join award_programReference apr on a.id = apr.awardId           /*
1:n   */
```

```
left outer join programReference pr on apr.programReferenceCode = pr.code  /*
1:n:1 */
{
  @EnterpriseSearch.key: true
  @EnterpriseSearch.presentationMode: [ #TITLE ]
  a.id,
  @EnterpriseSearch.highlighted.enabled: true
  @Search.defaultSearchElement: true
  @EnterpriseSearch.defaultValueSuggestElement: true
  @Search.fuzzinessThreshold: 0.8
  @EnterpriseSearch.searchOptions: 'textSearch=compare,
similarCalculationMode=searchcompare, composeWords=2, decomposeWords=2,
compoundWordWeight=0.95, excessTokenWeight=0.05'
  a.title,
  @EnterpriseSearch.snippets.enabled: true
  @Search.defaultSearchElement: true
  @Search.fuzzinessThreshold: 0.8
  @EnterpriseSearch.searchOptions: 'ts=compare, scm=searchcompare, cw=2, dw=2,
cww=0.95, etw=0.01'
  a.abstract,
  @EnterpriseSearch.presentationMode: [ #DETAIL ]
  @EnterpriseSearch.filteringFacet.default: true
  @Search.defaultSearchElement: true
  @Search.ranking: #LOW
  a.effectiveDate,
  @EnterpriseSearch.presentationMode: [ #DETAIL ]
  a.expirationDate,
  a.minAmdLetterDate,
  a.maxAmdLetterDate,
  @EnterpriseSearch.presentationMode: [ #DETAIL ]
  @EnterpriseSearch.filteringFacet: { default: true, numberOfValues: 5 }
  @Search.defaultSearchElement: true
  @Search.ranking: #LOW
  a.instrument,
  @EnterpriseSearch.presentationMode: [ #DETAIL ]
  @EnterpriseSearch.filteringFacet: { default: true, numberOfValues: 5 }
  a.amount,
  a.arraAmount,

  @EnterpriseSearch.presentationMode: [ #TITLE ]
  @Search.defaultSearchElement: true
  @EnterpriseSearch.defaultValueSuggestElement: true
  @Search.ranking: #MEDIUM
  @EnterpriseSearch.searchOptions: 'textSearch=compare'
  a.institutionName as "instName",
  a.institutionPhone as "instPhone",
  @EnterpriseSearch.presentationMode: [ #DETAIL ]
  @EnterpriseSearch.filteringFacet: { default: true, numberOfValues: 5 }
  a.institutionCountry as "instCountry",
  a.institutionState as "instStateCode",
  @EnterpriseSearch.presentationMode: [ #DETAIL ]
  @EnterpriseSearch.filteringFacet: { default: true, numberOfValues: 5 }
  @Search.defaultSearchElement: true
  @Search.ranking: #LOW
  s.name as "instState",
  a.institutionZip as "instZip",
  @EnterpriseSearch.presentationMode: [ #DETAIL ]
  @EnterpriseSearch.filteringFacet: { default: true, numberOfValues: 5 }
  @Search.defaultSearchElement: true
  @Search.ranking: #MEDIUM
  @Search.fuzzinessThreshold: 0.8
  a.institutionCity as "instCity",
  a.institutionStreet as "instStreet",

  a.programOfficer,

  a.organizationCode as "orgCode",
  @EnterpriseSearch.presentationMode: [ #DETAIL ]
```

```
  @EnterpriseSearch.filteringFacet: { default: true, numberOfValues: 5 }
  @Search.defaultSearchElement: true
  @Search.ranking: #LOW
  @EnterpriseSearch.searchOptions: 'textSearch=compare'
  o.directorate as "orgDirectorate",
  @EnterpriseSearch.presentationMode: [ #DETAIL ]
  @EnterpriseSearch.filteringFacet: { default: true, numberOfValues: 5 }
  @Search.defaultSearchElement: true
  @Search.ranking: #LOW
  @EnterpriseSearch.searchOptions: 'textSearch=compare'
  o.division as "orgDivision",

  @EnterpriseSearch.presentationMode: [ #DETAIL ]
  @Search.defaultSearchElement: true
  @EnterpriseSearch.defaultValueSuggestElement: true
  @Search.ranking: #MEDIUM
  @Search.fuzzinessThreshold: 0.8
  i.firstName as "invFirstName",
  @EnterpriseSearch.presentationMode: [ #DETAIL ]
  @Search.defaultSearchElement: true
  @EnterpriseSearch.defaultValueSuggestElement: true
  @Search.ranking: #MEDIUM
  @Search.fuzzinessThreshold: 0.8
  i.lastName as "invLastName",
  @EnterpriseSearch.presentationMode: [ #DETAIL ]
  @Search.defaultSearchElement: true
  @Search.ranking: #LOW
  i.email as "invEmail",
  @EnterpriseSearch.presentationMode: [ #DETAIL ]
  ai.investigatorRole as "invRole",
  ai.investigatorStartDate as "invStartDate",
  ai.investigatorEndDate as "invEndDate",

  @EnterpriseSearch.presentationMode: [ #DETAIL ]
  pe.code as "programElementCode",
  @EnterpriseSearch.presentationMode: [ #DETAIL ]
  @EnterpriseSearch.filteringFacet: { default: true, numberOfValues: 5 }
  @EnterpriseSearch.snippets.enabled: true
  @Search.defaultSearchElement: true
  @Search.ranking: #LOW
  @EnterpriseSearch.searchOptions: 'textSearch=compare'
  pe.text as "programElementText",
  pr.code as "programReferenceCode",
  @EnterpriseSearch.presentationMode: [ #DETAIL ]
  @EnterpriseSearch.filteringFacet: { default: true, numberOfValues: 5 }
  @Search.defaultSearchElement: true
  @Search.ranking: #LOW
  @EnterpriseSearchHana.arbitraryCardinality: true
  @EnterpriseSearch.searchOptions: 'textSearch=compare'
  pr.text as "programReferenceText"
};
```

3.3 Modeling with CDS Annotations (XS Classic)

To define a search model on XS Classic, you need to use the CDS annotations.

SAP HANA with XS Classic supports the following annotations, which are important elements of the CDS documents used to define CDS-compliant catalog objects (these catalog objects are used to create search models):

- `Search`
- `EnterpriseSearch`

You can use further configuration options by defining values on the metadata of these annotations directly, or on the higher view, entity, or element level.

3.3.1 Annotation Search (XS Classic)

This annotation marks a view as searchable. You define the fuzziness threshold as well as the specifics of term mappings at element level.

Syntax

```
annotation Search
{
  /* annotation at view level */
  searchable:            Boolean;

  /* annotations at element level */
  defaultSearchElement:  Boolean;

  ranking:               String(6) enum {
                           HIGH='HIGH';
                           MEDIUM='MEDIUM';
                           LOW='LOW';
  };

  fuzzinessThreshold:    Decimal(3,2);

  termMappingDictionary: String(128);
  termMappingListID:     array of String(32);
};
```

Annotations on View Level

Table 16:

Annotation	Possible Values	Description
@Search.searchable	true, false	Defines if a CDS view or entity is generally relevant for search scenarios. The annotation offers a general switch and a means to quickly detect whether a view is relevant or not. Set to **true** to enable **@Search annotations**. At least one column has to be defined as `defaultSearchElement`.

Annotations on Element Level

Table 17:

Annotation	Possible Values	Description
@Search.defaultSearchElement	true, false	Defines a column as full-text search column. **Note** At least one column has to be defined as default full-text search column. Search in views without default full-text search elements is not supported.
@Search.fuzzinessThreshold	0..1	Defines the threshold for a fuzzy search. Default value is 1. This means that an exact search is performed.
@Search.ranking	#HIGH #MEDIUM #LOW	Defines the ranking weight for a column ('HIGH' = 1.0, 'MEDIUM' = 0.7, 'LOW' = 0.5)

Annotation	Possible Values	Description
@Search.termMappingDictionary	string	Defines the name of the term mapping table (format: `schemaname.tablename`). passed to the search option '`termMappingTable`' of the `CONTAINS()` predicate.
@Search.termMappingListID	array of strings	Defines the names of the term mapping list IDs, passed to the search option '`termMappingListId`' of the `CONTAINS()` predicate.

Related Information

FUZZY Search [page 149]

Term Mappings [page 218]

3.3.2 Annotation EnterpriseSearch (XS Classic)

You use the annotation `EnterpriseSearch` to define response attributes in your model.

Syntax

```
annotation EnterpriseSearch {
  /* annotation at view level */
  enabled : Boolean;
  /* annotations at element level */
  usageMode : array of String(20) enum {
                 AUTO_FACET='AUTO_FACET';
                SUGGESTION='SUGGESTION';
               };
  key :        Boolean;
  snippets {
    enabled:   Boolean;
  };
  highlighted {
    enabled:   Boolean;
  };
  presentationMode : array of String(20) enum {
     DETAIL='DETAIL';
    HIDDEN='HIDDEN';
    IMAGE='IMAGE';
    SUMMARY='SUMMARY';
    THUMBNAIL='THUMBNAIL';
    TITLE='TITLE';
```

```
      NONE='NONE';
   };
};
```

Annotations on View Level

Table 18:

Annotation	Possible Values	Description
@EnterpriseSearch.enabled	true, false	Has to be true. The view has to be defined as `@Search.searchable` and needs a valid anchor key definition.

Annotations on Element Level

Table 19:

Annotation	Possible Values	Description
@EnterpriseSearch.key	true, false	Defines a column that is part of the anchor key. All anchor key columns have to be part of the same database table and this has to be the first table that is used in the view definition. **Note** At least one column of the view has to be marked as the anchor key. Search in views without an anchor key is not supported.
@EnterpriseSearch.usageMode	#AUTO_FACET #SUGGESTION	AUTO_FACET: defines a column as a facet column. SUGGESTION: defines a column that is used to calculate suggestions.

<table>
<tr><th>Annotation</th><th>Possible Values</th><th>Description</th></tr>
<tr><td>@EnterpriseSearch.presentationMode</td><td>#DETAIL
#HIDDEN,
#IMAGE
#SUMMARY
#THUMBNAIL
#TITLE
#NONE</td><td>Only columns with a presentation mode other than 'NONE' will be returned in the search result.

If presentation modes are not defined, all columns of the view will be returned.

Note

Only the following types of columns are allowed as response columns:
columns from the anchor table
columns from other tables that are joined as a 1:1 join
columns from other tables that are joined as a 1:n join and that are either marked as a multi-value column or that belong to a subobject.
Setting a presentation mode for other columns of a table that is joined as a 1:n join results in wrong search results.</td></tr>
<tr><td>@EnterpriseSearch.snippets.enabled</td><td>true, false</td><td>Enables snippet calculation for a column.

The column has to support text search. This means it needs a fulltext index or has to be of SQL type TEXT, BINTEXT or SHORTTEXT.

Note

This annotation must not be used for columns that are part of a 1:n join.

Setting this annotation for other columns results in wrong search results.</td></tr>
</table>

Annotation	Possible Values	Description
@EnterpriseSearch.highlighted.enabled	true, false	Enables highlighting for a column. The column has to support text search. This means it needs a fulltext index or has to be of SQL type TEXT, BINTEXT or SHORTTEXT. **Note** This annotation must not be used for columns that are part of a 1:n join. Setting this annotation for other columns results in wrong search results.

3.3.3 Example: Defining Search Configurations (XS Classic)

See the example code, which contains tables, view definitions, and models.

In this example, we define the indexing and views and models, which are required for the search.

```
entity test
{
  key id: String(10);
  col1: String(100);
  col2: String(100);
  col3: String(100);
}
technical configuration
{
  fulltext index fti_col2   on (col2);
  fulltext index fti_col3   on (col3);
}
;

@Search.searchable: true
@EnterpriseSearch.enabled: true
define view test_view as select from test
{
    @EnterpriseSearch.key: true
    @EnterpriseSearch.presentationMode: [ #TITLE ]
    id as vid,
    @Search.defaultSearchElement: true
    @EnterpriseSearch.presentationMode: [ #TITLE ]
    col1,
    @Search.defaultSearchElement : true
    col2;
    @EnterpriseSearch.usageMode: [ #AUTO_FACET ]
    col3;
};
```

3.4 Modeling with Built-In Procedure sys.esh_config()

You can use the built-in procedure `sys.esh_config()` as an alternative to modeling using CDS.

Overview

You can use the built-In procedure `sys.esh_config()` to define search configurations for existing tables and views.

The search configurations are based on the OData metadata service but are implemented as a SAP HANA SQL built-in procedure.

Supported Database Objects

Search configurations can be created for database objects of the following types:

Table 20:

Database Object Type	Description
SQL Views	SQL Views are views created using the `CREATE VIEW` statement. Only views that can be transformed internally to a join view are supported for search. Other, more complex views cannot be used. The source are OpenCDS views without parameters.
Table Functions	Table functions are functions that are created using the `CREATE FUNCTION` statement, that return a table object, and that contain a single `SELECT` statement only. Function parameters can be used in where conditions or in join conditions as 'search constraints' to exclude certain rows from the tables before the search is performed. This is similar to join views with constraints that are used by Enterprise Search. Restrictions for table functions are the same as for SQL views.

3.4.1 Interface of sys.esh_config()

A NCLOB container is used to pass the URI, the http method, and the payload to the built-in procedure. Parameters of the http header cannot be defined, so the built-in procedure uses meaningful default values.

The interface of `sys.esh_config()` looks as follows:

Source Code

```
CREATE PROCEDURE sys.esh_config
(
    IN  request  NCLOB,
    OUT response TABLE ( response NCLOB )
)
AS BUILTIN;
```

3.4.1.1 Bulk Requests and Transaction Handling

The input parameter allows bulk requests to make a large number of configuration changes in a single call.

For each element of the bulk request, one row in the response table is returned.

JSON Format for Bulk Requests

Source Code

```
CALL sys.esh_config('
[
  {
    "uri":"~/$metadata/EntitySets",
    "method":"POST",
    "content":{
      ...
    }
  },
  {
    "uri":"~/$metadata/EntitySets",
    "method":"POST",
    "content":{
      ...
    }
  },
  ...
  {
    ...
  }
]
')
```

Each configuration change of the bulk request is processed independently of other changes. So some elements might fail while others finish successfully.

Each change is processed in a single transaction. So it either fails completely and leaves the existing configuration table untouched, or all changes are visible after the function call ends.

The call to `sys.esh_config()` is not embedded in the transaction of the caller, similar to a http service that is independent of the caller's database transactions.

3.4.1.2 Error Handling

If the request object does not contain a valid JSON array, a SQL exception is returned.

Examples of errors that can occur include a malformed JSON or missing `uri`, `method`, or `content` elements.

If a part of the bulk request cannot be processed for any reason, the corresponding row in the response parameter contains a JSON error response. Examples of errors that can occur include an invalid uri, an invalid method, content not following the format described below, or validation errors because of an invalid configuration.

Format of an Error Message

Source Code

```
{
  "error":{
    "code":"8103101",
    "message":"Configuration failed",
    "details":[
      {
        "code":"8103005",
        "message":"\"TXT1\": parameter \"@EnterpriseSearch.kei\" must not be
used for view column"
      }
    ]
  }
}
```

Error responses can also contain more than one error. In this case, an array of error messages is returned, as shown in the following example:

Example: Error messsage containing two errors

Source Code

```
{
  "error":{
    "code":"8103101",
    "message":"Configuration failed",
    "details":[
      {
        "code":"8103005",
        "message":"\"ID1\": parameter \"@EnterpriseSearch.clnt\" must not be
used for view column"
      },
      {
        "code":"8103005",
        "message":"\"TXT1\": parameter \"@EnterpriseSearch.kei\" must not be
used for view column"
      }
    ]
  }
}
```

3.4.2 Creating a Configuration with Method 'POST'

To create a new configuration, the configuration API accepts a metadata document that contains the columns of the join view, SQL view, or table function that are relevant for search. For these columns, the search annotations have to be defined in the metadata document. Columns without search annotations that are not relevant to the search model can be ommitted in the configuration.

The example below shows a call that creates two search configurations. The URI `"~/$metadata/EntitySets"` and the `POST` method are used to create the new configurations. `"Fullname"` references the schema and the name of the database object to be used for the search.

For details about annotations, see the description of the metadata annotations available for `sys.esh_config()`. If the configuration is created without errors, an empty row in the response table is returned. If errors occur, an error message is returned in the corresponding row of the response table.

Source Code

```
[
  {
    "uri":"~/$metadata/EntitySets",
    "method":"POST",
    "content":{
      "Fullname":"DEVELOP/Customer",
      "EntityType":{
        "@Search.searchable":true,
        "@EnterpriseSearch.enabled":true,
        "Properties":[
          {
            "Name":"CustomerNumber",
            "@Search.defaultSearchElement":true
          },
          {
            "Name":"FirstName",
            "@Search.defaultSearchElement":true,
            "@Search.fuzzinessThreshold":0.7,
            "@Search.ranking":"HIGH"
          },
          {
            "Name":"LastName",
            "@Search.defaultSearchElement":true,
            "@Search.fuzzinessThreshold":0.7
          },
          {
            "Name":"YearofBirth",
            "@Search.defaultSearchElement":true,
            "@EnterpriseSearch.usageMode":[
              "AUTO_FACET"
            ]
          }
        ]
      }
    }
  },
  {
    "uri":"~/$metadata/EntitySets",
    "method":"POST",
    "content":{
      "Fullname":"DEVELOP/Wine",
      "EntityType":{
        "@Search.searchable":true,
        "@EnterpriseSearch.enabled":true,
        "Properties":[
```

```
            {
              "Name":"WineNumber",
              "@Search.defaultSearchElement":true
            },
            {
              "Name":"Vintner",
              "@Search.defaultSearchElement":true,
              "@Search.fuzzinessThreshold":0.7
            },
            {
              "Name":"Name",
              "@Search.defaultSearchElement":true,
              "@Search.fuzzinessThreshold":0.7
            },
            {
              "Name":"Vintage",
              "@EnterpriseSearch.usageMode":[
                "AUTO_FACET"
              ]
            },
            {
              "Name":"TypeofGrape",
              "@Search.defaultSearchElement":true,
              "@Search.fuzzinessThreshold":0.7,
              "@EnterpriseSearch.usageMode":[
                "AUTO_FACET"
              ]
            },
            {
              "Name":"Color",
              "@EnterpriseSearch.usageMode":[
                "AUTO_FACET"
              ]
            },
            {
              "Name":"Aroma",
              "@Search.defaultSearchElement":true,
              "@Search.fuzzinessThreshold":0.7
            },
            {
              "Name":"Quality",
              "@EnterpriseSearch.usageMode":[
                "AUTO_FACET"
              ]
            },
            {
              "Name":"Content",
              "@EnterpriseSearch.usageMode":[
                "AUTO_FACET"
              ]
            },
            {
              "Name":"Price",
              "@EnterpriseSearch.usageMode":[
                "AUTO_FACET"
              ]
            },
            {
              "Name":"Award",
              "@EnterpriseSearch.usageMode":[
                "AUTO_FACET"
              ]
            }
          ]
        }
      }
    }
]
```

3.4.3 Updating or Creating a Configuration with Method 'PUT'

To update existing configurations, the same call is used as when creating a configuration with method `PUT`.

The existing configuration is then deleted, and the complete configuration has to be specified again. It is not possible to overwrite parts of the configuration. If the configuration to be updated does not exist, it is created ("upsert").

If the configuration is created without errors, an empty row in the response table is returned. If errors occur, an error message is returned in the corresponding row of the response table.

3.4.4 Deleting a Configuration with Method 'DELETE'

The example below shows how to delete an existing configuration. Method `DELETE` is used, and the only other information given in the URI is the schema name and the name of the database object to be deleted.

Only the configuration is deleted. The database object (view or table function) is not deleted or changed.

If the configuration can be deleted without errors, an empty response is returned.

> **Note**
>
> Search configurations are not deleted when the database object or the database schema are deleted. Make sure that configurations are deleted by calling `sys.esh_config()`.

Example: OData metadata document that deletes a search configuration

Sample Code

```
[
  {
    "uri":"~/$metadata/EntitySets(DEVELOP%2FCustomer)",
    "method":"DELETE",
    "content":{
    }
  }
]
```

3.4.5 Privileges

To call `sys.esh_config()`, write or delete configurations, you need certain privileges.

For all calls to `sys.esh_config()`, the database user needs the `EXECUTE` privilege on built-in procedure `sys.esh_config`.

When `sys.esh_config()` is called with method `POST`, `PUT`, or `DELETE`, the caller of `sys.esh_config()` has to be the owner of the database schema or needs the `CREATE ANY` privilege on the database schema.

A `DELETE` call is also possible if the database schema does not exist, so it is possible to delete configurations after the schema has been deleted.

In all other cases, the call to `sys.esh_config()` fails because of missing privileges.

3.4.6 Annotations for sys.esh_config()

The built-in procedure `sys.esh_config()` supports various annotations. These are described in the chapters below.

3.4.6.1 Entity Type Annotations

The table shows all annotations that are available for an entity type.

Table 21: Entity Type Annotations for @Search

Annotation	Allowed Values	Description
`@Search.searchable`	true, false	The annotion has to be **true** to enable `@Search` annotations. At least one column has to be defined as `defaultSearchElement`.

Annotation	Allowed Values	Description
`@Search.configurationID`	<= 5000 characters	Can be set to an application-specific value that is returned in the metadata document. This annotation is used to set, a time-stamp, a version number or a git commit hash for example that identifies the version of the configuration. When getting the metadata document, this value can be used to verify if the configuration stored in the database is the same as the version expected by the application.

Table 22: Entity Type Annotations for @EnterpriseSearch

Annotation	Allowed Values	Description
`@EnterpriseSearch.enabled`	true, false	The annotion has to be **true**. The view has to be defined as `@Search.searchable` and needs a valid anchor key definition.
`@EnterpriseSearch.fieldGroupForSearchQuery`	-	A field group defines a set of view columns that belong together. The field group is used in the query language to restrict the search for a search term to the columns of the field group. An example is the field group '`name`' that contains the columns '`firstName`', '`lastName`', '`companyName`'.
`@EnterpriseSearch.fieldGroupForSearchQuery.Elements`	array of strings	Contains a list of column names. This defines the view columns that belong to the field group.
`@EnterpriseSearch.fieldGroupForSearchQuery.Name`	string	Defines the name of a field group. The name is used in the query language as a reference to the field group.

Table 23: Entity Type Annotations for @Aggregation

Annotation	Allowed Values	Description
`@Aggregation.LeveledHierarchy`	-	Defines a leveled hierarchy that is used to define leveled facets. Examples of leveled hierarchies: • year, month, date • country, region, city
`@Aggregation.LeveledHierarchy.Value`	array of strings	Contains the names of the properties that define the specific hierarchy levels. The sequence in which the properties are specified also defines which level the properties appear on in the hierarchy. **Note** The elements of the leveled hierarchy have to be annotated with `"@EnterpriseSearch.usageMode": "AUTO_FACET"`. Otherwise they will not be returned in the search response.
`@Aggregation.LeveledHierarchy.Qualifier`	string	Defines the name of the hierarchy.

Example

Sample Code

```
[
  {
    "uri":    "~/$metadata/EntitySets",
    "method": "POST",
    "content":
    {
      "Fullname": "AWARDSSQL/awardssql",
      "EntityType": {
        "@EnterpriseSearch.fieldGroupForSearchQuery": [
          { "Name": "award", "Elements": [ "id", "title", "abstract" ] },
          { "Name": "investigator", "Elements": [ "invFirstName",
"invLastName", "invEmail" ] }
        ],
        "@Aggregation.LeveledHierarchy": [
          { "Qualifier": "instLocation", "Value": [ "instCountry",
"instState", "instCity" ] },
          { "Qualifier": "organization", "Value": [ "orgDirectorate",
"orgDivision" ] }
        ],
        "@Search.searchable": true,
        "@EnterpriseSearch.enabled": true,
        "Properties": [
```

```
            ...
          ]
        }
      }
    }
]
```

3.4.6.2 Property Annotations

The tables below show the annotations that are available for properties (columns) of the entity set.

Table 24: Property Annotations for @Search

Annotation	Possible Values	Description
`@Search.defaultSearchElement`	true, false	Defines a column as freestyle search column. At least one column has to be defined as a default freestyle search column. Searching in views without default freestyle search elements is not supported.
`@Search.fuzzinessThreshold`	0..1	Defines the threshold for a fuzzy search. The default value is `1`. This means that an exact search is performed.
`@Search.ranking`	string possible values: HIGH, MEDIUM, LOW	Defines the ranking weight for a column:`HIGH = 1.0`, `MEDIUM = 0.7`, `LOW = 0.5`.
`@Search.termMappingDictionary`	string	Defines the name of the term mapping table (format: `schemaname.tablename`). Passed to the search option `termMappingTable` of the `CONTAINS()` predicate.
`@Search.termMappingListID`	array of strings	Defines the names of the term `mapping list ids`, passed to the search option `termMappingListId` of the `CONTAINS()` predicate
`@Search.mode`	'ALPHANUM'	Enables alphanumeric searches on (N)VARCHAR columns.

Table 25: Property Annotations for @Enterprise Search

Annotation	Possible Values	Description
`@EnterpriseSearch.client`	true, false	Defines the client column and is used by ABAP applications only. `sys.esh_search()` takes the current client from the session context and adds it to any read access to the database object as an additional where condition. The ABAP application server automatically copies the current client to the session context for the primary database connection and for all secondary database connections.
`@EnterpriseSearch.highlighted.enabled`	true, false	Enables highlighting for a column. This annotation must not be used for columns that are part of a 1:n join.
`@EnterpriseSearch.key`	true, false	Defines a column that is part of the anchor key. All anchor key columns have to be part of the same database table. At least one column of the view has to be marked as an anchor key. Searching in views without an anchor key is not supported.
`@EnterpriseSearch.presentationMode`	array of strings possible values: `DETAIL`, `HIDDEN`, `IMAGE`, `SUMMARY`, `THUMBNAIL`, `TITLE`, `NONE`	Only columns with a presentation mode other than `NONE` will be returned in the search result. If presentation modes are not defined, all columns of the view will be returned.
`@EnterpriseSearch.snippets.enabled`	true, false	Enables snippet calculation for a column. This annotation must not be used for columns that are part of a 1:n join.
`@EnterpriseSearch.usageMode`	array of strings possible values: `AUTO_FACET`	`AUTO_FACET`: defines a column as a facet column.

Example

Sample Code

```
[
  {
    "uri":    "~/$metadata/EntitySets",
    "method": "POST",
    "content":
    {
      "Fullname": "AWARDSSQL/awardssql",
        "@Search.searchable": true,
        "@EnterpriseSearch.enabled": true,
```

```
        "Properties": [
          {
            "@EnterpriseSearch.key": true,
            "@EnterpriseSearch.presentationMode": [ "TITLE" ],
            "Name": "id"
          },
          {
            "@EnterpriseSearch.presentationMode": [ "TITLE" ],
            "@Search.defaultSearchElement": true,
            "@Search.fuzzinessThreshold": 0.8,
            "Name": "title"
          },
          {
            "@EnterpriseSearch.presentationMode": [ "TITLE" ],
            "@Search.defaultSearchElement": true,
            "@Search.fuzzinessThreshold": 0.8,
            "Name": "abstract"
          },
          {
            "@EnterpriseSearch.presentationMode": [ "DETAIL" ],
            "@EnterpriseSearch.usageMode": [ "AUTO_FACET" ],
            "@Search.defaultSearchElement": true,
            "Name": "instrument"
          },
          ...
        ]
      }
    }
  }
]
```

3.4.6.3 Definition of Multi-Value Properties

A property (or column) that comes from a 1:n join cannot be returned as a single value in the search response because the column usually contains multiple distinct values for each anchor object.

To get all values in the search response, the property has to be defined as a collection. As a result, it will be returned as an array in the search response.

Table 26:

Metadata Parameter	Possible Values	Description
`IsCollection`	true, false	Defines a column as a multi-value column. This means that the column comes from an 1:n join and that it is returned in the search result as an array. **Note** This multi-value configuration is only allowed for columns that are of a SQL type that can be used in a group by clause.

Sample Code

Multi-Value Example

```
[
  {
    "uri":    "~/$metadata/EntitySets",
    "method": "PUT",
    "content":
    {
      "Fullname": "AWARDSSQL/awardssql",
      "EntityType": {
        "@Search.searchable": true,
        "@EnterpriseSearch.enabled": true,
        "Properties": [
          {
            "@EnterpriseSearch.key": true,
            "@EnterpriseSearch.presentationMode": [ "TITLE" ],
            "Name": "id"
          },
          ...
          {
            "@EnterpriseSearch.presentationMode": [ "DETAIL" ],
            "IsCollection": true,
            "Name": "programReferenceText"
          },
          ...
        ]
      }
    }
  }
]
```

3.4.6.4 Definition of Subobjects

To get back the columns of a complex object that comes from a 1:n join, these columns have to be configured as a subobject.

To define the subobject, all properties of the subobject have to be added to the `sys.esh_config()` call as a collection of a complex type.

Table 27:

Annotation/Metadata Parameter	Possible Values	Description
`@EnterpriseSearchHana.layoutStructuredObject.defaultExpand`	string possible values: `ALL`, `WHY_FOUND`	Defines which subobjects are returned in the search result. `ALL`: all subobjects of an anchor object are returned. `WHY_FOUND`: only the subobject of the anchor object are returned that contain at least one of the search terms. If this annotation is not made, the default value is '`ALL`'.
`@EnterpriseSearchHana.layoutStructuredObject.key`	true, false	Defines if a column is part of the subobject's key. Used to identify the distinct subobjects of an anchor object. If there are no key columns defined for a subobject, all columns are used to identify the distinct subobjects. **Note** All key columns have to be of a SQL type that can be used in a group by clause.
Name	string	Defines the name of the subobject.
IsCollection	true	Defines that the subobject comes from a 1:n join and that there can be multiple subobject instances for a given anchor object.

Sample Code

Subobject Example

```
[
  {
    "uri":     "~/$metadata/EntitySets",
    "method": "PUT",
```

```
    "content":
    {
      "Fullname": "AWARDSSQL/awardssql",
      "EntityType": {
        "@Search.searchable": true,
        "@EnterpriseSearch.enabled": true,
        "Properties": [
          {
            "@EnterpriseSearch.key": true,
            "@EnterpriseSearch.presentationMode": [ "TITLE" ],
            "Name": "id"
          },
          {
            "Name": "programElement",
            "IsCollection":true,
            "@EnterpriseSearchHana.layoutStructuredObject.defaultExpand":
"ALL",
            "Type": {
              "@odata.type": "Meta.ComplexType",
              "Name": "programElementType",
              "Properties": [
                {
                  "@EnterpriseSearch.presentationMode": [ "DETAIL" ],
                  "Name": "programElementCode"
                },
                {
                  "@EnterpriseSearch.presentationMode": [ "DETAIL" ],
                  "@EnterpriseSearch.usageMode": [ "AUTO_FACET" ],
                  "@Search.defaultSearchElement": true,
                  "Name": "programElementText"
                }
              ]
            }
          },
          ...
        ]
      }
    }
  }
]
```

3.4.6.5 Example Call to sys.esh_config()

The following example shows a call to `sys.esh_config()` that can be executed in SAP HANA Studio.

Sample Code

```
call esh_config(
'
[
  {
    "uri":    "~/$metadata/EntitySets",
    "method": "PUT",
    "content":
    {
      "Fullname": "AWARDSSQL/awardssql",
      "EntityType": {
        "@EnterpriseSearch.fieldGroupForSearchQuery": [
          { "Name": "award", "Elements": [ "id", "title", "abstract" ] },
          { "Name": "investigator", "Elements": [ "invFirstName",
"invLastName", "invEmail" ] },
```

```
          { "Name": "organization", "Elements": [ "orgDirectorate",
"orgDivision" ] }
        ],
        "@Aggregation.LeveledHierarchy": [
          { "Qualifier": "instLocation", "Value": [ "instCountry",
"instState", "instCity" ] },
          { "Qualifier": "organization", "Value": [ "orgDirectorate",
"orgDivision" ] }
        ],
        "@Search.searchable": true,
        "@EnterpriseSearch.enabled": true,
        "Properties": [
          {
            "@EnterpriseSearch.key": true,
            "@EnterpriseSearch.presentationMode": [ "TITLE" ],
            "Name": "id"
          },
          {
            "@EnterpriseSearch.presentationMode": [ "TITLE" ],
            "@Search.defaultSearchElement": true,
            "@Search.fuzzinessThreshold": 0.8,
            "Name": "title"
          },
          {
            "@EnterpriseSearch.presentationMode": [ "TITLE" ],
            "@Search.defaultSearchElement": true,
            "@Search.fuzzinessThreshold": 0.8,
            "Name": "abstract"
          },
          {
            "Name": "effectiveDate"
          },
          {
            "Name": "expirationDate"
          },
          {
            "Name": "minAmdLetterDate"
          },
          {
            "Name": "maxAmdLetterDate"
          },
          {
            "@EnterpriseSearch.presentationMode": [ "DETAIL" ],
            "@EnterpriseSearch.usageMode": [ "AUTO_FACET" ],
            "@Search.defaultSearchElement": true,
            "Name": "instrument"
          },
          {
            "@EnterpriseSearch.presentationMode": [ "DETAIL" ],
            "Name": "amount"
          },
          {
            "Name": "arraAmount"
          },
          {
            "@EnterpriseSearch.presentationMode": [ "TITLE" ],
            "@Search.defaultSearchElement": true,
            "Name": "instName"
          },
          {
            "Name": "instPhone"
          },
          {
            "@EnterpriseSearch.presentationMode": [ "DETAIL" ],
            "@EnterpriseSearch.usageMode": [ "AUTO_FACET" ],
            "Name": "instCountry"
          },
          {
```

```
              "Name": "instStateCode"
            },
            {
              "@EnterpriseSearch.presentationMode": [ "DETAIL" ],
              "@EnterpriseSearch.usageMode": [ "AUTO_FACET" ],
              "@Search.defaultSearchElement": true,
              "Name": "instState"
            },
            {
              "Name": "instZip"
            },
            {
              "@EnterpriseSearch.presentationMode": [ "DETAIL" ],
              "@EnterpriseSearch.usageMode": [ "AUTO_FACET" ],
              "@Search.defaultSearchElement": true,
              "@Search.fuzzinessThreshold": 0.8,
              "Name": "instCity"
            },
            {
              "Name": "instStreet"
            },
            {
              "Name": "programOfficer"
            },
            {
              "Name": "orgCode"
            },
            {
              "@EnterpriseSearch.presentationMode": [ "DETAIL" ],
              "@EnterpriseSearch.usageMode": [ "AUTO_FACET" ],
              "@Search.defaultSearchElement": true,
              "@Search.ranking": "LOW",
              "Name": "orgDirectorate"
            },
            {
              "@EnterpriseSearch.presentationMode": [ "DETAIL" ],
              "@EnterpriseSearch.usageMode": [ "AUTO_FACET" ],
              "@Search.defaultSearchElement": true,
              "@Search.ranking": "LOW",
              "Name": "orgDivision"
            },
            {
              "Name": "investigator",
              "IsCollection":true,
              "@EnterpriseSearchHana.layoutStructuredObject.defaultExpand":
"ALL",
              "Type": {
                "@odata.type": "Meta.ComplexType",
                "Name": "investigatorType",
                "Properties": [
                  {
                    "@EnterpriseSearch.presentationMode": [ "DETAIL" ],
                    "@Search.defaultSearchElement": true,
                    "@Search.fuzzinessThreshold": 0.8,
                    "Name": "invFirstName"
                  },
                  {
                    "@EnterpriseSearch.presentationMode": [ "DETAIL" ],
                    "@Search.defaultSearchElement": true,
                    "@Search.fuzzinessThreshold": 0.8,
                    "Name": "invLastName"
                  },
                  {
                    "@EnterpriseSearch.presentationMode": [ "DETAIL" ],
                    "@Search.defaultSearchElement": true,
                    "Name": "invEmail"
                  },
                  {
```

```
                    "@EnterpriseSearch.presentationMode": [ "DETAIL" ],
                    "Name": "invRole"
                  },
                  {
                    "Name": "invStartDate"
                  },
                  {
                    "Name": "invEndDate"
                  }
                ]
              }
            },
            {
              "Name": "programElement",
              "IsCollection":true,
              "@EnterpriseSearchHana.layoutStructuredObject.defaultExpand":
"ALL",
              "Type": {
                "@odata.type": "Meta.ComplexType",
                "Name": "programElementType",
                "Properties": [
                  {
                    "@EnterpriseSearch.presentationMode": [ "DETAIL" ],
                    "Name": "programElementCode"
                  },
                  {
                    "@EnterpriseSearch.presentationMode": [ "DETAIL" ],
                    "@EnterpriseSearch.usageMode": [ "AUTO_FACET" ],
                    "@Search.defaultSearchElement": true,
                    "Name": "programElementText"
                  }
                ]
              }
            },
            {
              "Name": "programReferenceCode"
            },
            {
              "@EnterpriseSearch.presentationMode": [ "DETAIL" ],
              "@EnterpriseSearch.usageMode": [ "AUTO_FACET" ],
              "@Search.defaultSearchElement": true,
              "IsCollection": true,
              "Name": "programReferenceText"
            }
          ]
        }
      }
    }
]
'
, ?);
```

3.4.6.6 Entity Set Annotations for Parameters of Table Functions and Join Views

When using search with table functions or join views, the caller has to pass parameters to the `sys.esh_search()` function. These parameters are used as parameters for the table function call or as constraints for join views.

Table 28:

Annotation	Allowed Values	Description
@Search.constraints		Defines one or more table function parameters or join view constraints. Each parameter has the following attributes: **Name**: The name of the table function parameter or join view constraint. **DefaultValue**: The default value is used if the paramter is not specified in the search URI. **Alias**: Defines an alternative name of the parameter that has to be used in the search URI.

For join views, defining constraints with `sys.esh_search()` is optional. If no constraints are specified in the configuration, the constraints are taken from the join view definition and can still be used in calls to `sys.esh_search()`.

For table functions, all parameters have to be specified with `sys.esh_search()`, including a default value.

Example

The following example shows how parameters are defined in the metadata document that describes a search configuration.

Sample Code

```
[
  {
    "uri": "~/$metadata/EntitySets",
    "method": "POST",
    "content": {
    // This annotation is only needed if a table function with
    // parameters or a join view with constraints is used.
    // If the annotation is missing, no parameters can be passed
    // to the table function or join view.
    "@Search.constraints": [
      {
        // "name" defines the name of the table function parameter
        // or the name of the join view constraint.
        "Name": "par1",
        // a constant default value
        "DefaultValue": "'This is the default value.'"
      },
      {
        "Name": "par2",
        // a function or session variable is used to define the
        // default value.
        "DefaultValue": "CURRENT_DATE"
      }
    ],
      "Fullname": "myschema/myfunction",
      "EntityType": {
        "@Search.searchable": true,
```

```
        "@EnterpriseSearch.enabled": true,
        "Properties": [
          {
            "Name": "AColumn",
            "@Search.defaultSearchElement": true,
            "@EnterpriseSearch.key": true
          },
          {
            "Name": "AnotherColumn",
            "@EnterpriseSearch.usageMode": [
              "AUTO_FACET"
            ]
          }
        ]
      }
    }
  }
]
```

3.4.6.6.1 Default Values

The default value of a parameter is either a constant string or a function that is evaluated when `sys_esh_search()` is called.

Table 29:

Default Value	Description
"'a string'"	The default value of the parameter is 'a string'.
"CURRENT_DATE"	The current date (server time zone, format 'YYYYMMDD'.
"APPLICATIONUSER"	The name of the end user, for example a SAP user name or operating system user.
"LOCALE_SAP"	The single byte language code (SAP language code).
"LOCALE_ISO2"	The two-character language code (as defined by ISO 639-1), taken from the session variable 'LOCALE'. Default value is 'en' if 'LOCALE' is not set.

3.4.6.6.2 Renaming of Parameters

It is possible to rename parameters so that the ABAP client can always use the same set of parameter names regardless of the real name of the table function parameter or join view constraint.

A table function can define a parameter called 'lang' for example, which is defined as an iso language parameter. The calling application defines a parameter 'language_iso' that contains the iso language to be used in the call to `sys.esh_search()`.

Sample Code

Configuration Without Parameter Renaming

```
"@Search.constraints": [
  {
    "Name": "lang",
    "DefaultValue": "'fr'"
  }
]
```

With the configuration above, the call to esh_search has to be performed as follows:

```
esh_search('[ "/MYSCHEMA/MYVIEW(lang='en')" ]', ?);
```

Sample Code

Configuration with Parameter Renaming

```
"@Search.constraints": [
  {
    "Name": "lang",
    "DefaultValue": "'fr'",
    "Alias": "language_iso"
  }
]
```

As soon as the configuration is changed as shown above, the call has to use the alias name of the parameter:

```
esh_search('[ "/MYSCHEMA/MYVIEW(language_iso='en')" ]', ?);
```

3.5 Modeling With Attribute Views

To create search models with attribute views, use search-specific properties in the SAP HANA modeler.

Context

In the SAP HANA modeler, create an attribute view of type standard using the tables that you want to enable for the search. Create joins and add the attributes you want to use for searching and displaying. You can also join additional attributes derived from the text analysis.

Caution

If your business object contains attributes with a 1:n relationship, for example, customer with more than one address joined in from an address table, note the following: If you are using such an attribute as a response attribute for your UI, the same customer might show up several times in the result list and thus, the result count might deviate from the actual number of results.

When activating the view, do not enable or apply any analytic privilege. The user access to search views must be restriced using object privileges instead. For more information, see section *Authorizations for SAP HANA Info Access Users*.

To configure the search methods for certain attributes, proceed as follows:

Procedure

1. In the *Output* pane, select the attributes.
2. In the *Properties* view, choose *Search Properties*.
3. For text attributes, set the *Freestyle Search* property to true (default for text attributes).

   The values of these attributes are searched using a special method for natural text. You get hits based on single words or parts of words.

   > Note
   >
   > This setting is only relevant for search queries. Queries to retrieve suggestions while the user is typing always use all full-text indices. If you set this property to false, the user might select a suggested search term but does not get a result.

4. Set the *Weights for Ranking* to a value between 0.0 and 1.0 (default 0.5).

   This property influences the ranking of items in the results list. The higher the weight of the attribute compared to others in the view, the higher up the list an item with a hit in this attribute is positioned.
5. Decide if you want to enable an error-tolerant search (*Fuzziness* true) or use exact search only (false).

   If you are using the fuzzy search, the *Fuzziness Threshold* property defines the grade of error tolerance for a search on this attribute. The higher the threshold, the more exactly the search terms must match the text to produce a result. Set the threshold to one of the predefined values (default 0.8).

Related Information

FUZZY Search [page 149]

4 Accessing Data Using Full-Text Search

You can access your data using either ODATA services or SQL statements.

Context

There are several ways to access your data using full-text search:

- Full-text search with ODATA
  Use this option if your search models are based on CDS annotations (XS Classic).
- Federated full-text search with built-in procedure `sys.esh_search()`
  Use this option if your search models are defined with the built-in procedure `sys.esh_config()` or if they are based on CDS annotations (XS Advanced).
- Full-text search with SQL
  Use this option if you modeled your search models with attribute views.

Related Information

Full-Text Search with OData [page 83]
Federated Full-Text Search with Built-In Procedure sys.esh_search() [page 88]
Full-Text Search with SQL [page 138]

4.1 Full-Text Search with OData

You define OData services to your views to execute search requests on the data.

> **Note**
>
> This chapter is only valid for developments with XS Classic.

As a prerequisite for exposing information via OData to applications using SAP HANA XS Classic, you have to define database views that provide the data you want to search for with the required granularity.

Having defined the views, you can now create an OData service definition. This is a file that you use to specify which database views or tables are exposed as OData collections.

An OData search service for SAP HANA XS is defined in a text file with the file suffix `.xsodata`, for example, `OdataSearch.xsodata`.

The file must contain the entry `service {}`. You use the service definition to expose objects (views) in the database catalog. You have to define the key columns of the view.

Examples for OData service definitions: `OdataSearch.xsodata`

```
service
{
  "develop.mypackage::mypackage.mysqlview" as "mysqlview_search"
  key ("BP_NUMBER", "ADDRESS_NUMBER")
  "develop.bporg::mypackage.mysqlview" as "mysqlview_search_genkey"
  key generate local "GeneratedLocalKey"
}
```

This exampe file includes two service definitions: `mysearchdef` and `mysearchdef_genkey`.

After you have defined the search services, you can call the services by their OData service URL. As soon as one of the custom query options for search (`search` or `facets`) is part of the URL, the OData service calls the search runtime to create the expected search result.

Examples for OData service calls:

```
http://mo-bfafcf4d8.mo.sap.corp:8000/develop/bporg/bporg.xsodata/bporg?
search=electronics&$format=json&facets=all
```

```
http://mo-bfafcf4d8.mo.sap.corp:8000/develop/bporg/bporg.xsodata/bporg?
search=electronics&$top=10&$format=json&$select=NAME_LAST,NAME_FIRST,SEARCH_TERM
```

```
http://mo-bfafcf4d8.mo.sap.corp:8000/develop/bporg/bporg.xsodata/bporg?
search=federal%20OR%20delta&$top=10&$format=json&$inlinecount=allpages
```

Related Information

Custom Query Option 'search' [page 84]
Custom Query Option 'facets' [page 85]
Custom Query Option 'facetlimit' [page 85]
Custom Query Option 'estimate' [page 86]
Custom Query Option 'wherefound' [page 86]

4.1.1 Custom Query Option 'search'

The search term that is given with the `search` option is passed to the `CONTAINS()` predicate.

The syntax of the search term is the syntax that is defined for the `CONTAINS()` predicate and not the syntax that is defined for the OData version 4 system query option `$search`.

When using the `search` option on a CDS view with `@Search` annotations, the search properties defined with these annotations are used.

When using the `@EnterpriseSearch.key` annotation, freestyle search is done with query parts, so all rows belonging to the same anchor object are treated as one large object that is searched.

4.1.2 Custom Query Option 'facets'

The `facets` option defines how many facets are calculated.

With the option you can specify the number of facets:

Table 30:

Value	Description
`facets=0`	No facets are returned. Default, when the option is not given.
`facets=n`(n > 0)	n facets are returned. This always is a subset of the columns marked with the `#AUTO_FACET` annotation.
`facets=all`	Facets for all columns marked with the `#AUTO_FACET` annotation are returned.
`facets = column1,column2,...,columnn`	Facets for the columns given are returned. All columns have to be marked with the `#AUTO_FACET` annotation.

For each facet, the top 10 values are returned if the custom query option `facetlimit` is not used.

When using the `@EnterpriseSearch.key` annotation to define the keys of the anchor table, facets count distinct anchor keys instead of counting rows.

> **Note**
>
> When using facets, you have to specify `$format=json` in the search URI. There is only preliminary support for `$format=atom`.

Related Information

Custom Query Option 'facetlimit' [page 85]

4.1.3 Custom Query Option 'facetlimit'

Defines how many values for each facet are returned.

Table 31:

Value	Description
`facetlimit=n`	Defines the number of values for each facet to be returned. Default: 10

4.1.4 Custom Query Option 'estimate'

When using this option in combination with `$inlinecount=allpages`, an estimated result count is returned in some cases to reduce response times.

> **Note**
>
> The option is without effect if facets are requested.
>
> Using `estimate` without `$inlinecount=allpages` returns an error.

Table 32:

Value	Description
`estimate=true`	The search may return an estimated result count instead of an exact count. This could help to reduce response times.
`estimate=false`	The search will return an exact result count. Switched off by default.

4.1.5 Custom Query Option 'wherefound'

This option returns the output of the SQL function `WHY_FOUND()` for each row in the response.

The "where found" information is returned as annotation `@com.sap.vocabularies.Search.v1.WhereFound`.

> **Note**
>
> `wherefound` can only be used if a search term is given using the custom query option `search`.

Table 33:

Value	Description
`wherefound=true`	Returns the "where found" information.
`wherefound=false`	Switched off by default.

4.1.6 OData Features and Limitations

System query options can be used in combination with custom query options defined for search, as well as features offered by SAP HANA XS.Certain limitations apply however.

OData Features Supported by Search

The search runtime and the custom query options `search` and `facets` work for OData resource paths that access a collection. In other words, a valid URI is, for example, `.../<service>.xsodata/<collection>?search=<search term>`.

Optionally, `$count` can be used to count the number of search results (`.../<service>.xsodata/<collection/$count?search=<search term>`).

The following OData v2 system query options can be used in combination with the custom query options defined for search:

- `$top`
- `$skip`
- `$format` (facets are supported for `$format=json` only)
- `$select`
- `$filter`
- `$inlinecount`
- `$orderby`

The following XS OData features are supported by the search runtime (see the SAP HANA Developer Guide for a description of these features):

- OData namespace definition
  OData key specification (existing key properties and generated local keys).
- OData property projection (when using property projection, all freestyle columns of the database object will be searched, including columns that are not part of the projection).

OData Limitations

The following OData v2 system query options cannot be used in combination with the custom query options defined for search:

- `$expand XS`

The following OData features are not supported by the search runtime (see the SAP HANA Developer Guide for a description of these features):

- OData associations and navigation properties (these can be defined for the XS OData service, but navigation properties cannot be used as part of a search URI)
- OData aggregation

- OData parameter entity sets

The following OData features are not supported:

- `$metadata` does not return metadata for search configuration and facets

4.2 Federated Full-Text Search with Built-In Procedure sys.esh_search()

With the built-in procedure `sys.esh_search()`, it is possible to use an existing SQL connection instead of an XS OData service, such as an existing ABAP SQL connection. This means that the customer does not have to configure the http connection, and that any configured search view does not have to be exposed as an OData service, which would make it necessary to create SAP HANA repository objects like `.xsodata`, `.xsapp` and `.xsaccess`. Instead, SQL privileges are used to grant access to the search runtime.

`sys.esh_search()` provides an interface similar to the OData interface available with an XS OData service.

The following example shows the syntax of a search request:

```
call esh_search('[ "/$all?$filter=Search.search(query=''mysearchterm'')" ]', ?);
```

> **Note**
>
> When calling `esh_search()` in the SQL console of SAP HANA Studio, all single quotes inside the request have to be doubled because they are used inside a SQL string.

4.2.1 Interface of sys.esh_search()

A NCLOB container is used to pass a search request to the procedure.

The response is a table that contains a NCLOB containing the search result of the search request.

The interface of `sys.esh_search()` looks as follows:

Source Code

```
CREATE PROCEDURE sys.esh_search
(
    IN  request  NCLOB,
    OUT response TABLE ( response NCLOB )
)
AS BUILTIN;
```

4.2.1.1 Error Handling

If the request object does not contain a valid JSON array, a SQL exception is returned.

Examples of errors that can occur are a malformed JSON or missing "uri", "method", or "content" elements.

For a federated search request, the following types of errors can occur:

Search Request cannot be processed

If the search request is malformed and cannot be processed, an error message similar to the following will be returned:

Source Code

```
{
  "error":{
    "code": "9620044",
    "message": "exception 9620044: Function call all() not allowed in
federation"
  }
}
```

Search Request cannot be processed for one or a few Views

If a federated search can be processed for most but not all views, the `@com.sap.vocabularies.Search.v1.SearchStatistics` annotation of the search response will contain information for each individual view in which an error occurred:

Source Code

```
{
  "value": [...],
  "@com.sap.vocabularies.Search.v1.SearchStatistics":{
    "ConnectorStatistics":[
      {
        "Name":"CUSTOMER",
        "Schema":"VIEW1",
        "StatusCode":200
      },
      {
        "Name":"COMPANY",
        "Schema":"VIEW1",
        "StatusCode":500,
        "error":{
          "code":"9620042",
          "message":"exception 9620042: attribute 'firstname' does not exist
in view 'COMPANY'"
        }
      }
    ]
  }
}
```

4.2.2 Method 'GET' - $metadata Call

It is possible to get the metadata of all views that the user is allowed to search in.

> **Note**
>
> The `$metadata` call uses the `sys.esh_search()` procedure. With `sys.esh_config()`, the `$metadata` call is not possible.

Example: Get metadata document for all views (by calling `sys.esh_search()`)

Sample Code

```
[
  {
    "URI": [
      "/v5/$metadata"
    ]
  }
]
```

As a result, the caller gets the metadata for all views that fulfill the following conditions:

1. The user has the `SELECT` privilege on the view.
2. The view is marked as `@EnterpriseSearch.searchable: true` in the configuration.

The metadata call is done by passing the URI `/$metadata` to `sys.esh_search()` as shown in the following example:

Sample Code

`sys.esh_search()` metadata call

```
call esh_search('[ { "URI": [ "/v5/$metadata" ] } ]', ?);
```

It is also possible to get the metadata for a single schema by only using the URI `/<SCHEMA_NAME>/$metadata`, as shown below for schema 'AWARDS'.

Sample Code

`sys.esh_search()`

```
call esh_search('[ { "URI": [ "/v5/AWARDS/$metadata" ] } ]', ?);
```

Version-Dependent Features of Metadata and Handling of Incompatible Changes

With newer versions of SAP HANA, the metadata information may be changed in a way that is incompatible with previous versions. To avoid existing applications failing, the new metadata information is available only if an application explicitly calls the new version of the `sys.esh_search()` API.

The following example shows a metadata call to `sys.esh_search()` that uses all new features of API version 5.

Sample Code

Metadata Call in the SQL Console

```
call esh_search('[ { "URI": [ "/v5/$metadata" ] } ]', ?);
```

The example below shows how to call `sys.esh_search()` with features of API version 3 (plus additional compatible changes introduced by later versions).

Sample Code

Metadata Call in the SQL Console

```
call esh_search('[ "/$metadata" ]', ?);
```

OData Identifiers

Each searchable view is returned as an entity set and an entity type in the $metadata document.

The name of the entity set equals the odata identifier for the view. The name of the entity type is derived from the odata identifier (by adding the suffix `Type`).

By default, the odata identifier is calculated using the name of the 'real' database object based on the following rules:

1. Schema names are ignored.
2. Package name and context are removed from CDS view names.
   Example: the OData identifier of CDS view "p1.p2.p3::ctxt.myView" is "myView"
3. All characters that are not allowed in an OData identifier are removed.
   Example: the OData identifier of ESH view "S_ARTICLE_H~~ARTICLE" is "S_ARTICLE_HARTICLE"

For a definition of valid characters for OData identifiers: see 'odataIdentifier' in the OData v4 ABNF construction rules.

Alternatively, the Odata identifier can be defined in the search configuration by using the annotation `@EnterpriseSearchHana.identifier`.

Anchor Key Definition and Data Type Information

The metadata call returns the EDM types of all columns. The mapping of SQL types to EDM types is shown in the metadata example below.

- A property, named `C_DATATYPE` in this example, is defined using SQL type `DATATYPE`.
  - `C_ST_POINT_xxxx` and `C_ST_GEOMETRY_xxxx` use SQL type `ST_POINT` and `ST_GEOMETRY` with the spatial reference identifier `xxxx`.
- The data type mapping includes properties for `type`, `maxLength`, `precision`, `scale`, and `spatial reference identifier` (SRID).
- Columns that are defined as `NOT NULL` get the property `Nullable="false"`.
- Columns that define a default value get the property `DefaultValue`.

The section `<Key>` contains the columns of the anchor key.

Sample Code

Mapping of SQL Types

```
<?xml version="1.0" encoding="UTF-8"?>
<edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx" Version="4.0">
  <edmx:DataServices>
    <Schema xmlns="http://docs.oasis-open.org/odata/ns/edm" Namespace="esh">
      <EntityType Name="myviewType">
        <Key>
          <PropertyRef Name="ID_INTEGER" />
        </Key>
        <Property Name="ID_INTEGER" Type="Edm.Int32" Nullable="false" />
        <Property Name="C_VARCHAR" Type="Edm.String" DefaultValue="abc"
MaxLength="100" />
        <Property Name="C_NVARCHAR" Type="Edm.String" Nullable="false"
MaxLength="100" />
        <Property Name="C_ALPHANUM" Type="Edm.String" MaxLength="10" />
        <Property Name="C_SHORTTEXT" Type="Edm.String" MaxLength="100" />
        <Property Name="C_VARBINARY" Type="Edm.Binary" />
        <Property Name="C_TINYINT" Type="Edm.Byte" />
        <Property Name="C_SMALLINT" Type="Edm.Int16" />
        <Property Name="C_INTEGER" Type="Edm.Int32" />
        <Property Name="C_BIGINT" Type="Edm.Int64" />
        <Property Name="C_SMALLDECIMAL" Type="Edm.Decimal" Precision="16" />
        <Property Name="C_DECIMAL" Type="Edm.Decimal" Precision="34" />
        <Property Name="C_DECIMAL_5_4" Type="Edm.Decimal" Precision="5"
Scale="4" />
        <Property Name="C_DECIMAL_34_0" Type="Edm.Decimal" Precision="34"
Scale="0" />
        <Property Name="C_DECIMAL_34_33" Type="Edm.Decimal" Precision="34"
Scale="33" />
        <Property Name="C_REAL" Type="Edm.Single" />
        <Property Name="C_DOUBLE" Type="Edm.Double" />
        <Property Name="C_DATE" Type="Edm.DateTime" />
        <Property Name="C_TIME" Type="Edm.Time" />
        <Property Name="C_SECONDDATE" Type="Edm.DateTime" />
        <Property Name="C_TIMESTAMP" Type="Edm.DateTime" />
        <Property Name="C_BLOB" Type="Edm.Binary" />
        <Property Name="C_CLOB" Type="Edm.String" MaxLength="2147483647" />
        <Property Name="C_NCLOB" Type="Edm.String" MaxLength="2147483647" />
        <Property Name="C_TEXT" Type="Edm.String" MaxLength="2147483647" />
        <Property Name="C_BINTEXT" Type="Edm.String" MaxLength="2147483647" />
        <Property Name="C_ST_POINT_4326" Type="Edm.GeographyPoint"
SRID="4326" />
        <Property Name="C_ST_POINT_1000004326" Type="Edm.GeometryPoint"
SRID="1000004326" />
```

```
        <Property Name="C_ST_GEOMETRY_4326" Type="Edm.GeographyPoint"
SRID="4326" />
        <Property Name="C_ST_GEOMETRY_1000004326" Type="Edm.GeometryPoint"
SRID="1000004326" />
      </EntityType>
      <EntityContainer Name="esh">
        <EntitySet Name="myview" EntityType="esh.myviewType" />
      </EntityContainer>
    </Schema>
  </edmx:DataServices>
</edmx:Edmx>
```

Search Annotations

For each property of an entity type, the values of the search annotations are returned as defined in the search model.

In addition to the annotations described for `sys.esh_config()` and CDS, the following annotations are returned by the metadata service. These annotations cannot be used with `sys.esh_config()` or CDS:

Table 34:

Annotation	Values	Description
`EnterpriseSearch.displayOrder`	integer >= 0	The display order defines the order of the columns the UI will use to display a search result. For configurations defined with `sys.esh_config()`, the value of the corresponding annotation used in the `sys.esh_config()` call is returned. For configurations defined using CDS, the display order is defined by the sequence of the columns in the view definition.
`EnterpriseSearchHana.supportsTextSearch`	`true`, `false`	`true`, if a column supports full-text search.
`SAP.Common.Label`	string	Label for a column name. The column name is returned as label for all languages.

The following example shows a metadata document that contains annotations:

Sample Code

Metadata Document with Search Annotations

```xml
<?xml version="1.0" encoding="UTF-8"?>
<edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx" Version="4.0">
  <edmx:Reference Uri="http://vecocomst.dhcp.wdf.sap.corp:1080/coco/
vocabularies/Common.xml">
    <edmx:Include Alias="SAP.Common"
Namespace="com.sap.vocabularies.Common.v1"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://xyz.dhcp.wdf.sap.corp:1080/coco/vocabularies/
Search.xml">
    <edmx:Include Alias="Search" Namespace="com.sap.vocabularies.Search.v1"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://xyz.dhcp.wdf.sap.corp:1080/coco/vocabularies/
EnterpriseSearch.xml">
    <edmx:Include Alias="EnterpriseSearch"
Namespace="com.sap.vocabularies.EnterpriseSearch.v1"/>
  </edmx:Reference>
  <edmx:DataServices>
    <Schema xmlns="http://docs.oasis-open.org/odata/ns/edm" Namespace="esh">
      <EntityType Name="awardsType">
        <Annotation Term="Search.searchable" Bool="true" />
        <Annotation Term="EnterpriseSearch.enabled" Bool="true" />
        <Key>
          <PropertyRef Name="id" />
        </Key>
        <Property Name="id" Type="Edm.Int32" Nullable="false">
          <Annotation Term="SAP.Common.Label" String="id" />
          <Annotation Term="EnterpriseSearch.displayOrder" Int="0" />
          <Annotation Term="EnterpriseSearch.key" Bool="true" />
          <Annotation Term="EnterpriseSearch.presentationMode">
            <Collection>
              <String>TITLE</String>
            </Collection>
          </Annotation>
        </Property>
        <Property Name="abstract" Type="Edm.String" MaxLength="2147483647">
          <Annotation Term="SAP.Common.Label" String="abstract" />
          <Annotation Term="EnterpriseSearch.displayOrder" Int="2" />
          <Annotation Term="EnterpriseSearch.presentationMode">
            <Collection>
              <String>TITLE</String>
            </Collection>
          </Annotation>
          <Annotation Term="Search.defaultSearchElement" Bool="true" />
          <Annotation Term="Search.fuzzinessThreshold" Decimal="0.8" />
        </Property>
        <Property Name="title" Type="Edm.String" MaxLength="500">
          <Annotation Term="SAP.Common.Label" String="title" />
          <Annotation Term="EnterpriseSearch.displayOrder" Int="1" />
          <Annotation Term="EnterpriseSearch.presentationMode">
            <Collection>
              <String>TITLE</String>
            </Collection>
          </Annotation>
          <Annotation Term="Search.defaultSearchElement" Bool="true" />
          <Annotation Term="Search.fuzzinessThreshold" Decimal="0.8" />
        </Property>
      </EntityType>
      <EntityContainer Name="esh">
        <EntitySet Name="awards" EntityType="esh.awardsType">
          <Annotation String="awards" Term="SAP.Common.Label" />
        </EntitySet>
      </EntityContainer>
    </Schema>
  </edmx:DataServices>
</edmx:Edmx>
```

Multi-Values and Subobjects

The following example shows the metadata for multi-values and subobjects:

Sample Code

Metadata for Multi-Values and Subobjects

```
...
    <Schema xmlns="http://docs.oasis-open.org/odata/ns/edm" Namespace="esh">
      <EntityType Name="awardsType">
        <Annotation Term="Search.searchable" Bool="true" />
        <Annotation Term="EnterpriseSearch.enabled" Bool="true" />
        <Key>
          <PropertyRef Name="id" />
        </Key>
        <Property Name="id" Type="Edm.Int32" Nullable="false">
          <Annotation Term="SAP.Common.Label" String="id" />
          <Annotation Term="EnterpriseSearch.displayOrder" Int="0" />
          <Annotation Term="EnterpriseSearch.key" Bool="true" />
          <Annotation Term="EnterpriseSearch.presentationMode">
            <Collection>
              <String>TITLE</String>
            </Collection>
          </Annotation>
        </Property>
        ...
        <Property Name="programReferenceText" Type="Collection(Edm.String)"
MaxLength="100">
          <Annotation Term="SAP.Common.Label" String="programReferenceText" />
          <Annotation Term="EnterpriseSearch.displayOrder" Int="17" />
          <Annotation Term="EnterpriseSearch.presentationMode">
            <Collection>
              <String>DETAIL</String>
            </Collection>
          </Annotation>
        </Property>
        ...
        <Property Name="programElement"
Type="Collection(esh.programElementType)">
          <Annotation
Term="EnterpriseSearchHana.layoutStructuredObject.defaultExpand"
String="ALL" />
        </Property>
        ...
      </EntityType>
      <ComplexType Type="programElementType">
        <Property Name="programElementCode" Type="Edm.String" MaxLength="4"
Nullable="false">
          <Annotation Term="EnterpriseSearch.displayOrder" Int="15" />
          <Annotation Term="EnterpriseSearch.presentationMode">
            <Collection>
              <String>DETAIL</String>
            </Collection>
          </Annotation>
          <Annotation Term="EnterpriseSearchHana.layoutStructuredObject.key"
Bool="true" />
        </Property>
        <Property Name="programElementText" Type="Edm.String" MaxLength="100"
Nullable="false">
          <Annotation Term="EnterpriseSearch.displayOrder" Int="16" />
          <Annotation Term="EnterpriseSearch.presentationMode">
            <Collection>
              <String>DETAIL</String>
            </Collection>
          </Annotation>
```

```
            <Annotation Term="Search.defaultSearchElement" Bool="true" />
            <Annotation Term="EnterpriseSearchHana.layoutStructuredObject.key"
Bool="true" />
        </Property>
      </ComplexType>
      <EntityContainer Name="esh">
        <EntitySet Name="awards" EntityType="esh.awardsType">
          <Annotation String="awards" Term="SAP.Common.Label" />
        </EntitySet>
      </EntityContainer>
    </Schema>
...
```

Leveled Hierarchies

The following example shows the metadata for leveled hierarchies.

Sample Code

Metadata for Leveled Hierarchies

```
...
    <Schema xmlns="http://docs.oasis-open.org/odata/ns/edm" Namespace="esh">
      <EntityType Name="awardsType">
        <Annotation Term="Search.searchable" Bool="true" />
        <Annotation Term="EnterpriseSearch.enabled" Bool="true" />
        <Key>
          <PropertyRef Name="id" />
        </Key>
        <Annotation Term="Aggregation.LeveledHierarchy"
Qualifier="instLocation">
          <Collection>
            <PropertyPath>instCountry</PropertyPath>
            <PropertyPath>instState</PropertyPath>
            <PropertyPath>instCity</PropertyPath>
          </Collection>
        </Annotation>
        <Annotation Term="Aggregation.LeveledHierarchy"
Qualifier="organization">
          <Collection>
            <PropertyPath>orgDirectorate</PropertyPath>
            <PropertyPath>orgDivision</PropertyPath>
          </Collection>
        </Annotation>
        <Property Name="id" Type="Edm.Int32" Nullable="false">
          ...
        </Property>
        ...
      </EntityType>
      ...
    </Schema>
...
```

Field Groups

The following example shows the metadata for field groups.

Sample Code

Metadata for Field Groups

```
...
    <Schema xmlns="http://docs.oasis-open.org/odata/ns/edm" Namespace="esh">
      <EntityType Name="awardsType">
        <Annotation Term="Search.searchable" Bool="true" />
        <Annotation Term="EnterpriseSearch.enabled" Bool="true" />
        <Key>
          <PropertyRef Name="id" />
        </Key>
        <Annotation Term="EnterpriseSearch.fieldGroupForSearchQuery">
          <Collection>
            <Record>
              <PropertyValue Property="Name">award</PropertyValue>
              <PropertyValue Property="Elements">
                <Collection>
                  <String>abstract</String>
                  <String>id</String>
                  <String>title</String>
                </Collection>
              </PropertyValue>
            </Record>
            <Record>
              <PropertyValue Property="Name">investigator</PropertyValue>
              <PropertyValue Property="Elements">
                <Collection>
                  <String>invEmail</String>
                  <String>invFirstName</String>
                  <String>invLastName</String>
                </Collection>
              </PropertyValue>
            </Record>
          </Collection>
        </Annotation>
        <Property Name="id" Type="Edm.Int32" Nullable="false">
          ...
        </Property>
        ...
      </EntityType>
      ...
    </Schema>
...
```

Returning Metadata for more than one View

If the metadata of more than one searchable view is returned, the result contains one '<EntityType>' element and one '<EntitySet>' element for each view. The views may be created in different database schemas, but they are returned from a single call to the `$metadata` service.

All views of all database schemas are returned in a single '<Schema>' element. The name given in the '<Schema>' element is not related to the name of the database schema.

Sample Code

Metadata for more than one view

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx" Version="4.0">
  <edmx:DataServices>
    <Schema xmlns="http://docs.oasis-open.org/odata/ns/edm" Namespace="esh">
      <EntityType Name="firstViewType">
        <Key>
          <PropertyRef Name="ID_INTEGER" />
        </Key>
        <Property Name="ID_INTEGER" Type="Edm.Int32" Nullable="false" />
        <Property Name="C_VARCHAR" Type="Edm.String" DefaultValue="abc"
MaxLength="100" />
      </EntityType>
      <EntityType Name="secondViewType">
        <Key>
          <PropertyRef Name="ID_INTEGER" />
        </Key>
        <Property Name="ID_INTEGER" Type="Edm.Int32" Nullable="false" />
        <Property Name="C_VARCHAR" Type="Edm.String" DefaultValue="abc"
MaxLength="100" />
      </EntityType>
      <EntityType Name="anotherViewType">
        <Key>
          <PropertyRef Name="ID_INTEGER" />
        </Key>
        <Property Name="ID_INTEGER" Type="Edm.Int32" Nullable="false" />
        <Property Name="C_VARCHAR" Type="Edm.String" DefaultValue="abc"
MaxLength="100" />
      </EntityType>
      <EntityContainer Name="esh">
        <EntitySet Name="firstView" EntityType="esh.firstViewType" />
        <EntitySet Name="secondView" EntityType="esh.secondViewType" />
        <EntitySet Name="anotherView" EntityType="esh.anotherViewType" />
      </EntityContainer>
    </Schema>
  </edmx:DataServices>
</edmx:Edmx>
```

4.2.3 Supported SQL Types

The query language supports all SQL types that can be used in a call to the `CONTAINS()` predicate.

(N)VARCHAR and Text Types

Usually each token of the search input is searched individually in the different columns, as the query language defines all `AND`, `OR`, and `NOT` operations as set based operations by default. This means that results would be missing when searching in string type columns (`VARCHAR` and `NVARCHAR`) or when searching in a text column with an active term mapping configuration.

To receive the search results as expected by the user, additional searches with combined adjacent search terms have to be implemented.

For example, the query **`SAP SE`** defines two searches for '**SAP**' and for '**SE**' in a company name column of type `NVARCHAR` (without a full-text index). As this does not find a company named '**SAP SE**', an additional search is done that combines both terms and searches for '**SAP SE**' in the `NVARCHAR` column.

Search terms given in the query language expression are combined if they are adjacent, if there are no other operators given between the search terms, and if there are no modifiers like fuzzy threshold, ranking weight, or column selectors.

Examples:

- '**hana database**' searches for '**hana**' and '**database**' or for '**hana database**'.
- '**hana** `AND` **database**' searches for '**hana**' and '**database**' only. Because of the `AND` operator the search terms are not considered to be adjacent anymore.
- '**hana database~0.7**' searches for '**hana**' and '**database~0.7**' only because of the fuzzy threshold given for '**database**'.

Date Types

When searching in columns with date types (SQL types `DATE`, `SECONDDATE`, `TIMESTAMP`) or in columns that contain dates (SQL types `NVARCHAR` and `DECIMAL` with `@Semantics` annotation for dates), the following date formats are supported by the query language:

- `YYYYMMDD` (ISO)
- `YYYY-MM-DD` (W3C)
- `MM/DD/YYYY` (US)
- `DD.MM.YYYY` (German)

Spatial Data

It is possible to define filters on spatial data types `ST_POINT` and `ST_GEOMETRY`.

The attribute conditions `:WITHIN:`, `:COVERED_BY:` and `:INTERSECTS:` are available and are internally mapped to the SQL functions `ST_Within`, `ST_CoveredBy`, and `ST_Intersects`.

The operators accept the name of a column of type `ST_POINT` or `ST_GEOMETRY` on the left side and a spatial object in well-known text format (WKT) on the right side. Accepted object types are Point, MultiPoint, LineString, MultiLineString, Polygon, MultiPolygon, CircularString, and GeometryCollection.

The coordinates in the WKT representation have to be given in the spatial reference system of the database column.

Sample Code

Spatial Filter Examples

```
location:COVERED_BY:POLYGON((13.4 52.5, 13.5 52.5, 13.5 52.6, 13.4 52.6, 13.4
52.5))
envelope:INTERSECTS:LINESTRING(13.123 52.543, 13.899 52.789)
```

4.2.3.1 Search with Partial Dates

It is possible to search with incomplete dates, either with a freestyle search over all columns or with a search in a single column.

Sample Code

Search with an Incomplete Date

```
// search for all awards related to 'Adams' with any date in 2015
SCOPE:awards adams 2015
// search for all awards related to 'Adams' ending in April 2016
SCOPE:awards adams expirationDate:2016-04
```

Searching with an incomplete date is supported for all date and timestamp columns (see below). When searching with a valid date pattern in any of these columns, all dates or timestamps that match the given pattern are returned.

Columns are treated as date columns if they are of SQL type `DATE`, or `(N)VARCHAR` with any of the `@Semantics` annotations for dates.

Columns are treated as timestamp columns if they are of SQL type `TIMESTAMP`, `SECONDDATE`, or `DECIMAL` with any of the `@Semantics` annotations for dates.

Supported date formats are:

- ISO: `YYYYMMDD`, `YYYYMM`, `YYYYMM*`
- W3C: `YYYY-MM-DD`, `YYYY-MM`, `YYYY-MM-*`
- US: `MM/DD/YYYY`, `MM/*/YYYY`
- German: `DD.MM.YYYY`, `MM.YYYY`, `*.MM.YYYY`

To search for all dates within a given year, the year has to be given only:

- `YYYY`, `YYYY*`

Note

For all of the above date patterns, leading zeros in the year (`YYYY`) are not supported. This means that with a date pattern, it is possible to search for dates between 1000-01-01 and 9999-12-31 only.

Sample Code

Search for all Dates Within a Given Year

```
// search for all dates in 2015
SCOPE:awards 2015

// search for all dates in 2015 using a wildcard
SCOPE:awards 2015*
```

To search for all dates within a given month, year and month have to be given in any of the supported date formats.

Sample Code

Search for all Dates Within a Given Year and Month

```
// search for all dates in April 2015
SCOPE:awards 201504
SCOPE:awards 2015-04
SCOPE:awards 04.2015
// search for all dates in April 2015 using a wildcard
SCOPE:awards 201504*
SCOPE:awards 2015-04*
SCOPE:awards 2015-04-*
SCOPE:awards *04.2015
SCOPE:awards *.04.2015
SCOPE:awards 04/*/2015
```

4.2.4 Row Operator

Usually all terms and attribute conditions given in the query language expression are treated as individual query parts. In some cases, when using 1:n joins, more than one search term or attribute condition will be valid for the same row of the view. The `ROW:` operator is used to combine these conditions.

Within the row operator, the `AND` operator, the `NOT/`– operator, and the fuzzy operator (~) can be used.

Sample Code

Row Operator Examples

```
ROW:(invlastname:wilson instcountry:"united states")
ROW:(programreferencetext:(chemistry environmental))  is equal to
ROW(programreferencetext:chemistry programreferencetext:environmental)
// error, freestyle search not supported
ROW:(organic carbon)
// error, OR operator, attribute lists and attribute groups not supported
ROW:(invfirstname:richard OR instcountry:"united states")
ROW:((invfirstname invlastname):wilson)
ROW:(investigator:wilson)
```

Because of limitations in the search engine, only one condition or search term within the `ROW:` operator can be assigned a ranking weight greater than 0. All other conditions and terms have to be set to a ranking weight of 0.

Sample Code

Row Operator and Ranking Weight

```
// weight given in query language
ROW:(invlastname:wilson instcountry:bermuda^0.8)   is equal to
ROW(invlastname:wilson^0 instcountry:bermuda^0.8)
// weight given for row operator
ROW:(invlastname:wilson instcountry:bermuda)^0.8   is equal to
ROW(invlastname:wilson^0.8 instcountry:bermuda^0)
// assumption: search configuration contains default weight=0.7 for column
CityName
ROW:(invlastname:wilson instcountry:bermuda)       is equal to
ROW(invlastname:wilson^0.7 instcountry:bermuda^0)
```

```
// search configuration does not contain a default weight for column CityName
ROW:(invlastname:wilson instcountry:bermuda)       is equal to
ROW(invlastname:wilson^1 instcountry:bermuda^0)

// complex expression
ROW(invlastname:wilson~0.7^0.8 AND NOT abstract:carbon~0.9 AND
instcountry:EQ:bermuda)

// error
ROW:(invlastname:wilson^1 instcountry:bermuda^0.8)
```

4.2.5 Separation of User Input and Other Filter Conditions

In the query language expression, it is possible to separate the search terms given by the user from other filter conditions that are automatically generated by the search application, for example, selected facet values and authorization checks.

Note

The separation of search terms and filter conditions is currently used by `esh_search()` to impliment some performance optimizations. While it does not change the search results, it is recommended to use this feature to improve response times.

For example, a user may enter a search term like 'carbon 2016', may have the authorization to see documents for the 'United States' only and may have selected the facet value 'Colorado' in the UI.

The resulting search without the separation of search terms (`carbon 2016`) and filter conditions (`instcountry:"United States" inststate:EQ:Colorado`) then displays as follows:

Sample Code

Query Language Expression Without Separation of Search Terms and Filter Conditions

```
SCOPE:awards carbon 2016 instcountry:"United States" inststate:EQ:Colorado
```

The following example shows the same query with filter conditions separated from the search term:

Sample Code

Query Language Expression with Separation of Search Terms and Filter Conditions

```
SCOPE:awards carbon 2016 FILTER:(instcountry:"United States"
inststate:EQ:Colorado)
```

4.2.6 Search in a Single View

With `sys.esh_search()`, it is possible to search in a single view.

This is done by searching in the object '`/$all`' with an additional scope parameter that defines the view that is searched.

The example shows how to search for the term 'carbon' in a view called 'awards'.

Sample Code

Call of esh_search() in the SQL Console

```
-- Enterprise Search api version 5 and higher
call esh_search('[ { "URI": [ "/v5/$all?$count=true&estimate=true&
$top=10&facets=all&$filter=Search.search(query=''SCOPE:awards
carbon'')" ] } ]', ?);
-- Enterprise Search api up to version 4
call esh_search('[ "/v4/$all?$count=true&estimate=true&$top=10&facets=all&
$filter=Search.search(query=''SCOPE:awards carbon'')" ]', ?);
```

4.2.7 Federated Search Request over Multiple Views

With `sys.esh_search()`, it is possible to run a federated search over multiple views in a single call.

To search in all available search views, the object '`/$all`' is used without an additional scope definition.

Example of a federated search request with `esh_search()` in the SQL console:

Sample Code

Federated Search Call in the SQL Console

```
-- Enterprise Search api version 5 and higher
call esh_search('[ { "URI": [ "/v5/$all?$count=true&estimate=true&
$top=10&facets=all&$filter=Search.search(query=''carbon'')" ] } ]', ?);
-- Enterprise Search api up to version 4
call esh_search('[ "/v4/$all?$count=true&estimate=true&$top=10&facets=all&
$filter=Search.search(query=''carbon'')" ]', ?);
```

As no search scope is given, the search is done over all views that the user is allowed to search in.

- Views on which the user has the `SELECT` privilege.
- Views with annotation `@EnterpriseSearch.enabled:true`.

It is also possible to limit the search scope to a subset of the views that the user is allowed to search in. In this case, the scope parameter contains a list of view names, as shown in the following example, that searches in views 'awards' and 'documents'.

Sample Code

Federated Search Call with Scope Definition in the SQL Console

```
-- Enterprise Search api version 5 and higher
call esh_search('[ { "URI": [ "/v5/$all?$count=true&estimate=true&
$top=10&facets=all&$filter=Search.search(query=''SCOPE:(awards OR documents)
carbon'')" ] } ]', ?);
-- Enterprise Search api up to version 4
call esh_search('[ "/v4/$all?$count=true&estimate=true&$top=10&facets=all&
$filter=Search.search(query=''SCOPE:(awards OR documents) carbon'')" ]', ?);
```

4.2.8 Query Options Supported by Federated Search

Federated search supports the following OData query options.

System Query Options:

- `$filter`
- `$top`
- `$skip`
- `$count`
- `$apply`
- `$select`
- `$orderby`

Note

`$select` reduces the search scope to views that contain all of the columns that are given in the `$select` parameter. If no view contains all of the given columns, search fails with an error.

Note

`$orderby` can only be used if the search is limited to a single view - either because only one view with search configuration exists or because the scope operator of the current query limits the scope to a single view. As soon as search is done in two or more views, the use of `$orderby` will result in an error.

Custom Query Options:

- `estimate`
- `facets`
- `facetlimit`
- `wherefound`
- `whyfound`

Related Information

[Supported OData System Query Options [page 120]](#)
[OData Custom Query Options for Search [page 123]](#)

4.2.9 Limit Search to one Schema

You can limit the search call to a single schema.

If a user has `SELECT` privileges for search views of multiple applications that are installed in multiple schemas, the user may want to limit the search to a single schema to only search in views that belong to a specific application. Or there may be conflicts because views in different schemas use the same name, and as a result the search call fails with an error.

In these cases, it is possible to limit the search call to a single schema only. The object '/<SCHEMA_NAME>/$all' is used to specify the schema, as shown in the example below that searches in schema 'AWARDS' only.

Sample Code

Search in one Schema Only

```
call esh_search('[ { "URI": [ "/v5/AWARDS/$all?$count=true&estimate=true&
$top=10&facets=all&$filter=Search.search(query=''carbon'')" ] } ]', ?);
call esh_search('[ { "URI": [ "/v5/AWARDS/$all?$count=true&estimate=true&
$top=10&facets=all&$filter=Search.search(query=''SCOPE:(awards OR documents)
carbon'')" ] } ]', ?);
```

4.2.10 Response of a Federated Search

The search response of a federated search is returned in OData format. It contains the search results from all views in a single result list.

Example of a federated search response:

Source Code

```
{
  "value": [
    {
      "@com.sap.vocabularies.Search.v1.Ranking": 0.9915987,
      "@com.sap.vocabularies.Search.v1.WhyFound": {
        "abstract": [
          "... fellowship is belowground <b>carbon</b> improving soil health
in low <b>carbon</b> soils the host institution for ..."
        ]
      },
      "@odata.context": "$metadata#awards",
      "id": 1523821,
      "title": "NSF Postdoctoral Fellowship in Biology FY 2015",
      ...
```

```
    },
    {
      "@com.sap.vocabularies.Search.v1.Ranking": 0.9914942,
      "@com.sap.vocabularies.Search.v1.WhyFound": {
        "abstract": [
          "... regulate the uptake and storage of <b>carbon</b> within soils
of cattle pastures ..."
        ],
        "title": [
          "... root exudation and accumulation of soil <b>carbon</b> in
perennial grasslands"
        ]
      },
      "@odata.context": "$metadata#awards",
      "id": 1501686,
      "title": "DISSERTATION RESEARCH: Does grazing stimulate root exudation
and accumulation of soil carbon in perennial grasslands?",
      ...
  ],
  "@odata.count": 700,
  "@com.sap.vocabularies.Search.v1.ResponseTime": 0.89329996,
  "@com.sap.vocabularies.Search.v1.SearchTime": 0.68600001,
  "@com.sap.vocabularies.Search.v1.SearchStatistics": {
    "ConnectorStatistics": [
      {
        "Schema": "AWARDS",
        "Name": "search::awards",
        "OdataID": "awards",
        "StatusCode": 200,
        "@com.sap.vocabularies.Search.v1.SearchTime": 0.67499984
      },
      {
        ...
      }
    ],
    "StatusCode": 200
  }
}
```

4.2.10.1 Snippets and Highlighted Text

Snippets and highlighted text are returned using the annotations `<columnname>@com.sap.vocabularies.Search.v1.Snippets` and `<columnname>@com.sap.vocabularies.Search.v1.Highlighted`.

A snippet contains either short text fragments that include any of the search terms or, if no search term has been found in the column, a text fragment showing the first few words of the text.

The highlighted text always contains the complete text of the column, with highlighted search terms that have been found in the column.

Sample Code

Search Response Containing Snippets and Highlighted Text

```
{
  "value": [
    {
      "@com.sap.vocabularies.Search.v1.Ranking": 0.9246466,
```

```
      "id": 1535764,
      "abstract@com.sap.vocabularies.Search.v1.Snippets": "The dissolved
organic <b>carbon</b> (DOC) in the world ocean ... system through its
connection with atmospheric <b>carbon</b> dioxide. Keeping track of this ...",
      "title@com.sap.vocabularies.Search.v1.Highlighted": "Preparation and
Distribution of Dissolved Organic <b>Carbon</b> Consensus Reference Materials",
      ...
    },
    ...
  ]
}
```

4.2.10.2 Multi-Value Columns

Multi-value columns are columns that contain more than one value for each anchor object and that come from a 1:n join.

The following example shows a search response, where the 'programReferenceText' for object 1234567 contains four different values. These are returned as a JSON array.

Sample Code

Example of a search response containing a multi-value column

```
{
  "value": [
    {
      "@com.sap.vocabularies.Search.v1.Ranking": 0.990905,
      "id": 1234567,
      "title": "Postdoctoral Fellowship in Biology",
      "programReferenceText": [
        "ARCTIC RESEARCH",
        "INTERDISCIPLINARY PROPOSALS",
        "ENVIRONMENTAL CHEMISTRY",
        "WATER RESOURCES/COASTAL & MARINE ENVIRON"
      ],
      ...
    },
    ...
  ]
}
```

4.2.10.3 Subobjects

Subobjects are defined as a group of columns that define an object, for example the first name and last name of a person or the address of a company.

The object can occur more than once for each anchor object, so subobjects are similar to multi-values and are also joined as a 1:n join. Subobjects are returned as a JSON array that contains a complex type, as shown in the example below.

The example shows a search response containing subobjects for investigators that are defined as first name, last name, email address, and role of a person.

Sample Code

Example of a search result containing subobjects

```
{
  "value": [
    {
      "@com.sap.vocabularies.Search.v1.Ranking": 0.9957994,
      "id": 1523821,
      "title": "Postdoctoral Fellowship in Biology",
      "investigator": [
        {
          "invFirstName": "Julie",
          "invLastName": "Armstrong",
          "invEmail": "jarmstrong@...",
          "invRole": "Co-Principal Investigator"
        },
        {
          "invFirstName": "Richard",
          "invLastName": "Wilson",
          "invEmail": "rwilson@...",
          "invRole": "Co-Principal Investigator"
        },
        {
          "invFirstName": "Michel",
          "invLastName": "Leclerq",
          "invEmail": "michel.l@...",
          "invRole": "Principal Investigator"
        }
      ],
      ...
    },
    ...
  ]
}
```

If a subobject definition uses option '`defaultExpand`' set to '`ALL'`, the search response contains all subobjects of each anchor object.

If a subobject definition uses option '`defaultExpand`' set to '`WHY_FOUND`', the search response contains only the subobjects of the anchor object that contain at least one of the search terms.

4.2.10.4 Spatial Data

`sys.esh_search()` returns spatial data (coordinates) in GeoJSON format.

Columns of SQL type `ST_POINT` and `ST_GEOMETRY` are returned in GeoJSON format. GeoJSON is a geospatial data interchange format based on JavaScript Object Notation (JSON).

Coordinates are returned in the spatial reference system of the database column. The spatial reference identifier is returned in the metadata and is not included in the search response.

The example below shows a search response for a view containing two spatial columns:

1. a column called `LOCATION` with SQL type `ST_POINT`
2. a column called `ENVELOPE` with SQL type `ST_GEOMETRY` that contains objects of spatial type `ST_POLYGON`

The additional column ID is the key column of the view. All columns are returned in the search response.

Sample Code


```
{
  "value": [
    {
      "@com.sap.vocabularies.Search.v1.Ranking": 1,
      "ENVELOPE": {
        "type": "Polygon",
        "coordinates": [ [ [ 13.404953821, 52.520006821 ],
                           [ 13.40495418,  52.520006821 ],
                           [ 13.40495418,  52.52000718  ],
                           [ 13.404953821, 52.52000718  ],
                           [ 13.404953821, 52.520006821 ]
                       ] ]
      },
      "ID": 1,
      "LOCATION": {
        "type": "Point", "coordinates": [ 13.404954, 52.520007 ]
      }
    },
    {
      "@com.sap.vocabularies.Search.v1.Ranking": 0.965,
      "ENVELOPE": {
        "type": "Polygon",
        "coordinates": [ [ [ 9.99368182, 53.55108482 ],
                           [ 9.99368218, 53.55108482 ],
                           [ 9.99368218, 53.55108518 ],
                           [ 9.99368182, 53.55108518 ],
                           [ 9.99368182, 53.55108482 ]
                       ] ]
      },
      "ID": 2,
      "LOCATION": {
        "type": "Point", "coordinates": [ 9.993682, 53.551085 ]
      }
    },
    ...
  ]
}
```


4.2.10.5 Facets

If facets are requested in a search call, the search response contains more than just the objects that are found, as it also includes additional result sets for the columns that are marked as facet columns.

For each facet, a list of the most frequent values in the search response is returned. For each value, the number of anchor objects that contain this value is also returned.

The example below shows a search response that includes the output of a facet calculation, as it might be returned by a search call such as

`/$all?$top=10&facets=all&$filter=Search.search(query='carbon').`

Sample Code

Example of a search response containing facet counts

```
{
```

```
  "value": [
    {
      "@com.sap.vocabularies.Search.v1.Ranking": 0.9915987,
      "id": 1523821,
      ...
    },
    ...
  ],
  "@com.sap.vocabularies.Search.v1.Facets": [
    {
      "@com.sap.vocabularies.Common.v1.Label": "Count by instrument",
      "@odata.context": "$metadata#Collection(Edm.EntityType)",
      "@com.sap.vocabularies.Search.v1.Facet": {
        "URI": "",
        "Dimensions": [
          {
            "PropertyName": "instrument",
            "PropertyType": "Edm.String",
            "FilterProperty": "instrument"
          }
        ]
      },
      "Items": [
        {
          "@odata.id": null,
          "instrument": "Standard Grant",
          "_Count": 548
        },
        {
          "@odata.id": null,
          "instrument": "Continuing grant",
          "_Count": 126
        },
        {
          "@odata.id": null,
          "instrument": "Fellowship",
          "_Count": 21
        },
        ...
      ]
    },
    {
      "@com.sap.vocabularies.Common.v1.Label": "Count by programElementText",
      "@odata.context": "$metadata#Collection(Edm.EntityType)",
      "@com.sap.vocabularies.Search.v1.Facet": {
        "URI": "",
        "Dimensions": [
          {
            "PropertyName": "programElementText",
            "PropertyType": "Edm.String",
            "FilterProperty": "programElementText"
          }
        ]
      },
      "Items": [
        {
          "@odata.id": null,
          "programElementText": "CHEMICAL OCEANOGRAPHY",
          "_Count": 29
        },
        {
          "@odata.id": null,
          "programElementText": "BIOLOGICAL OCEANOGRAPHY",
          "_Count": 22
        },
        ...
      ]
    },
```

```
    ...
  ]
```

4.2.10.6 Facets for Numeric Columns

Facets for numeric columns are returned as intervals. The borders of the intervals are calculated based on the search results:

1. The first interval contains the minimum value of the numeric column in the current search result.
2. The last interval contains the maximum value of the numeric column in the current search results.
3. Interval borders and sizes are optimized to get 'nice' borders (like, for example, '100-200', '200-300', '300-400' instead of '153.57-230.27','230.27-306.97','306.97-383.67').

The number of intervals returned varies. The optimal number of intervals returned is defined by the value of the custom query option `facetlimit` or the value of the `@EnterpriseSearch.filteringFacet.numberOfValues` annotation. If possible, the number of intervals returned equals this optimal number of intervals. The maximum number of intervals created is the optimal number of intervals plus three.

> **Note**
>
> If there are only a few distinct values in the search response (up to the optimal number of intervals), the facet returned contains the distinct values instead of intervals.

Numeric SQL types that are returned as the intervals:

1. `TINYINT`, `SMALLINT`, `INTEGER`, `BIGINT`
2. `DECIMAL`, `SMALLDECIMAL`
3. `REAL`, `DOUBLE`, `FLOAT`

Intervals are not returned ordered by count. Instead, they are ordered by value.

> **Note**
>
> Empty intervals are not returned in the search response.
>
> All intervals with a count greater than 0 are always returned in the search response.

Interval borders are returned in the format of the SQL type of the numeric column. The facet's property type equals the name of the EDM type followed by 'Range', as, for example, 'DecimalRange' for 'Edm.Decimal'.

The following example shows a search response with intervals:

> **Sample Code**
>
> Interval Facet for a Numeric Column Called 'amount'

```
...
  "@com.sap.vocabularies.Search.v1.Facets": [
    {
      "@com.sap.vocabularies.Common.v1.Label": "Count by amount",
```

```
      "@odata.context": "$metadata#Collection(Edm.EntityType)",
      "@com.sap.vocabularies.Search.v1.Facet": {
        "URI": "",
        "Dimensions": [
          {
            "PropertyName": "amount",
            "PropertyType": "Edm.Int32",
            "FilterProperty": "amount"
          }
        ]
      },
      "Items": [
        {
          "@odata.id": null,
          "amount": {
            "From": null,
            "To": 500000
          },
          "_Count": 729
        },
        {
          "@odata.id": null,
          "amount": {
            "From": 500000,
            "To": 1000000
          },
          "_Count": 101
        },
        {
          "@odata.id": null,
          "amount": {
            "From": 1000000,
            "To": 1500000
          },
          "_Count": 8
        },
        {
          "@odata.id": null,
          "amount": {
            "From": 2000000,
            "To": 2500000
          },
          "_Count": 3
        },
        {
          "@odata.id": null,
          "amount": {
            "From": 3000000,
            "To": null
          },
          "_Count": 1
        }
      ]
    },
...
```

The example below shows a search response with few distinct values. Therefore, the facet returned contains distinct values instead of intervals.

Sample Code

Numeric Facet with Discrete Values

```
...
  "@com.sap.vocabularies.Search.v1.Facets": [
```

```
    {
      "@com.sap.vocabularies.Common.v1.Label": "Count by amount",
      "@odata.context": "$metadata#Collection(Edm.EntityType)",
      "@com.sap.vocabularies.Search.v1.Facet": {
        "URI": "",
        "Dimensions": [
          {
            "PropertyName": "amount",
            "PropertyType": "Edm.Int32",
            "FilterProperty": "amount"
          }
        ]
      },
      "Items": [
        {
          "@odata.id": null,
          "amount": 500000,
          "_Count": 5
        },
        {
          "@odata.id": null,
          "amount": 661501,
          "_Count": 2
        },
        {
          "@odata.id": null,
          "amount": 515901,
          "_Count": 1
        },
        {
          "@odata.id": null,
          "amount": 560971,
          "_Count": 1
        },
        {
          "@odata.id": null,
          "amount": 689502,
          "_Count": 1
        }
      ]
    },
...
```

4.2.10.7 Facets for Date and Timestamp Columns

Facets for date and timestamp columns are returned using a set of predefined date intervals. All rows of the search result that lie within an interval are added to the corresponding document count.

The predefined date intervals contain some overlapping intervals. As a consequence, some documents are counted in more than one interval.

SQL types that are returned as date intervals are:

1. `DATE`
2. `TIMESTAMP`
3. `SECONDDATE`
4. `(N)VARCHAR` with any of the `@Semantics` annotations for dates
5. `DECIMAL` with any of the `@Semantics` annotations for dates

The table below shows the predefined intervals and an example for a search done on 2016-09-14.

Table 35:

Interval Label	From	To	Example Start	Example End
Future	<tomorrow>	<no end date>	2016-09-15	
Last 3 years	<today> - 3 years	<today>	2013-09-14	2016-09-14
Last 5 years	<today> - 5 years	<today>	2011-09-14	2016-09-14
Last seven days	<today> - 7	<yesterday>	2016-09-07	2016-09-13
Last two weeks	<today> - 14	<yesterday>	2016-08-31	2016-09-13
Next seven days	<tomorrow>	<today> + 7	2016-09-15	2016-09-21
Next two weeks	<tomorrow>	<today> + 14	2016-09-15	2016-09-28
Next year	<next year>-01-01	<next year>-12-31	2017-01-01	2017-12-31
Older	0001-01-01	<today> - 5 years - 1	0001-01-01	2011-09-13
This month	<this month>-01	<last day of month>	2016-09-01	2016-09-30
This year	<this year>-01-01	<this year>-12-31	2016-01-01	2016-12-31
Today	<today>	<today>	2016-09-14	2016-09-14

Intervals are not returned ordered by count. Instead, they are ordered by date, so they are returned in the order given in the table above.

Note

Empty intervals are not returned in the search response.

All intervals with a count greater than 0 are always returned in the search response. The query option 'facetlimit' does not change the number of intervals returned for a date or timestamp facet.

Interval borders are returned in the format of the SQL type of the date column, as for example '2016-08-11' for a `DATE` column and '20160811' for a `VARCHAR` column. The facet's property type equals the name of the EDM type followed by 'Range', as, for example, 'DateTimeRange' for 'Edm.DateTime'.

Sample Code

Date Facet

```
...
  "@com.sap.vocabularies.Search.v1.Facets": [
    {
      "@com.sap.vocabularies.Common.v1.Label": "Count by effectiveDate",
      "@odata.context": "$metadata#Collection(Edm.EntityType)",
      "@com.sap.vocabularies.Search.v1.Facet": {
        "URI": "",
        "Dimensions": [
```

```
          {
            "PropertyName": "effectiveDate",
            "PropertyType": "DateTimeRange",
            "FilterProperty": "effectiveDate"
          }
        ]
      },
      "Items": [
        {
          "@odata.id": null,
          "effectiveDate": {
            "From": "2016-08-11",
            "To": null,
            "@com.sap.vocabularies.Common.v1.Label": "Future"
          },
          "_Count": 4
        },
        {
          "@odata.id": null,
          "effectiveDate": {
            "From": "2016-08-11",
            "To": "2016-08-24",
            "@com.sap.vocabularies.Common.v1.Label": "Next two weeks"
          },
          "_Count": 1
        },
        {
          "@odata.id": null,
          "effectiveDate": {
            "From": "2016-08-01",
            "To": "2016-08-31",
            "@com.sap.vocabularies.Common.v1.Label": "This month"
          },
          "_Count": 1
        },
        {
          "@odata.id": null,
          "effectiveDate": {
            "From": "2016-01-01",
            "To": "2016-12-31",
            "@com.sap.vocabularies.Common.v1.Label": "This year"
          },
          "_Count": 177
        },
        {
          "@odata.id": null,
          "effectiveDate": {
            "From": "2013-08-10",
            "To": "2016-08-10",
            "@com.sap.vocabularies.Common.v1.Label": "Last 3 years"
          },
          "_Count": 832
        },
        {
          "@odata.id": null,
          "effectiveDate": {
            "From": "2011-08-10",
            "To": "2016-08-10",
            "@com.sap.vocabularies.Common.v1.Label": "Last 5 years"
          },
          "_Count": 832
        }
      ]
    }
  ],
...
```

4.2.10.8 Spatial Facets for ST_POINT Columns

Facets for ST_POINT columns are calculated using the grid-based aggregation functions of HANA.

Each facet value consists of three elements:

1. The '**Envelope**' defines the border of a grid cell and is returned as a polygon in GeoJSON format.
2. The '**Centroid**' defines the centroid of the grid cell and is returned as a point in GeoJSON format.
3. The '**_Count**' defines the number of anchor objects that lie within a grid cell.

> **Note**
>
> The `facetlimit` parameter defines the width and height of the grid. If the parameter is not given, a 10x10 grid is created.
>
> Empty grid elements are not returned in the search response.
>
> All elements with a count greater than 0 are always returned in the search response. The query option `facetlimit` does not change the number of grid elements returned for a spatial facet.

The example shows a spatial facet for a column called 'LOCATION' with SQL type `ST_POINT`.

> **Sample Code**
>
> Spatial Facet

```
{
  "value": [
    ...
  ],
  "@com.sap.vocabularies.Search.v1.Facets": [
    {
      "@com.sap.vocabularies.Common.v1.Label": "Count by LOCATION",
      "@odata.context": "$metadata#Collection(Edm.EntityType)",
      "@com.sap.vocabularies.Search.v1.Facet": {
        "URI": "",
        "Dimensions": [
          {
            "PropertyName": "LOCATION",
            "PropertyType": "GeometryPolygonFacet",
            "FilterProperty": "LOCATION"
          }
        ]
      },
      "Items": [
        {
          "@odata.id": null,
          "LOCATION": {
            "Envelope": {
              "type": "Polygon",
              "coordinates": [ [ [ 8.466039,  49.218317 ],
                                 [ 8.9931613, 49.218317 ],
                                 [ 8.9931613, 49.759913 ],
                                 [ 8.466039,  49.759913 ],
                                 [ 8.466039,  49.218317 ]
                             ] ]
            },
            "Centroid": {
              "type": "Point",
              "coordinates": [ 8.72960015, 49.489115001 ]
            }
```

```
          },
          "_Count": 23
        },
        {
          "@odata.id": null,
          "LOCATION": {
            "Envelope": {
              "type": "Polygon",
              "coordinates": [ [ [ 13.210139701, 52.467893001 ],
                                 [ 13.737262,    52.467893001 ],
                                 [ 13.737262,    53.009489001 ],
                                 [ 13.210139701, 53.009489001 ],
                                 [ 13.210139701, 52.467893001 ]
                             ] ]
            },
            "Centroid": {
              "type": "Point",
              "coordinates": [ 13.47370085, 52.738691 ]
            }
          },
          "_Count": 11
        },
        ...
      ]
    }
  ],
  "@odata.count": 241,
  ...
}
```

4.2.10.9 Leveled Hierarchy Facets

A leveled hierarchy facet counts the number of distinct anchor objects on the first column of the hierarchy that returns more than one distinct value.

The following examples show a search on a view with a leveled hierarchy called '`instLocation`' with levels '`instCountry`', '`instState`', '`instcity`'.

Sample Code

Definition of a leveled hierarchy

```
@Hierarchy.leveled: [ {
  name:   'instLocation',
  levels: [ { element : 'instCountry' },
            { element : 'instState' },
            { element : 'instCity' } ]
  }
]
```

A search such as `/$all?$top=10&facets=all&$filter=Search.search(query='carbon') might return the object count for column 'instCountry'`.

Sample Code

Leveled facet for column 'instCountry'

```
 {
  "value": [
    {
      "@com.sap.vocabularies.Search.v1.Ranking": 0.9915987,
      "id": 1523821,
      ...
    },
    ...
  ],
  "@com.sap.vocabularies.Search.v1.Facets": [
    {
      "@com.sap.vocabularies.Common.v1.Label": "Count by instCountry",
      "@odata.context": "$metadata#Collection(Edm.EntityType)",
      "@com.sap.vocabularies.Search.v1.Facet": {
        "Dimensions": [
          {
            "HierarchyName": "instLocation",
            "PropertyName": "instCountry",
            "PropertyType": "Edm.String",
            "FilterProperty": "instCountry"
          }
        ]
      },
      "Items": [
        {
          "@odata.id": null,
          "instCountry": "United States",
          "_Count": "695"
        },
        {
          "@odata.id": null,
          "instCountry": "Bermuda",
          "_Count": 4
        },
        {
          "@odata.id": null,
          "instCountry": "Germany",
          "_Count": 1
        }
      ]
    },
    ...
  ]
}
```

A follow-up search with an additional filter for country such as `/$all?$top=10&facets=all&$filter=Search.search(query='carbon instcountry:"United States"')` might return the object count for column 'instState'.

Sample Code

Leveled facet for column 'instState'

```
{
  "value": [
    {
      "@com.sap.vocabularies.Search.v1.Ranking": 0.9957994,
      "id": 1523821,
      ...
```

```
    },
    ...
  ],
  "@com.sap.vocabularies.Search.v1.Facets": [
    {
      "@com.sap.vocabularies.Common.v1.Label": "Count by instState",
      "@odata.context": "$metadata#Collection(Edm.EntityType)",
      "@com.sap.vocabularies.Search.v1.Facet": {
        "URI": "",
        "Dimensions": [
          {
            "HierarchyName": "instLocation",
            "PropertyName": "instState",
            "PropertyType": "Edm.String",
            "FilterProperty": "instState"
          }
        ]
      },
      "Items": [
        {
          "@odata.id": null,
          "instState": "California",
          "_Count": "88"
        },
        {
          "@odata.id": null,
          "instState": "New York",
          "_Count": 56
        },
        {
          "@odata.id": null,
          "instState": "Massachusetts",
          "_Count": 47
        },
        {
          "@odata.id": null,
          "instState": "Texas",
          "_Count": 42
        },
        ...
      ]
    },
    ...
  ],
  ...
}
```

4.2.11 Privileges Needed to Call sys.esh_search()

To use search on a database object (view or table function) as a built-in procedure, the user needs several permissions.

You need the following permissions:

- `Execute` permission on the built-in procedure (sys.esh_search).
- `Select` privileges on `_SYS_RT.ESH_MODEL` and `_SYS_RT.ESH_MODEL_PROPERTY`.
- `Read` privileges for all database objects to be searched (`select` privilege for views or `execute` privilege for table functions).
- `Select` privileges on all term mapping tables that are used in the search configuration.

No other privileges are needed to enable search using the built-in procedure `sys.esh_search()`.

4.2.12 Supported OData System Query Options

The built-in procedure `esh_search()` supports the following system query options:

Table 36:

System Query Option	Description

`$orderby`	Sort the response by any column in ascending or descending order. Results can be sorted by the output of the SQL function `SCORE()` using `Search.score()` as a function.
	
	Example: `$orderby=Search.score() desc`
	
	`$orderby=CityName`
	
	`$orderby=CityName asc,Name1 desc`
	
	`$orderby=CityName,Search.score() desc`
	
	.
	
	Note
	`$orderby=Search.score()` can only be used if the request contains a call to `Search.search()` and if the search term is not empty.
	
	Note
	`$orderby` can only be used if the search is limited to a single view - either because only one view with the search configuration exists or because the scope operator of the current query limits the scope to a single view. As soon as the search is completed in two or more views, the use of `$orderby` will result in an error.

System Query Option	Description
`$select`	Defines the columns that are returned in the JSON response. If the search configuration contains columns with a `presentationMode`, the response columns defined with `$select` have to be a subset of these columns. **Note** `$select` restricts the search scope to views that contain all the columns specified in the `$select` parameter. If none of the views contain all the specified columns, the search fails with an error. Example: `$select=ID,Name1,Name2`
`$filter`	Filter condition supports only a subset of the Boolean expressions defined in the OData v4 standard: • logical operators (eq, ne, gt, ge, lt, le, and, or, not) • primitive literals as operands: property names (column names), numeric and string constants, null value • search specific functions: `Search.search()` Arithmetic operators, canonical functions, path expressions and so on are not allowed. Example: Preferred to have everything in a query language expression:`$filter=Search.search(query='carbon ROW:(inststate:EQ:colorado instcity:EQ:boulder)')` Not recommended, but also working: `$filter=Search.search(query='carbon') and instState eq 'Colorado' and instCity eq 'Boulder'`
`$top`	Defines the number of items returned in the response. Example: `$top=10`
`$skip`	Defines how many items are skipped in the response. Example: `$skip=20`

System Query Option	Description
`$count`	If `$count=true`, the total number of matching items is returned in the response. This is the number of items that would be returned if `$top` and `$skip` were not used. See also the custom query option `estimate`. When the `@EnterpriseSearch.key` annotation is used to define the keys of the anchor table, distinct anchor keys are counted instead of counting the rows in the database view. Example: `$count=true&estimate=true`
`$apply`	Example: `$apply=filter(...)/groupby(...)`

> **Note**
>
> The following OData v4 system query options are not supported:
>
> - `$expand`: Associations to other entity sets are not available, so there is nothing to be expanded.
> - `$format`: The JSON format is always returned.
> - `$search`: The `Search.search()` function is used instead.

4.2.13 OData Custom Query Options for Search

The following OData custom query options are supported:

Table 37:

OData Custom Query Option	Description
`facets`	The 'facets' option defines how many facets are calculated: • `facets = 0`: No facets are returned (default if the option is not given). • `facets = n (n > 0)`: Maximum n facets are returned. This is always a subset of the columns tagged with the `@EnterpriseSearch.filteringFacet.default:true` annotation. • `facets = all`: Facets for all columns tagged with the `@EnterpriseSearch.filteringFacet.default:true` annotation are returned. • `facets = column1,column2,...,columnn`: Facets for the specified columns are returned. All columns have to be tagged with the `@EnterpriseSearch.filteringFacet.default` annotation (set to true or false). For each facet, the top 10 values are returned if the `facetlimit` option is not selected. When using the `@EnterpriseSearch.key` annotation to define the keys of the anchor table, facets count distinct anchor keys instead of counting rows.
`facetlimit`	By default, the top 10 values for each facet are returned. Using the 'facetlimit' option, it is possible to define how many values are returned for each facet. `facetlimit=10` is the default value.
`estimate`	When using `estimate=true` with `$count=true`, an estimated result count is returned in some cases in order to reduce response times. The option has no effect if facets are requested. In this case, the exact result count is always returned. Using `estimate` without `$count=true` returns an error: • `estimate=true`: Search might return an estimated result count instead of an exact count. • `estimate=false`: This is the default setting.

<table>
<tr><th>OData Custom Query Option</th><th>Description</th></tr>
<tr><td>

`wherefound`

</td><td>

If `wherefound=true` is specified, the search response returns the names of the attributes where the search term has been found.

The where found information is returned as the annotation `@com.sap.vocabularies.Search.v1.WhereFound`.

Wherefound can only be used if a search term is entered using the search option:

- `wherefound=true`: The where found information is returned.
- `wherefound=false`: This is the default setting.

</td></tr>
<tr><td>

`whyfound`

</td><td>

If whyfound=true is specified, the search response returns the names and values of the attributes where the search term has been found. The location in the values where the search term has been found in the value is highlighted by `<b>...</b>` tags.

The why found information is returned as the annotation `@com.sap.vocabularies.Search.v1.WhyFound`:

Sample Code

```
"@com.sap.vocabularies.Search.v1.Why
Found":{
  "CITYNAME":[
    "<b>Walldorf</b>"
  ],
  "HOTELNAME":[
    "<b>Walldorf</b> Astoria"
  ]
}
```

Whyfound can only be used if a search term is entered using the search option:

- `whyfound=true`: The why found information is returned.
- `whyfound=false`: This is the default setting.

</td></tr>
</table>

Note

`wherefound` and `whyfound` return information for columns used in a search only. For freestyle search terms (example: 'searchTerm'), this includes all columns with `defaultSearchElement` set to `true`. For searches in a single column (example: 'aColumn:searchTerm'), this includes the given column.

Columns that are only used in attribute conditions (example: 'aColumn:eq:term' or 'aColumn:[1 5]') are not added to the `whyfound` and `wherefound` information.

4.2.14 OData Custom Functions for Search

Search.score()

`Search.score()` can be used with `$orderby` in order to arrange the search results by `SCORE()` in ascending or descending order.

The results can only be sorted by `Search.score()` if the output of the `SCORE()` function is available in the search response.

Search.search()

`Search.search()` is used to define the search term. It accepts the following parameter:

Table 38:

Parameter	Possible Values	Description
`query`	<search term>	Defines the search term. The syntax of the search query language is used.

Examples

```
$filter=Search.search(query='sap')
```

```
$apply=filter(Search.search(query='Walldorf'))
```

Note

Escaping of single quotes: The string parameters that are passed to `search.search()` cannot contain unquoted single quotes. Each single quote in a string has to be replaced by two single quotes, as specified by the OData standard.

Related Information

Search Query Language [page 126]

4.2.15 Search Query Language

You use the search query language to define a search query, which is handled by the built-in procedure `sys.esh_search()`. The search query language offers a range of operators,wildcards and escaping rules while specifying a search request.

Basic Search Terms

The most simple search query consists of a single term, for example `car`.

This search returns all items where the term `car` is found in any relevant column of the search model.

The relevant columns are predefined in the search configuration. See the documentation for `sys.esh_config()`.

> Note
>
> The search is case-insensitive. Searching with `car`, `Car`, `CAR`, or `cAr` returns the same result.

Searching with several terms returns all items where all of the search terms are found in one or more fields, regardless of the order of the terms or the field.

A search for `my car` returns all items where both the term **my** and the term **car** occur in any field. Tokenization during query parsing is done exclusively at the space `" "` character.

Wildcards

The `*` wildcard can be used at any place inside a search token and stands for zero or more characters.

A search for `car*` returns all items where any relevant column contains a term starting with **car**, e.g. **car**, **cars**, **care**, **carry**, **carbon**, **careful**.

A search for `*car` returns all items where any relevant column contains a term ending with **car**, e.g. **car**, **streetcar**, **autocar**, **flatcar**.

A search for `*car*` returns all items where any relevant column contains a term containing the subterm **car** like **car**, **cars**, **care**, **carry**, **carbon**, **careful**, **streetcar**, **autocar**, **flatcar**, **flatcars**.

The `?` wildcard can be used at any place in the search token and stands for any one character.

A search for `ca?`, for example, matches **car**, **cat**, **can, but not cars**, **caterpillar**, **case**.

You can also use wildcards within tokens, for example `c*ar` or `c?ar`.

Phrases

You can also use phrases in your search query. A phrase is a group of words written in double quotes. Phrase searches are used to find an exact phrase, for example `"my car"`.

The search for `"my car"` returns all items where the complete phrase is found as it is in any relevant column.

Phrase searches can be combined with wildcards. The search for `"my ca?"` matches the phrases **"my cat"**, **"my car"**, **"my can"**.

Operators

You can use the operators **AND** and **NOT** in your search queries.

The **AND** operator is the default operator.

The following search queries

`my car`

`my AND car`

`my and car`

return the same result.

> **Note**
>
> Unlike phrase search, it does not matter in which order or in which field or fields the terms match.

The **NOT** operator is used to find items where one term is included but another is not.

A search for `my NOT car` returns all items where the term **"my"** is found in any freestyle field, but the term **car** is found is not found in any of the freestyle fields.

> **Note**
>
> The query `my NOT car` could also be written as `my AND -car`.

A shortcut for the **NOT** operator is the **-**. The following variants of the search query return the same result:

`my NOT car`

`my AND NOT car`

`my and not car`

`my -car`

The **NOT** operator can also be used with one term. A search for `-car` returns all items that do not have the term *"car"*, same as `* AND NOT car`.

The **OR** operator is used to express that either search term should lead to a search result: `my OR car` returns all items where any of the search terms are found in any relevant column.

The **NOT** operator could also be used in conjunction with **OR**: `my OR NOT car` or `my OR -car` returns all items that have the term **my** in any field or **car** in no field. To help you understand mor easily, imagine a search execution as a set-based operation. Take the whole set of items and remove all items with the term **car** in any field. Now calculate the union with the set of items having the term **my** in any field. The same result would be returned with `-(car -my)` .

Boosting

A term can be boosted with the `^` operator.

The query `my car^2` gives the term **car** a double weight compared to the term **my**.

The default value is 1. The weight must be a value >=0.

Grouping

Grouping can be used to build more complex search expressions using brackets, for example `(my AND car) OR (her AND dog)`.

Searching in Fields

The search field can be specified using the field name followed by a colon `title:car`. This returns all items where the term **car** is found in the **title** field.

Field identifiers shall be case insensitive. **Title**, **TITLE**, **TiTlE**, **title** would address the same field **title**.

Field identifiers must be single words. There should be a configuration to declare the field identifier, which can differ from the technical name of the column. Also, it should be possible to define a list of fields as search scope.

Any search term from above could be used as the second part of the search. If there is more than one term and you want to search for all terms in only one field, then a grouping with brackets is necessary:

```
title:(my car)
```

```
title:"my car"
```

```
title:((my AND car) OR (her AND dog))
```

```
title:car~0.9
```

The search queries `title:my car` and `car title:my` would get the same results as the term **car** is searched for in all relevant columns.

You can extend the search scope using multiple fields. The default operator for scope is `OR`.

The following queries return all items where the term **car** is found in the **title**, **author** or **abstract** field:

```
(title OR author OR abstract):car
```

`(title author abstract):car`

You can also use the `AND` and `NOT` operators:

The query `(title AND author AND abstract):car` returns all items where the term **car** is found in the **title** and**author** and **abstract** field.

IN-Lists

You specify an IN-list as follows:

```
id:OR(K1 K2 K3 K4 K5 K6 K7)
```

This is a shorter and more convenient way from of the following statement:

```
id:K1 OR id:K2 OR id:K3 OR id:K4 OR id:K5 OR id:K6 OR id:K7
```

Fuzzy Search

Fuzzy search can be used to find similar terms.

The search query `car~` matches with **car**, **cars**, **can**, **cat**.

If no value for fuzziness is provided, the default value of `0.8` is used. Use can specify a higher fuzziness value if you want to receive more precise results, or you can specify a lower fuzziness value if you want to match more tokens.

The fuzziness value needs to be a floating point number between 0.0 and 1.0 (e.g. `car~0.9`).

> **Note**
>
> Fuzzy search does not work in combination with a wildcard search.

Escaping

The escaping of the special characters **^ # : ~ ( ) [ ] NOT_ AND_ OR_ NOT AND OR** is done with double quotes **""**.

Escaping of the special characters *** ? " ** is done with **** within a quoted token. `"\*\*\*"` searches for the token ***.

A search for `"(3:3^2~#)-1"` searches for the token **(3:3^2~#)-1.**. The special character "-" must be escaped if it is the first character of a token.

A search with a token having a **-** inside is supported.

A search for `check-in` searches for **check-in**, wheras `check -in` searches for **check** and excludes **in**.

A search for `check "-in"` searches for **check** and **-in**.

Precedence of Phrases and Operators

The following precedence is forged by the system handling a search:

1. Special characters: **?*\^**
2. Phrases **""** and special characters: **():**
3. Special characters: **[]**
4. Special character: **~**
5. The **NOT_** operator
6. The **AND_** operator
7. The **OR_** operator
8. The **NOT** operators: **- NOT**
9. The **AND** operator
10. The **OR** operator

Search on Single or Multiple Views

All examples above show searches on a single view. This is the case if only one searchable view exists in a system, or the UI restricts the search to one view. If more than one search view is available, a federated search is performed where the user needs the option to select a search scope. The search scope identifier is derived from the view name. Characters which are not allowed for OData identifiers are removed, such as **. : / \ $**. Package prefixes and context of SAP HANA CDS annotations are also removed.

Examples

Perform a search in all relevant columns of all connectors by just providing the token: `car`

Perform a search specifically in the document view: `scope:document car`

Perform a search in a list of views: `scope:(document OR customer) car`.

Perform a complex search query: `scope:document car title:my` This query searches in all **document** fields with the term **car** and additionally in the **title** field with the term **my**.

Example

Example of a call with `sys.esh_search()`:

```
call esh_search('[ "/$all?$filter=Search.search(query=''scope:(document OR
customer) car'')&$top=10" ]', ?);
```

4.2.16 Annotations in the Search Response

The following annotations are returned in the search response.

Table 39:

Annotation	Description	Example
`Facets`	Contains the result list of a facet column.	`"@com.sap.vocabularies.Search.v1.Facets": [ ... ]`
`Label`	The text to be displayed. The example returns: 1. A heading for the facet list 2. The display value of a single item in the facet list	`"@com.sap.vocabularies.Common.v1.Label": "Count by FACET1"` `"FACET1@com.sap.vocabularies.Common.v1.Label": "mysearchquery"`
`Ranking`	The ranking value, output of the `SCORE()` function, returned for each item in the result.	`"@com.sap.vocabularies.Search.v1.Ranking": 0.75`
`ResponseTime`	The runtime of the search, returned for each search call of a bulk request.	`"@com.sap.vocabularies.Search.v1.ResponseTime": 0.0143`
`URI`	Currently always empty. In future this will be `URI` that has to be used to get all values of a facet.	`"@com.sap.vocabularies.Search.v1.Facet.URI": ""`
`WhereFound`	The output of the `WHY_FOUND()` function, returned for each item in the result if a freestyle search has been called, and where found=true has been given.	`"@com.sap.vocabularies.Search.v1.WhereFound": "<TERM>mysearchquery</TERM><FOUND>TXT1</FOUND>"`

4.2.17 @com.sap.vocabularies.Search.v1.Ranking

The `SCORE()` is returned in the result list if a search is performed using `Search.search(query=<searchterm>)` with a non-empty search term.

4.2.18 Dynamic Search Configurations

Overview

> **Note**
>
> Some applications define their own persistence to store search configurations. With `sys.esh_config()` and static configurations, the configurations have to be copied from the application's persistence to the SAP HANA search models by calling `sys.esh_config()` after every configuration change. In some cases, this may lead to inconsistencies between the search model stored in the SAP HANA configuration and the model stored in the application's persistence layer. To avoid the synchronization effort and possible inconsistencies, such applications want to pass the configuration to SAP HANA dynamically with each call to `sys.esh_config()`.
>
> Other applications define varying configurations that depend on, for example, the user settings or other parameters. In this case, each call to `sys.esh_config()` needs a new configuration and it is not possible to write these configurations to SAP HANA with`sys.esh_config()`, as multiple searches with different configurations may run in parallel.
>
> For the above use cases, `sys.esh_search()` allows the definition of dynamic search configurations at runtime. If a search configuration is passed to `sys.esh_search()`, all search configurations already stored in the SAP HANA configuration tables are ignored. This means that only the given search configurations are visible to this search call.
>
> The format of the search configurations for `sys.esh_config()` is identical to the configurations for `sys.esh_config()`. For details, see the `sys.esh_config()` documentation. At runtime, the search configurations that are passed to `sys.esh_config()` have to be validated and the search views have to be analyzed. As this is time-consuming, dynamic search configurations lead to slightly higher response times of `sys.esh_config()` compared to calls with static search configurations that had been created before with `sys.esh_config()` and SAP HANA CDS.

> **Note**
>
> Only static search configurations created with SAP HANA CDS or `sys.esh_config()` are supported.

Basic Example

> **Note**
>
> The configurations given in the examples below have been shortened to keep the sample code clear.

Sample Code

Dynamic Search Configuration


```
[
  {
    "Configuration": [
      {
        "Fullname": "AWARDSDB/awards",
        "EntityType": {
          "@Search.searchable": true,
          "@EnterpriseSearch.enabled": true,
          "Properties": [
            {
              "@EnterpriseSearch.key": true,
              "@EnterpriseSearch.presentationMode": [
                "TITLE"
              ],
              "Name": "id"
            },
            {
              "@EnterpriseSearch.presentationMode": [
                "TITLE"
              ],
              "@Search.defaultSearchElement": true,
              "Name": "title"
            }
          ]
        }
      }
    ],
    "URI": [
      "/v5/$all?$count=true&$top=10&$filter=Search.search(query='SCOPE:awards
carbon')"
    ]
  }
]
```


Bulk Request

It is possible to run many searches in parallel by specifying multiple pairs of configurations and search URIs. The following example shows a bulk request with two configurations and two search URIs. Both searches are executed in parallel.

Sample Code

Bulk Request with two Configurations and two Search URIs


```
[
  {
    "Configuration": [
      {
        "Fullname": "AWARDSDB/awards",
        "EntityType": {
          "@Search.searchable": true,
          "@EnterpriseSearch.enabled": true,
          "Properties": [
            {
              "@EnterpriseSearch.key": true,
```

```
                "@EnterpriseSearch.presentationMode": [
                  "TITLE"
                ],
                "Name": "id"
              },
              {
                "@EnterpriseSearch.presentationMode": [
                  "TITLE"
                ],
                "@Search.defaultSearchElement": true,
                "Name": "title"
              }
            ]
          }
        }
      ],
      "URI": [
        "/v5/$all?$count=true&$top=10&$filter=Search.search(query='carbon')"
      ]
    },
    {
      "Configuration": [
        {
          "Fullname": "AWARDSDB/documents",
          "EntityType": {
            "@Search.searchable": true,
            "@EnterpriseSearch.enabled": true,
            "Properties": [
              ...
            ]
          }
        }
      ],
      "URI": [
        "/v5/$all?$count=true&$top=10&$filter=Search.search(query='carbon')"
      ]
    }
]
```

Federated Search

As with static configurations, it is possible to run a federated search over multiple dynamic configurations. In this case, multiple configurations are given together with a single search URI.

The example below shows a federated search over two dynamic search configurations.

Sample Code

Federated Search Using two Dynamic Search Configurations

```
[
  {
    "Configuration": [
      {
        "Fullname": "AWARDSDB/awards",
        "EntityType": {
          "@Search.searchable": true,
          "@EnterpriseSearch.enabled": true,
          "Properties": [
            {
              "@EnterpriseSearch.key": true,
```

```
                "@EnterpriseSearch.presentationMode": [
                  "TITLE"
                ],
                "Name": "id"
              },
              {
                "@EnterpriseSearch.presentationMode": [
                  "TITLE"
                ],
                "@Search.defaultSearchElement": true,
                "Name": "title"
              }
            ]
          }
        },
        {
          "Fullname": "AWARDSDB/documents",
          "EntityType": {
            "@Search.searchable": true,
            "@EnterpriseSearch.enabled": true,
            "Properties": [

            ]
          }
        }
      ],
      "URI": [
        "/v5/$all?$count=true&$top=10&$filter=Search.search(query='carbon')"
      ]
    }
]
```

Sharing a Configuration

In some cases, an application wants to run more than one search with a single configuration. To avoid that the configuration has to be repeated and evaluated for each URI, it is possible to define multiple URIs that share the same configuration.

The example shows how to define two URIs that share the same configuration.

Sample Code

Two Search URIs Sharing a Single Configuration

```
[
  {
    "Configuration": [
      {
        "Fullname": "AWARDSDB/awards",
        "EntityType": {
          "@Search.searchable": true,
          "@EnterpriseSearch.enabled": true,
          "Properties": [
            {
              "@EnterpriseSearch.key": true,
              "@EnterpriseSearch.presentationMode": [
                "TITLE"
              ],
              "Name": "id"
            },
```

```
            {
              "@EnterpriseSearch.presentationMode": [
                "TITLE"
              ],
              "@Search.defaultSearchElement": true,
              "Name": "title"
            }
          ]
        }
      }
    ],
    "URI": [
      "/v5/$all?$count=true&$top=10&$filter=Search.search(query='carbon')",
      "/v5/$all?$count=true&$top=10&$filter=Search.search(query='fiber')"
    ]
  }
]
```

Privileges Needed for Dynamic Search Configurations

When using dynamic search configurations, the database user needs the following permissions:

- **Execute permission** on the built-in procedure (`sys.esh_search()`)
- **Read** privileges for all database objects that will be searched (select privilege for views or execute privilege for table functions)
- **Select** privileges on all term mapping tables that are used in the search configuration

Select privileges on `_SYS_RT.ESH_MODEL` and `_SYS_RT.ESH_MODEL_PROPERTY` are not needed when using dynamic search configurations.

4.2.19 Definition of the User Language

By default, `sys.esh_search()` uses the language that is defined in the database session context to get language-dependent texts for response columns.

Alternatively, it is possible to define the language as a parameter to `sys.esh_search()`.

The language may be given as a 2-character iso language code or as a 1-character ABAP language code. If needed, `sys.esh_search()` internally converts a given language code to the other format (for example, an ABAP application gives a 1-character ABAP language code, while the search view is defined with 2-character iso language codes).

Sample Code

Language Code Example

```
[
  {
    "Configuration": [
      {
        "Fullname": "AWARDSDB/awards",
        "EntityType": {
```

```
          "@Search.searchable": true,
          "@EnterpriseSearch.enabled": true,
          "Properties": [
            ...
          ]
        }
      }
    ],
    "URI": [
      "/v5/$all?$count=true&$top=10&$filter=Search.search(query='carbon')"
    ],
    "Language": [
      "en"
    ]
  }
]
```

4.2.20 Writing the Search Response to a Table

Applications may want to call `sys.esh_search()` and to process the result set in a single SQL Script procedure (without first passing the search response to the application, parsing the JSON response, and then passing the search result back to the database).

It is possible to optionally specify the name of a result table in the call to `sys.esh_search()`. In this case, instead of copying the search response to the JSON response object, the keys and optionally the score of the search results are copied to the result table.

The lifecycle of the data in the result table has to be handled by the application. This means that the application has to make sure that the result table is empty before`sys.esh_search()` is called.Otherwise the new search results will be appended to the existing table contents. The application also has to make sure that the results are removed from the result table if they are no longer needed. To simplify this, a temporary table may be used as a result table.

The result table has to be defined as follows:

- For each anchor key column (column with `@EnterpriseSearch.key` annotation), the result table has to define a column with the same name and with an identical or compatible SQL data type.
- Optionally, a column called '`SCORE`' or '`_SCORE`' can be added. The SQL type of this column has to be compatible with the `SCORE()` function,. `DECIMAL` or `DECIMAL(4,3)`, for example, can be used.

The columns are identified by their name and it is not possible to define another mapping of columns.

If a result table is specified, the search result is only copied to the result table. In other words, the JSON response object returned by `sys.esh_search()` only contains status information and error messages, but no search results (the 'value' array is empty), no result count, and no facets.

Most of the search parameters (for example, '`facets`', '`facetlimit`' or '`$count`') are ignored. '`$top`', '`$skip`', and '`$orderby`' are used to define which columns are copied to the result table.

A result table can only be specified, if the search URI references a single view. In other words, the scope of the search has to be limited to one view only.

Both static search configurations and dynamic search configurations can be used in combination with a result table.

Sample Code

Definition of a Result Table

```
[
  {
    "URI": [
      "/v5/$all?$count=true&$top=10&$filter=Search.search(query='SCOPE:awards
carbon')"
    ],
    "ResultTable": {
      "SchemaName": "AWARDSDB",
      "TableName": "TEMP_RESULT_TABLE"
    }
  }
]
```

4.3 Full-Text Search with SQL

In column-oriented tables, you can perform searches using the SQL `SELECT` statement.

Prerequisites

Before building SQL search queries, the following prerequisites must be met:

- The tables you want to search are column-oriented.
- You have created the required views for the tables you want to search.
- You have created the required full-text indexes for the columns you want to search.

Context

SAP HANA supports the standard SQL search syntax and functions for search queries on nearly all visible data types. You build SQL queries using the `SELECT` statement, as in the following example:

```
SELECT * FROM Contacts WHERE ID=1
```

However, in SAP HANA, columns of large object types and text have additional requirements. To enable search on columns of these types, you must ensure the following requirements are met:

- Full-text indexes have been created for the search-relevant columns. For columns of type `TEXT` and `SHORTTEXT`, this is done automatically.
- Search queries on the columns use the `CONTAINS` predicate.

For large object types and text, you build SQL queries using the `SELECT` statement and `CONTAINS` predicate, as in the following example:

```
SELECT * FROM Documents WHERE CONTAINS (*,'Comment')
```

To build a search query, proceed as follows:

Procedure

1. Use the SQL `SELECT` statement and specify the table or view and column you want to search. If required, include the `CONTAINS` predicate.
2. If required, specify scalar functions for the search.
3. Specify the search terms and the search type (EXACT, LINGUISTIC, or FUZZY). The search type is optional.

   > **Note**
   >
   > If you do not specify a search type, the search query is performed as an exact search by default.

> **Example**
>
> For further examples of the syntax used with the `SELECT` statement, see *SAP HANA SQL and System Views Reference*.

Related Information

SAP HANA SQL and System Views Reference

4.3.1 Search Queries with CONTAINS

In SAP HANA, you can search one or multiple columns by creating a query that includes the `CONTAINS` predicate. In SAP HANA, a search query with `CONTAINS` has a look and feel similar to common Internet search engines.

The CONTAINS predicate is optional for search queries on columns of most data types; however, for large object types and text, this predicate is mandatory. You can build a search query with the `CONTAINS` predicate as follows:

```
SELECT * FROM <tablename>
WHERE CONTAINS ((<column1>, <column2>, <column3>), <search_string>)
```

When you specify the `CONTAINS` predicate, SAP HANA runs the following internal checks:

- SAP HANA checks if the query contains one or more terms. If the query contains multiple terms, the terms are tokenized and concatenated.
- SAP HANA checks whether the query is to be run on one or more columns. If you only specify one column, to optimize the search, additional processes are skipped and the query is run on the single column. If you specify a wildcard, and therefore possibly numerous columns, SAP HANA automatically determines which columns are relevant for the search query.

After the checks are performed, SAP HANA builds and runs an internal query on the relevant columns only.

> **Note**
>
> If a column has a full-text index assigned, SAP HANA will automatically search on the index rather than on the original column.

Determination of Search-Relevant Columns

You can specify the search-relevant columns either at the creation of the view or directly for the query. SAP HANA determines which relevant columns to search based on the following hierarchy:

1. You specify a list of columns within the `CONTAINS` predicate. Even if a list of columns has been defined for the view, it is overridden by the columns stated in the query.
2. If you enter an asterisk (*) instead of a column list but you specified a list of relevant columns when creating the view, this list is used for the query.
3. If you enter an asterisk (*) and no list was provided when the view was created, all visible columns of the view or table are considered as search-relevant.

For information about creating views, see *Creating Views* in the *SAP HANA Administration Guide*.

Search Operators and Syntax

With the `CONTAINS` predicate, SAP HANA supports the following search operators:

- `OR`
  Matches are returned that contain at least one of the terms joined by the `OR` operator.
- – (minus)
  With a minus sign, SAP HANA searches in columns for matches that do not contain the term immediately following the minus sign.
- `" "` (quotation marks)
  Terms within the quotation marks are not tokenized and are handled as a string. Therefore, all search matches must be exact.
- `*`
  The asterisk sign replaces 0 or more characters in a search term (e.g., `cat*` would match **cats** and **catalogues**.
- `?`
  The question mark replaces a single character in a search term (e.g., `cat?` would match **cats**.

> **Note**
>
> If you enter multiple search terms, the `AND` operator is automatically interpreted. Therefore, you do not need to specify it.

For more information about the unique syntax requirements of the `CONTAINS` predicate, see the *SAP HANA SQL and System Views Reference*.

Scalar Functions

For search queries using the `CONTAINS` predicate, you can use different scalar functions to either return additional information about the results of your search queries or enhance how the results are displayed. These functions include `SNIPPETS`, `HIGHLIGHTED`, and `SCORE`.

Limitations

The following limitations apply to search queries using the `CONTAINS` predicate:

- You cannot search on more than one table or view at a time. If more than one table is joined in the `SELECT` statement, then all columns mentioned in the `CONTAINS` predicate must come from only one of the tables.
- You cannot enter a minus (-) search operator directly after OR.
- Brackets are not supported as search operators.
- Searches using the `CONTAINS` predicate do not consider non-physical columns, such as calculated columns, as search-relevant because these columns are created during the search and, therefore, are not available when SAP HANA internally checks the `CONTAINS` search query.
- The `CONTAINS` predicate only works on column-oriented tables.
- If you specify multiple `CONTAINS` predicates in the `WHERE` clause of the `SELECT` statement, only one of the predicates is allowed to consist of more than one column in the list of <contains_columns>.

4.3.1.1 SNIPPETS Function

For search queries using the `CONTAINS` predicate, you can use the function `SNIPPETS` to return search results with an excerpt of the text with your search term highlighted in bold. This short text excerpt provides some context for you to see where and how the term is used in the document.

This function uses the following syntax:

```
SELECT *, SNIPPETS (<text_column>) FROM <tablename>
WHERE CONTAINS (<search_term>)
```

Limitations

The SNIPPETS function has the following limitations:

- Only the first search term specified with the CONTAINS predicate is highlighted in the returned text.
- The query result contains only the first hit of the first search term.
- The text excerpt that is displayed with the search term is limited to a string of 12 tokens.
- This function only works on columns of the TEXT data type or columns with a full-text index.

> **Note**
>
> When processing HTML documents, the results of the SNIPPETS function depend on the column type. For columns of type TEXT, [N]VARCHAR, or [N]CLOB, HTML markup from the original input will appear in the SNIPPETS results. For columns of type BINTEXT and BLOB, HTML markup from the original input will not appear in the SNIPPETS results.

4.3.1.2 HIGHLIGHTED Function

For search queries using the CONTAINS predicate, you can use the function HIGHLIGHTED to return the content of the found document with your search term highlighted in bold.

Search queries using the HIGHLIGHTED function return the data type NCLOB.

This function uses the following syntax:

```
SELECT *, HIGHLIGHTED (<text_column>) FROM <tablename>
WHERE CONTAINS (<search_term>)
```

Limitations

The HIGHLIGHTED function has the following limitations:

- Only the first search term specified with the CONTAINS predicate is highlighted in the returned text.
- The query result contains all hits of the first search term.
- This function only works on columns of the TEXT data type or columns with a full-text index.

> **Note**
>
> When processing HTML documents, the results of the HIGHLIGHTED function depend on the column type. For columns of type TEXT, [N]VARCHAR, or [N]CLOB, HTML markup from the original input will appear in the HIGHLIGHTED results. For columns of type BINTEXT and BLOB, HTML markup from the original input will not appear in the HIGHLIGHTED results.

4.3.1.3 SCORE Function

For search queries using the `CONTAINS` predicate, you can use the function `SCORE` to get the score, that means the relevance, of a record found.

SAP HANA calculates a score based on the following information:

- The relevance or weighting of attributes in a search using the `CONTAINS` predicate. The relevance of a hit depends on the weight of the column that caused the hit. You can specify weights when you create the view or in the `CONTAINS` predicate.
- Fuzziness in fuzzy search. The more exact a hit is, the higher the score is.
- Text ranking (TF-IDF).

This function uses the following syntax:

```
SELECT SCORE (),* FROM <tablename>
WHERE CONTAINS (<search_term>)
```

4.3.2 EXACT Search

An exact search returns records only if the search term or search phrase is contained in the table column exactly as specified. In the `SELECT` statement of the search query, you can specify the `EXACT` search type.

In an exact search, the search engine uses the word dictionary and the phrase index to detect the possible matches. The search engine then checks whether the words appear and use exactly the same spelling.

- For text columns, the search term must match at least one of the tokenized terms to return a column entry as a match.
- For string columns, the search term must match the entire string to return a column entry as a match.

Note

For more flexibility in a search query, you can use the supported wildcards `%` and `*`. Wildcards are supported for both text and string columns.

You can perform an exact search by using the `CONTAINS` predicate with the `EXACT` option in the `WHERE` clause of a `SELECT` statement. The exact search is the default search type. If you do not specify any search type in the search query, an exact search will be executed automatically.

Example

```
SELECT * FROM <tablename>
WHERE CONTAINS (<column_name>, <search_string>, EXACT)
```

```
SELECT * FROM <tablename>
WHERE CONTAINS (<column_name>, <search_string>)
--- Exact search will be executed implicitly.
```

```
SELECT * FROM <tablename>
WHERE CONTAINS (<column_name>, '"cats and dogs"')
```

```
--- Phrase search.
```

4.3.3 LINGUISTIC Search

A linguistic search finds all words that have the same word stem as the search term. It also finds all words for which the search term is the word stem. In the `SELECT` statement of the full-text search query, you can specify the `LINGUISTIC` search type.

When you execute a linguistic search, the system has to determine the stems of the searched terms. It will look up the stems in the stem dictionary. The hits in the stem dictionary point to all words in the word dictionary that have this stem

You can call the linguistic search by using the `CONTAINS` predicate with the `LINGUISTIC` option in the `WHERE` clause of a `SELECT` statement.

A linguistic search for *produced* will also find *producing* and *produce*.

Example

```
SELECT * FROM <tablename>
WHERE CONTAINS (<column_name, 'produced' , LINGUISTIC)
```

Limitations

You can only perform linguistic searches on columns that meet the following conditions:

- The columns contain text.
- For the columns, the `FAST_PREPROCESS` parameter is specified as `OFF`.

4.3.3.1 Term Mappings for Linguistic Search

A linguistic search finds all words that have the same word stem as the search term. By using term mappings the search term is expanded by synonyms, hypernyms, and hyponyms, which you specified in a term mapping table. The combination of linguistic search and term mappings could bring more records to the search result that could be useful to the users.

How do term mappings get into a linguistic search?

After entering a search term the preprocessor tokenizes the search term and creates stems. The stem dictionary of the fulltext index is checked on these stems and corresponding terms are returned. On the new and the initial search terms a term mapping is executed and synonyms are looked up for in the term mapping

table. For all found synonyms, hypernyms, or hyponyms the stem dictionary is checked again. Finally a new search request, where all found terms, synonyms and the initial search term are connected with OR, is executed and the result is returned to the user.

Example

The term mapping table contains the entry *shipping addresses* which is linked to *delivery address*.

A search for *ship addresses* results in the following technical actions:

1. The initial search term is split by the preprocessor into the stems *ship* and *address*.
2. The stem dictionary returns the corresponding phrases *shipping* and *address* and builds the search queries *shipping address*, *shipping addresses* and *ship address*
3. The term mapping table is checked and provides for *shipping adress* the term mapping *delivery address*.
4. For a last time the stemming dictionary is asked for the new input. For the new word *delivery* the entry *deliveries* is found and the addtional search input *deliveries address* is generated.

In the end we have 6 search inputs which are connected with OR in the final search request:

- *ship addresses*
- *shipping address*
- *shipping addresses*
- *ship address*
- *delivery address*
- *deliveries address*

The Required Term Mapping Table

For the linguistic search the same term mapping table as for fuzzy search is used. If you did not use fuzzy search with term mappings before, you have to create a new term mapping table.

The Select Statement

You can call the linguistic search by using the `CONTAINS` predicate with the `LINGUISTIC` option in the `WHERE` clause of a `SELECT` statement.

Example: `SELECT * FROM example WHERE CONTAINS(destination, 'ship addresses', LINGUISTIC('termMappingTable=termmappings,termMappingListId=01')) ORDER BY score DESC, id;`

> **Note**
>
> If the term mapping table is neither a column store table nor a column store join view but a calc view, SQL view or row store table, you have to specify the names of both fulltext indexes via the search options `termMappingTableFulltextIndexTerm1` and `termMappingTableFulltextIndexTerm2`.

For example `LINGUISTIC('termMappingTable=termmappings, termMappingListId=01, termMappingTableFulltextIndexTerm1= $TA_IDX_MYTM_TERM1, termMappingTableFulltextIndexTerm2= $TA_IDX_MYTM_TERM2')`

Related Information

Creating the Term Mapping Table [page 146]
Creating the Full-Text Index for the Stems [page 147]
Example SQL Statements [page 148]

4.3.3.1.1 Creating the Term Mapping Table

The term mappings for the linguistic as well for the fuzzy search are stored in a designated table in the SAP HANA database.

Context

The term mapping table uses the following format:

Table 40:

Column Name	Type	Primary Key	Description	Comment
MAPPING_ID	VARCHAR(32)	x	Primary key	For example, a GUID
LIST_ID	VARCHAR(32)		Term mapping list ID	Used to group term mappings
LANGUAGE_CODE	VARCHAR(2)		Language code (ISO2)	NULL: term mapping is valid for all languages
TERM_1	NVARCHAR(200)		Term 1, the term to be replaced	
TERM_2	NVARCHAR(200)		Term 2, the term that replaces term 1	
WEIGHT	DECIMAL, DECIMAL(n,m)		Weight, 0.0 <= Weight <= 1.0	

Procedure

1. If you don't use a term mapping table yet, create a new term mapping table.

Example code for creating a term mapping table:

```
CREATE COLUMN TABLE termmappings
(
    mapping_id    VARCHAR(32)   PRIMARY KEY,
    list_id       VARCHAR(32)   NOT NULL,
    language_code VARCHAR(2),
    term_1        NVARCHAR(200) NOT NULL,
    term_2        NVARCHAR(200) NOT NULL,
    weight        DECIMAL       NOT NULL
);
```

2. Fill the term mapping table with terms and their corresponding synonyms, hypernyms and hyponyms.

   Example: `INSERT INTO termmappings VALUES ('1','01','de','shipping address','delivery address','0.9'); INSERT INTO termmappings VALUES ('2','01','de','delivery address','shipping address','0.9');`

4.3.3.1.2 Creating the Full-Text Index for the Stems

In order to automatically calculate the stems for the term mapping definitions, two full-text indexes must be created.

Procedure

Create one full-text index on the `<TERM_1>` and one full-text index on the `<TERM_2>` column of the term mapping table. Use the option `TEXT ANALYSIS ON`. Specify the language column and the text analysis configuration of type `LINGANALYSIS_STEMS`.

```
CREATE FULLTEXT INDEX <idx_mytm_term1> ON <mytm>(term_1)
LANGUAGE COLUMN language_code
TEXT ANALYSIS ON
CONFIGURATION 'LINGANALYSIS_STEMS';
CREATE FULLTEXT INDEX <idx_mytm_term2> ON <mytm>(term_2)
LANGUAGE COLUMN <language_code>
TEXT ANALYSIS ON
CONFIGURATION 'LINGANALYSIS_STEMS';
```

Results

The stems are visible in the `$TA` tables `$TA_IDX_MYTM_TERM1` and `$TA_IDX_MYTM_TERM2` in the example above.

The first `$TA` table (`$TA_IDX_MYTM_TERM1`) defines the stems of the left side of the term mapping definitions. The second `$TA` table (`$TA_IDX_MYTM_TERM2`) defines the stems of the right side of the term mapping definitions.

Both tables are used to read the stems of the term mappings during runtime.

4.3.3.1.3 Example SQL Statements

The following SQL statements show an complete example how to prepare all tables and indexes in order to use term mappings in lingustic search.

Create the term mapping table *termmappings* if it is not yet available:

```
CREATE COLUMN TABLE termmappings
(
    mapping_id    VARCHAR(32)    PRIMARY KEY,
    list_id       VARCHAR(32)    NOT NULL,
    language_code VARCHAR(2),
    term_1        NVARCHAR(200)  NOT NULL,
    term_2        NVARCHAR(200)  NOT NULL,
    weight        DECIMAL        NOT NULL
);
```

Create the a new table named *example* to store your variants and synonyms for *shipping address* in the column *destination*:

```
CREATE COLUMN TABLE example
(
    id          INTEGER          PRIMARY KEY,
    destination SHORTTEXT(200),
    language    VARCHAR(2)
);
```

Fill the table *example* and enter values into the column *destination*:

```
INSERT INTO example VALUES ('1','ship addresses', 'en');
INSERT INTO example VALUES ('2','ship address', 'en');
INSERT INTO example VALUES ('3','shipping addresses', 'en');
INSERT INTO example VALUES ('4','shipping address', 'en');
INSERT INTO example VALUES ('5','delivery address', 'en');
INSERT INTO example VALUES ('6','deliveries address', 'en');
```

Fill the term mapping table with synonyms:

```
INSERT INTO termmappings VALUES ('1','01','de','ship addresses','delivery
address','0.9');
INSERT INTO termmappings VALUES ('2','01','de','delivery address','ship
addresses','0.9');
```

Create two full-text indexes for the stems:

```
CREATE FULLTEXT INDEX idx_mytm_term1 ON termmappings(term_1)
LANGUAGE COLUMN language_code
TEXT ANALYSIS ON
CONFIGURATION 'LINGANALYSIS_STEMS';
CREATE FULLTEXT INDEX idx_mytm_term2 ON termmappings(term_2)
LANGUAGE COLUMN language_code
TEXT ANALYSIS ON
CONFIGURATION 'LINGANALYSIS_STEMS';
```

Create an `SELECT` statement executing a linguistic search:

```
SELECT * FROM example WHERE CONTAINS(destination, 'ship addresses',
LINGUISTIC('termMappingTable=termmappings,termMappingListId=01')) ORDER BY score
DESC, id;
```

Table 41:

SCORE	ID	DESTINATION
1.0	1	ship addresses
0.9	2	ship address
0.9	3	shipping addresses
0.9	4	shipping address
0.9	5	delivery address
0.9	6	deliveries address

4.3.4 FUZZY Search

Fuzzy Search is a fast and fault-tolerant search feature for SAP HANA. A fuzzy search returns records even if the search term contains additional or missing characters or other types of spelling errors.

> **Note**
>
> We advise you to read through the Enabling Search section in the *SAP HANA Developer Guide* before working with this reference. This provides you with information about how to create a full-text index, which is a prerequesite for fuzzy search in SAP HANA.

About this Reference

In this complete reference, you can find all the documentation that you need to use the SAP HANA fuzzy search function in your development projects.

You can learn about the basic concepts of the `CONTAINS()` predicate and the `SCORE()` function. In data type-specific chapters, you can find information about the parameters specific for a certain data type, for the data types string or text for example. Further chapters deal with language specifics, for Chinese for example, and offer support information, such as monitoring, tracing and sizing.

The reference also contains lots of information about using search rules. These are used to separate search queries from the code for easier modification or replacement. A tutorial is provided so that you can practice directly on the search rule feature.

Fuzzy Search in a Nutshell

The term "fault-tolerant search" means that a database query returns records even if the search term (the user input) contains additional or missing characters, or other types of spelling error.

Fuzzy search can be used in various applications, for example:

- *Fault-tolerant search in text columns (html or pdf for example)*: Search for documents on 'Driethanolamyn' and find all documents that contain the term 'Triethanolamine'.
- *Fault-tolerant search in structured database content*: Search for a product called 'coffe krisp biscuit' and find 'Toffee Crisp Biscuits'.
- *Fault-tolerant check for duplicate records*: Before creating a new customer record in a CRM system, search for similar customer records and verify that no duplicates are already stored in the system. When creating a new record called 'SAB Aktiengesellschaft & Co KG Deutschl.' in 'Wahldorf' for example, the system would bring up 'SAP Deutschland AG & Co. KG' in 'Walldorf' as a possible duplicate.

You can call the fuzzy search by using the `CONTAINS` predicate with the `FUZZY` option in the `WHERE` clause of a `SELECT` statement.

Example

```
SELECT * FROM <tablename>
 WHERE CONTAINS (<column_name>, <search_string>, FUZZY (0.8))
```

4.3.4.1 Supported Data Types

Fuzzy search is only available for column tables and attribute views and supports the following SQL types and column store types.

Table 42:

SQL Type	Column Store Type	Supported Fuzzy Search Features
Character String Types		
VARCHAR	CS_STRING	string features
NVARCHAR	CS_STRING	string features
SHORTTEXT	CS_STRING	text features
Large Object (LOB) Types		
TEXT	CS_TEXT	text features
Datetime Types		
DATE	CS_DAYDATE	date features
Other		
FULLTEXT INDEX	-	text features

No other SQL types are supported by fuzzy search.

String Types

String types support a basic fuzzy string search. The values of a column are compared with the user input, using the fault-tolerant fuzzy string comparison.

When working with string types, the fuzzy string comparison always compares the full strings. If searching with 'SAP', for example, a record such as 'SAP Deutschland AG & Co. KG' gets a very low score, because only a very small part of the string (3 of 27 characters) matches.

Text Types

Text types support a more sophisticated kind of fuzzy search. Texts are tokenized (split into terms) and the fuzzy comparison is done term by term.

For example, when searching with 'SAP', a record such as 'SAP Deutschland AG & Co. KG' gets a high score, because the term 'SAP' exists in both texts. A record such as 'SAPPHIRE NOW Orlando' gets a lower score, because 'SAP' is only a part of the longer term 'SAPPHIRE' (3 of 8 characters match).

Date Types

Fuzzy search on date values checks for date-specific errors like dates that lie within a given range of days or dates that have month and day exchanged (for example, American versus British date format).

Related Information

Fuzzy Search on String Columns [page 183]
Fuzzy Search on Text Columns [page 198]
Fuzzy Search on DATE Columns [page 230]

4.3.4.2 Syntax

You can call the fuzzy search by using the `CONTAINS()` function with the `FUZZY()` option in the `WHERE` clause of a `SELECT` statement.

Basic example without additional search options

```
SELECT SCORE() AS score, *
FROM documents
WHERE CONTAINS(doc_content, 'Driethanolamyn', FUZZY(0.8))
ORDER BY score DESC;
```

Example with additional search options

Additional search options that change the default behavior of the fuzzy search can be specified as additional string parameters in the FUZZY() function.

The search options are specified as a comma-separated list of key-value pairs.

```
SELECT SCORE() AS score, *
FROM documents
WHERE CONTAINS(doc_content, 'Driethanolamyn', FUZZY(0.8, 'option1=value1,
option2=value2'))
ORDER BY score DESC;
```

Related Information

The CONTAINS() Predicate [page 152]
Available Fuzzy Search Options [page 168]

4.3.4.3 The CONTAINS() Predicate

To enable fuzzy search, the FUZZY() predicate is used inside the CONTAINS() predicate. The FUZZY() predicate takes a fuzzy threshold as an argument. The fuzzy threshold defines the level of error tolerance for the search.

Supported Database Objects

Fuzzy search and the CONTAINS() predicate are available for the SELECT statements on the following database objects:

- One column table
- One attribute view
- In some cases, also on SQL views (created with the CREATE VIEW statement), and on joins of multiple tables and views

> Note
>
> The following operations and database objects are not supported:
>
> - Row tables
> - Calculation views
> - Analytic views

Usage Examples

A statement like `...WHERE CONTAINS(col, 'search term', FUZZY(0.7))...` on a VARCHAR column returns all values with a fuzzy score greater than or equal to 0.7.

A search with `FUZZY(x)` returns all values that have a fuzzy score greater than or equal to x.

Search with FUZZY(0.0)

It is not possible to search for all values that have a fuzzy score greater than or equal to 0. This would return all values of a column and would result in large result sets. A search with **`FUZZY(0.0)`** therefore returns all values that have a fuzzy score greater than 0.

Use Cases of CONTAINS()

The CONTAINS() predicate can be used in the WHERE clause of a SELECT statement. The type of search it performs depends on its arguments:

1. A freestyle search on multiple columns
2. A full-text search on one column containing large documents
3. A search on one database column containing structured data

All searches can be performed either as an exact search or as a fuzzy search with additional tolerance for writing errors.

Freestyle Search on Multiple Columns

```
-- exact search
SELECT ... WHERE CONTAINS((col1, col2, col3), 'term1 term2 term3') ...;
-- or
SELECT ... WHERE CONTAINS((col1, col2, col3), 'term1 term2 term3', EXACT) ...;
-- fuzzy search
SELECT ... WHERE CONTAINS((col1, col2, col3), 'term1 term2 term3',
FUZZY(0.7)) ...;
```

Full-Text Search on One Column Containing Large Documents

To perform a full-text search, the column to be searched must be a text column.

```
-- exact search
SELECT ... WHERE CONTAINS(col1, 'term1 term2 term3') ...;
-- fuzzy search
SELECT ... WHERE CONTAINS(col1, 'term1 term2 term3', FUZZY(0.7)) ...;
```

Search on One Database Column Containing Structured Data

```
-- exact search
SELECT ... WHERE CONTAINS(col1, 'term1 term2 term3') ...;
-- fuzzy search
SELECT ... WHERE CONTAINS(col1, 'term1 term2 term3', FUZZY(0.7)) ...;
```

Multiple CONTAINS() Predicates in one SELECT

It is possible to use the CONTAINS() predicate more than once in a WHERE clause. In this case, only one CONTAINS() can be used for a freestyle search on multiple columns. All other calls to CONTAINS() can only access a single column.

```
SELECT ...
WHERE CONTAINS((col1, col2, col3), 'a b c', FUZZY(0.8))
  AND CONTAINS(col4, 'x y z', FUZZY(0.7))
  AND CONTAINS(col5, 'u v w', FUZZY(0.7))
  AND ...
```

Reserved Words and Special Characters in the Search String

When searching with CONTAINS(), some terms and characters have a special meaning, as described below. For more information, see the text search documentation.

Table 43:

Reserved Word/Special Character	Description
OR	A search such as CONTAINS(col, 'sap OR hana') searches for all records that contain 'sap' or 'hana' in the column 'col'. The OR keyword is case-sensitive, so CONTAINS(col, 'sap or hana') searches for records that contain 'sap' and 'or' and 'hana'. **Note** If your search input contains an 'or', make sure that it is not in uppercase characters in order to avoid the OR semantics.
Minus Sign (-)	A search such as CONTAINS(col, 'sap -hana') searches for all records that contain 'sap' but not 'hana'. If the second term is given as a phrase, as in CONTAINS(col, 'sap "-hana"'), the database searches for records that contain 'sap' and '-hana'. In text columns, the '-' is removed from the second search term, since it is a delimiter symbol. **Note** If your search input contains terms starting with a minus sign, make sure that these terms are enclosed in double quotes in order to avoid the NOT semantics.
Double Quotes (")	A search that is enclosed in double quotes is searched as a phrase. For example, CONTAINS(col, '"sap hana"') searches for all records that contain 'sap hana' as a phrase without any additional terms in between.

Reserved Word/Special Character	Description
Asterisk (*)	The asterisk activates a wildcard search. In this case, no fuzzy search is performed. An exact match with wildcards is called instead.
Percent Sign (%)	The percentage sign is replaced with an asterisk (*), and a wildcard search is called.

4.3.4.3.1 The SCORE() Function

When using CONTAINS() in the WHERE clause of a SELECT statement, the SCORE() function can be used to retrieve the score. This is a numeric value between 0.0 and 1.0.

The score defines the similarity between the user input and the records returned by the search. A score of 0.0 means that there is no similarity. The higher the score, the more similar a record is to the search input.

When more than one CONTAINS() is given in the WHERE clause, the score is calculated as a weighted average of the scores of all columns.

```
SELECT SCORE(), col1, col2, ... FROM tab
WHERE CONTAINS(col1, 'x y z')
  AND CONTAINS(col2, 'a b c')
  AND ...
ORDER BY SCORE() DESC;
```

It is possible to assign a weight to each column. When a weight is not given, the default weight is 1.0.

```
SELECT SCORE(), ... FROM tab
WHERE CONTAINS((col1, col2, col3), 'a b c', FUZZY(0.8), WEIGHT(1.0, 0.5, 0.5))
  AND CONTAINS(col4, 'x y z', FUZZY(0.7), WEIGHT(0.7))
  AND CONTAINS(col5, 'u v w', FUZZY(0.7))
  AND ...
ORDER BY SCORE() DESC;
```

WHERE conditions other than CONTAINS() are not part of the score calculation. The condition **`col5 = 'u v w'`** is not used to calculate the score for example.

```
SELECT SCORE(), ... FROM tab
WHERE CONTAINS((col1, col2, col3), 'a b c', FUZZY(0.8))
  AND CONTAINS(col4, 'x y z', FUZZY(0.7))
  AND col5 = 'u v w'
  AND ...
ORDER BY SCORE() DESC;
```

4.3.4.3.2 Examples

This section contains examples showing how to use the CONTAINS() predicate and the SCORE() function.

4.3.4.3.2.1 Fuzzy Search on One Column

Procedure

1. Create the data.

```
CREATE COLUMN TABLE companies
(
  id          INTEGER        PRIMARY KEY,
  companyname SHORTTEXT(200) FUZZY SEARCH INDEX ON
);
INSERT INTO companies VALUES (1, 'SAP Corp');
INSERT INTO companies VALUES (2, 'SAP in Walldorf Corp');
INSERT INTO companies VALUES (3, 'ASAP');
INSERT INTO companies VALUES (4, 'ASAP Corp');
INSERT INTO companies VALUES (5, 'BSAP orp');
INSERT INTO companies VALUES (6, 'IBM Corp');
```

2. Perform the search on one column.

```
SELECT SCORE() AS score, * FROM companies
WHERE CONTAINS(companyname,'xSAP Corp Walldorf',
FUZZY(0.7,'textSearch=compare,bestMatchingTokenWeight=0.7'))
ORDER BY score DESC;
```

Table 44:

SCORE	ID	COMPANYNAME
0.94	2	SAP in Walldorf Corp

4.3.4.3.2.2 Fuzzy Search on Two Columns

Procedure

1. Create the data.

```
CREATE COLUMN TABLE companies2
(
  id          INTEGER        PRIMARY KEY,
  companyname SHORTTEXT(200) FUZZY SEARCH INDEX ON,
  contact     SHORTTEXT(100) FUZZY SEARCH INDEX ON
);
INSERT INTO companies2 VALUES (1, 'SAP Corp',             'Mister Master');
INSERT INTO companies2 VALUES (2, 'SAP in Walldorf Corp', 'Master Mister');
INSERT INTO companies2 VALUES (3, 'ASAP',                 'Nister Naster');
INSERT INTO companies2 VALUES (4, 'ASAP Corp',            'Mixter Maxter');
INSERT INTO companies2 VALUES (5, 'BSAP orp',             'Imster Marter');
INSERT INTO companies2 VALUES (6, 'IBM Corp',             'M. Master');
```

2. Perform the search on two columns.

```
SELECT SCORE() AS score, * FROM companies2
WHERE CONTAINS(companyname, 'IBM',
FUZZY(0.7,'textSearch=compare,bestMatchingTokenWeight=0.7'))
```

```
AND   CONTAINS(contact, 'Master',
FUZZY(0.7,'textSearch=compare,bestMatchingTokenWeight=0.7'))
ORDER BY score DESC;
```

Table 45:

SCORE	ID	COMPANYNAME	CONTACT
0.91	6	IBM Corp	M. Master

3. Perform a freestyle search.

```
SELECT SCORE() AS score, * FROM companies2
WHERE CONTAINS((companyname,contact), 'IBM Master', FUZZY(0.7))
ORDER BY score DESC;
```

Table 46:

SCORE	ID	COMPANYNAME	CONTACT
0.8	6	IBM Corp	M. Master

> **Note**
>
> Freestyle searches always use TF/IDF to calculate the score and do not support parameters like 'textSearch=compare' or 'bestMatchingTokenWeight=0.7' which influence score calculation. This therefore results in a different score for the same record.

4.3.4.3.2.3 Perform a Freestyle Search on Two Columns

Procedure

1. Create the data.

```
CREATE COLUMN TABLE companies2
(
  id          INTEGER        PRIMARY KEY,
  companyname SHORTTEXT(200) FUZZY SEARCH INDEX ON,
  contact     SHORTTEXT(100) FUZZY SEARCH INDEX ON
);
INSERT INTO companies2 VALUES (1, 'SAP Corp',             'Mister Master');
INSERT INTO companies2 VALUES (2, 'SAP in Walldorf Corp', 'Master Mister');
INSERT INTO companies2 VALUES (3, 'ASAP',                 'Nister Naster');
INSERT INTO companies2 VALUES (4, 'ASAP Corp',            'Mixter Maxter');
INSERT INTO companies2 VALUES (5, 'BSAP orp',             'Imster Marter');
INSERT INTO companies2 VALUES (6, 'IBM Corp',             'M. Master');
```

2. Perform a freestyle search on two columns.

```
SELECT SCORE() AS score, * FROM companies2
WHERE CONTAINS((companyname,contact), 'IBM Master', FUZZY(0.7))
```

```
ORDER BY score DESC;
```

Table 47:

SCORE	ID	COMPANYNAME	CONTACT
0.83	6	IBM Corp	M. Master

> Note
>
> Freestyle search always uses TF/IDF to calculate the score and does not support parameters that influence score calculation, such as `textSearch=compare` or `bestMatchingTokenWeight=0.7`. This therefore results in a different score for the same record.

4.3.4.4 Fuzzy Score

This topic describes how fuzzy scores are calculated when comparing two strings or two terms.

The fuzzy search algorithm calculates a fuzzy score for each string comparison. The higher the score, the more similar the strings are. A score of 1.0 means the strings are identical. A score of 0.0 means the strings have nothing in common.

You can request the score in the SELECT statement by using the SCORE() function. You can sort the results of a query by score in descending order to get the best records first (the best record is the record that is most similar to the user input). If a fuzzy search of multiple columns is used in a SELECT statement, the score is returned as an average of the scores of all columns used.

When searching text columns, a TF/IDF (term frequency/inverse document frequency) score is returned by default instead of the fuzzy score. The fuzzy score influences the TF/IDF calculation, but it is important to keep in mind that, with TF/IDF, the range of the score values returned is normed to the interval between 0.0 and 1.0, and the best record always gets a score of 1.0, regardless of its fuzzy score.

The TF/IDF calculation can be disabled so that you get the fuzzy score instead. In particular, this makes sense for short-text columns containing data such as product names or company names. On the other hand, you should use TF/IDF for long-text columns containing data such as product descriptions, HTML data, or Microsoft Word and PDF documents.

4.3.4.4.1 Option similarCalculationMode

Option `similarCalculationMode` controls how the similarity of two strings (or, for TEXT attributes, terms) is calculated.

Score Calculation Modes

Basically, the similarity of two strings is defined by the number of common characters, wrong characters, additional characters in the search string, and additional characters in the reference string.

The following calculation modes exist:

Table 48: Modes

Mode	Impact on wrong characters	Impact on additional characters in search	Impact on additional characters in table
compare (default)	moderate	high	high
search	high	high	low
searchCompare	moderate/high	high	low
symmetricsearch	high	moderate	moderate
substringsearch	high	high	low
typeAhead	high	very high	low

Note that a high impact results in a lower score.

Table 49: Examples with score

Request	Reference	Compare	typeAhead	searchCompare	Search	Symmetric-search	substringSearch
search	searching	0.76	1.0	1.0	0.96	0.86	0.9
search	seerch	0.85	0.85	0.85	0.75	0.75	0.8
search	searchingforex-traterrestriallife	0.0	0.91	0.91	0.91	0.87	0.88
searchingforex-traterrestriallife	searching	0.0	0	0	0.35	0.84	0
searchingforex-traterrestriallife	seerch	0.0	0	0	0.24	0.79	0
searchingforex-traterrestriallife	searchingforthe-meaningoflife	0.6	0.6	0.6	0.57	0.6	0

SQL Examples

Preparations

```
DROP TABLE test_similar_calculation_mode;
CREATE COLUMN TABLE test_similar_calculation_mode
(
            id INTEGER          PRIMARY KEY,
            s  NVARCHAR(255)
);
INSERT INTO test_similar_calculation_mode VALUES ('1','stringg');
INSERT INTO test_similar_calculation_mode VALUES ('2','string theory');
INSERT INTO test_similar_calculation_mode VALUES ('3','this is a very very very
long string');
INSERT INTO test_similar_calculation_mode VALUES ('4','this is another very long
string');
```

similarCalculationMode compare

```
SELECT TO_INT(SCORE()*100)/100 AS score, id, s
FROM test_similar_calculation_mode
WHERE CONTAINS(s, 'strongtheory', FUZZY(0.5, 'similarCalculationMode=compare'))
ORDER BY score DESC;
```

Table 50:

Score	ID	S
0.84	2	string theory

similarCalculationMode searchCompare

The mode `searchCompare` combines the strength of modes `compare` and `search` while eliminating some of the shortcomings of search mode `search`. It is used when the user searches for similar words or for words starting with the user's input. For words with a similar length to the user's input, the tolerance for spelling errors is higher than for words that start with the user's input. The longer the user's input, the more spelling errors are allowed. In contrast to search mode `search`, the search term has to be found as a sequence within the database entry when using search mode `searchCompare`.

Sample Code

```
SELECT TO_INT(SCORE()*100)/100 AS score, id, s
FROM test_similar_calculation_mode
WHERE CONTAINS(s, 'stri', FUZZY(0.6, 'similarCalculationMode=searchCompare'))
ORDER BY score DESC
```

Table 51:

Score	ID	S
0.95	1	stringg
0.93	2	string theory
0.91	3	this is a very very very long string
0.91	4	this is another very long string

Sample Code

```
SELECT TO_INT(SCORE()*100)/100 AS score, id, s
FROM test_similar_calculation_mode
WHERE CONTAINS(s, 'strong', FUZZY(0.6, 'similarCalculationMode=searchCompare'))
ORDER BY score DESC;
```

Table 52:

Score	ID	S
0.8	4	this is another very long string
0.79	3	this is a very very very long string
0.74	2	string theory
0.7	1	stringg

The parameter 'spellCheckFactor' defines the influence of spelling errors on the score, as shown in the following example that uses 0.8 instead of the default value 0.9.

Sample Code

```
SELECT TO_INT(SCORE()*100)/100 AS score, id, s
FROM test_similar_calculation_mode
WHERE CONTAINS(s, 'strong', FUZZY(0.6,
'similarCalculationMode=searchCompare,spellCheckFactor=0.8'))
ORDER BY score DESC;
```

Table 53:

Score	ID	S
0.75	3	this is another very long string
0.75	4	this is a very very very long string
0.7	2	string theory
0.66	1	stringg

Note

`similarCalculationMode=searchCompare` cannot be used in combination with `interScriptMatching=true`.

similarCalculationMode typeAhead

The mode `typeAhead` is used when the user enters the beginning of a string and all strings starting with the user input will be returned. The tolerance for spelling errors at the beginning of the string is lower than with mode `compare`, but there may even be spelling errors in the first character of the string.

```
SELECT TO_INT(SCORE()*100)/100 AS score, id, s
FROM test_similar_calculation_mode
WHERE CONTAINS(s, 'stri', FUZZY(0.6, 'similarCalculationMode=typeAhead'))
ORDER BY score DESC
```

Table 54:

Score	ID	S
0.95	1	stringg
0.93	2	string theory

```
SELECT TO_INT(SCORE()*100)/100 AS score, id, s
FROM test_similar_calculation_mode
WHERE CONTAINS(s, 'strong', FUZZY(0.6, 'similarCalculationMode=typeAhead'))
ORDER BY score DESC;
```

Table 55:

Score	ID	S
0.74	2	string theory
0.7	1	stringg

The parameter `spellCheckFactor` defines the influence of spelling errors on the score, as shown in the following example that uses 0.8 instead of the default value 0.9.

```
SELECT TO_INT(SCORE()*100)/100 AS score, id, s
FROM test_similar_calculation_mode
WHERE CONTAINS(s, 'strong', FUZZY(0.6,
'similarCalculationMode=typeAhead,spellCheckFactor=0.8'))
ORDER BY score DESC;
```

Table 56:

Score	ID	S
0.7	2	string theory
0.66	1	stringg

> **Note**
>
> `similarCalculationMode=typeAhead` cannot be used in combination with `interScriptMatching=true`.

similarCalculationMode search

```
SELECT TO_INT(SCORE()*100)/100 AS score, id, s
FROM test_similar_calculation_mode
WHERE CONTAINS(s, 'strongtheory', FUZZY(0.5, 'similarCalculationMode=search'))
ORDER BY score DESC;
```

Table 57:

Score	ID	S
0.82	2	string theory
0.78	4	this is another very long string
0.70	3	this is a very very very long string

similarCalculationMode symmetricsearch

```
SELECT TO_INT(SCORE()*100)/100 AS score, id, s
FROM test_similar_calculation_mode
WHERE CONTAINS(s, 'strongtheory', FUZZY(0.5,
'similarCalculationMode=symmetricsearch'))
ORDER BY score DESC;
```

Table 58:

Score	ID	S
0.80	2	string theory
0.69	4	this is another very long string
0.62	3	this is a very very very long string
0.54	1	stringg

similarCalculationMode substringsearch

```
SELECT TO_INT(SCORE()*100)/100 AS score, id, s
```

```
FROM test_similar_calculation_mode
WHERE CONTAINS(s, 'strongtheory', FUZZY(0.5,
'similarCalculationMode=substringsearch'))
ORDER BY score DESC;
```

Table 59:

Score	ID	S
0.78	2	string theory

4.3.4.4.2 Option spellCheckFactor

Option `spellCheckFactor` defines the score for strings that are not identical but get a fuzzy score of 1.0.

There are two use cases for option `spellCheckFactor`:

- A) This option allows you to set the score for terms that are not fully equal but that would be a 100% match because of the internal character standardization used by the fuzzy search.
  For example, the terms 'Café' and 'cafe' give a score of 1.0 although the terms are not equal. For some users it might be necessary to distinguish between the terms.
  The decision whether two terms are equal is based on the term representation stored in the column dictionary. Option `spellCheckFactor` therefore works differently on string and text columns, as described in the following sections.
- B) The fuzzy search can return a 100% match for terms that are not identical but cannot be differentiated by the fuzzy-string-compare algorithm.
  For example, the fuzzy search cannot differentiate between the terms 'abaca' and 'acaba'. In this case, the `spellCheckFactor` can be used to avoid a score of 1.0.

If A) and B) are not needed by an application, you can set the `spellCheckFactor` to 1.0 to disable the feature.

4.3.4.4.2.1 Standardization of Letters and Terms

All characters are replaced by lowercase characters without any diacritics before the fuzzy comparison takes place. This is called standardization. It is therefore possible to get a 100% match when comparing two unequal terms, because the standardization process returned two identical terms.

Standardization Examples

Table 60:

Original Letter	Standardized Letter
E	e
e	e

Original Letter	Standardized Letter
É	e
é	e
Ė	e
ė	e

The letter i is treated differently, since it is not standardized to an ı as would be the 'standard' rule.

Table 61:

Original Letter	Standardized Letter
I	i
İ	i
i	i
ı	i

German umlauts are replaced by two characters.

Table 62:

Original Letter	Standardized Letter
Ä	ae
ä	ae
Ö	oe
ö	oe
Ü	ue
ü	ue
ß	ss

Due to this standardization we get high fuzzy scores for common differences in the spelling of words.

Table 63:

Original term	Standardized term
müller	mueller
Mueller	mueller
Cafe	cafe
Café	cafe

4.3.4.4.2.2 Search on a String Column (VARCHAR, NVARCHAR)

The decision as to whether two strings are the same is based on the string representation stored in the dictionary for the column in question. The contents of a string column are converted to lowercase characters before being stored in the dictionary. No other standardizations are carried out.

It is therefore not possible to use option spellCheckFactor distinguish between 'café' and 'cafe' for example.

Example

```
CREATE COLUMN TABLE test_spell_check_factor
(
    id INTEGER          PRIMARY KEY,
    s  NVARCHAR(255)
);
INSERT INTO test_spell_check_factor VALUES ('1','Muller');
INSERT INTO test_spell_check_factor VALUES ('2','Mueller');
INSERT INTO test_spell_check_factor VALUES ('3','Müller');
INSERT INTO test_spell_check_factor VALUES ('4','Möller');
SELECT SCORE() AS score, id, s
FROM test_spell_check_factor
WHERE CONTAINS(s, 'Müller', FUZZY(0.5, 'spellCheckFactor=0.9'))
ORDER BY score DESC;
DROP TABLE test_spell_check_factor;
```

Table 64:

SCORE	ID	T	Description
1.0	3	Müller	
0.9	2	Mueller	<< spellCheckFactor got used
0.88	1	Muller	
0.88	4	Möller	

4.3.4.4.2.3 Search on a Text Column (SHORTTEXT, TEXT or FULLTEXT INDEX)

Terms in text columns are standardized to lowercase characters without diacritics before being stored in the dictionary. In text columns it is therefore not possible to distinguish for example between 'café' and 'cafe' or 'Müller' and 'mueller'. In this case, the search always returns a score of 1.0.

The main use case of spellCheckFactor on text columns is therefore to avoid a score of 1.0 for terms like 'abaca' and 'acaba'.

Example

```
CREATE COLUMN TABLE test_spell_check_factor
(
    id INTEGER          PRIMARY KEY,
    t  SHORTTEXT(200)   FUZZY SEARCH INDEX ON
);
INSERT INTO test_spell_check_factor VALUES ('1','Muller');
INSERT INTO test_spell_check_factor VALUES ('2','Mueller');
INSERT INTO test_spell_check_factor VALUES ('3','Müller');
INSERT INTO test_spell_check_factor VALUES ('4','Möller');
SELECT SCORE() AS score, id, t
FROM test_spell_check_factor
WHERE CONTAINS(t, 'Müller', FUZZY(0.5,
'spellCheckFactor=0.9,textSearch=compare'))
ORDER BY score DESC;
DROP TABLE test_spell_check_factor;
```

Table 65:

SCORE	ID	T
1.0	2	Mueller
1.0	3	Müller
0.88	1	Muller
0.88	4	Möller

4.3.4.4.3 Option interScriptMatching

The option `interScriptMatching=on` is used to find Latin transliterations of non-Latin search terms, and vice versa.

Introduction to Inter-Script Matching

Fuzzy search supports all characters that are defined in the Unicode standard. When calculating a score, the characters of the search term and a database entry are compared, and each differing character is a spelling error that results in a reduced score value.

Often, users cannot enter data using the 'original' characters of a foreign language. As a result, a transliteration to the Latin alphabet is used to enter this data. A German user who wants to create a new Chinese business partner for example, types in the city name as 'Shanghai' instead of using the Chinese characters ('上海'). Later, a Chinese user who searches for the business partner in '上海' does not find the data because the search term '上海' and the city name 'Shanghai' stored in the database do not have any characters in common.

To support search requirements as in the example above, you can use the search option `interScriptMatching`. When `interScriptMatching=on` is used, it is possible to find Latin transliterations of non-Latin search terms, and vice versa. The behavior of the fuzzy search changes as follows:

1. Search with Latin characters
   - Latin alphabet database entries are searched as usual
   - Non-Latin alphabet database entries are searched using a Latin transliteration of the original data
2. Search with non-Latin characters
   - Latin alphabet database entries are searched using a Latin transliteration of the search term
   - Non-Latin alphabet database entries are searched using the original search term

Supported Character Sets

At present, only Chinese characters are supported for inter-script matching.

When comparing Chinese and Latin characters with `interScriptMatching=on`, a pinyin transcription is used to transcribe the sound of Chinese characters into Latin script.

Example

Preparations

```
CREATE COLUMN TABLE interscript
(
  str  NVARCHAR(100) PRIMARY KEY
);
INSERT INTO interscript VALUES ('Shanghai');
INSERT INTO interscript VALUES ('上海');
INSERT INTO interscript VALUES ('Beijing');
INSERT INTO interscript VALUES ('北京');
-- without inter-script matching
SELECT TO_DECIMAL(SCORE(),3,2), * FROM interscript WHERE CONTAINS(str,
'shanghai', FUZZY(0.7)) ORDER BY SCORE() DESC;
SELECT TO_DECIMAL(SCORE(),3,2), * FROM interscript WHERE CONTAINS(str, '上海',
FUZZY(0.7)) ORDER BY SCORE() DESC;
-- with inter-script matching
SELECT TO_DECIMAL(SCORE(),3,2), * FROM interscript WHERE CONTAINS(str,
'shanghai', FUZZY(0.7, 'interScriptMatching=on')) ORDER BY SCORE() DESC;
SELECT TO_DECIMAL(SCORE(),3,2), * FROM interscript WHERE CONTAINS(str, '上海',
FUZZY(0.7, 'interScriptMatching=on')) ORDER BY SCORE() DESC;
SELECT TO_DECIMAL(SCORE(),3,2), * FROM interscript WHERE CONTAINS(str,
'beijing', FUZZY(0.7, 'interScriptMatching=on')) ORDER BY SCORE() DESC;
SELECT TO_DECIMAL(SCORE(),3,2), * FROM interscript WHERE CONTAINS(str,
'Běijīng', FUZZY(0.7, 'interScriptMatching=on')) ORDER BY SCORE() DESC;
SELECT TO_DECIMAL(SCORE(),3,2), * FROM interscript WHERE CONTAINS(str, '北京',
FUZZY(0.7, 'interScriptMatching=on')) ORDER BY SCORE() DESC;
-- with spelling error
SELECT TO_DECIMAL(SCORE(),3,2), * FROM interscript WHERE CONTAINS(str, 'beijin',
FUZZY(0.7, 'interScriptMatching=on')) ORDER BY SCORE() DESC;
SELECT TO_DECIMAL(SCORE(),3,2), * FROM interscript WHERE CONTAINS(str, '被禁',
FUZZY(0.7, 'interScriptMatching=on')) ORDER BY SCORE() DESC;
DROP TABLE interscript;
```

For example, the second to last statement (...WHERE CONTAINS(str, 'beijin', FUZZY(0.7, 'interScriptMatching=on'))...) returns the following results.

Table 66:

Score	Str
0.93	Beijing
0.89	北京

4.3.4.5 Available Fuzzy Search Options

Note that some data types in the table below are data type combinations.

- **String**: SQL types VARCHAR and NVARCHAR
- **Text**: SQL types TEXT and SHORTTEXT and any columns that have an additional FULLTEXT INDEX
- **Date**: SQL type DATE

Table 67: Available Fuzzy Search Options

Name of Option	Short Name	Range	Default	Applies to Types	Description
`emptyScore`	es	0.0..1.0	not set	Text, String, Date	Defines how an empty value and a non-empty value match. For more information, see Option emptyScore and emptyMatchesNull [page 174]
`emptyMatchesNull`	emn	on,off,true, false	off	Text, String, Date	Returns null values if an empty value is searched. For more information, see Option emptyScore and emptyMatchesNull [page 174]
`returnAll`	ra	on,off,true, false	off	Text, String, Date	Returns all non-matching values of a column with a score of 0. For more information, see Option returnAll [page 180].
`similarCalculationMode`	scm	search, compare, symmetricsearch, substringsearch	compare	Text, String	Defines how the score is calculated for a comparison of strings (or terms in a text column). **Note** Note that '`scm=substringsearch`' is allowed for string columns only. For more information, see Option similarCalculationMode [page 158] .
`interScriptMatching`	ism	on,off,true, false	off	Text, String	Activates fuzzy matching across different scripts (for example, simplified chinese and pinyin). For more information, see Option interScriptMatching [page 166].

Name of Option	Short Name	Range	Default	Applies to Types	Description
`spellCheckFactor`	scf	0.0..1.0	0.9	Text, String	Sets the score for strings that get a fuzzy score of 1.0 but are not fully equal. For more information, see Option spellCheckFactor [page 163].
`abbreviationSimilarity`	abs	0.0..1.0	0.0	Text	Activates abbreviation similarity and sets the score. For more information, see Option abbreviationSimilarity [page 208].
`andSymmetric`	as	on,off,true, false	off	Text	Activates a symmetric AND content search. For more information, see Partially Matching with Parameter andThreshold [page 206].
`andThreshold`	at	0.0..1.0	1.0	Text	Activates a 'soft AND' and determines the percentage of the tokens that need to match. For more information, see Partially Matching with Parameter andThreshold [page 206].
`bestMatchingTokenWeight`	bmtw	0.0..1.0	0	Text	Influences the score, shifts total score value between best token score values and root mean square of score values. For more information, see Option bestMatchingTokenWeight [page 201].
`composeWords`	cw	1..5	1	Text	The maximum number of consecutive words from user input to be composed (default value 1 means composition is disabled by default). For more information, see Option composeWords [page 226].
`compoundWordWeight`	cww	0.0..1.0	0.9	Text	Term mapping weight for (de)compositions from (de)composeWords. For more information, see Option compoundWordWeight [page 228].
`considerNonMatchingTokens`	cnmt	max, min, all, input, table	max	Text	Influences the score, defines the number of terms used for score calculation. For more information, see Option considerNonMatchingTokens [page 202].

Name of Option	Short Name	Range	De-fault	Applies to Types	Description
`decomposeWords`	dw	1..5	1	Text	The maximum number of words into which a word from the user input is decomposed (default value 1 means composition is disabled by default). For more information, see Option decomposeWords [page 227].
`excessTokenWeight`	etw	0.0..1.0	1.0	Text	Defines the weight of excess tokens to improve sort order. For more information, see Option excessToken-Weight [page 204].
`minTextScore`	mts	0.0..1.0	0.0	Text	Minimum score of a TEXT field; if this score is not reached, the record is not part of the result. For more information, see Option minTextScore [page 210].
`phraseCheckFactor`	pcf	0.1..1.0	1.0	Text	The overall fuzzy score of a text column is multiplied with this value if the search terms do not appear in the correct order. For more information, see Option phraseCheckFactor [page 212]
`stopwordListId`	sli			Text	Activates the stopwords. For more information, see Usage [page 213].
`stopwordTable`	st		not set	Text	Activates the stopwords. For more information, see Usage [page 213]
`termMappingListId`	tmli			Text, String	Activates the term mappings. For more information, see Partially Matching with Pa-rameter andThreshold [page 206].
`termMappingTable`	tmt		not set	Text, String	Activates the term mappings. For more information, see Term Mappings [page 218].
`textSearch`	ts	fulltext, compare	full-text	Text, String, Date	Switches between full-text search with TF/IDF score and duplicate search with fuzzy score. For more information, see Option textSearch [page 211].

Name of Option	Short Name	Range	De-fault	Applies to Types	Description
`maxDateDistance`	mdd	0..100	0	Date	Specifies the allowed date distance when using fuzzy search on dates. For more information, see Fuzzy Search on DATE Columns [page 230].

4.3.4.5.1 Permitted Combinations of Fuzzy Search Parameters

Possible combinations of search options are shown in the table below.

Depending on the following conditions, some of the searchOptions parameters might not be permitted, because they do not make sense:

- The data type of a column
- The type of the search (freestyle or attribute search)
- The selected method for score calculation on text columns (fuzzy score or TF/IDF, search option textSearch)

If the user sets an option that is not allowed, an SQL error is thrown, and the SELECT aborts.

The search types used in the table below are:

- Freestyle search: the CONTAINS() predicate uses more than one column.
- Attribute search: the CONTAINS() predicate uses only a single column
- Search on TEXT...: attribute search on a column of type TEXT, SHORTTEXT, or another type with an additional FULLTEXT INDEX.
- Search on types other than TEXT: attribute search that is not performed on a TEXT column as defined above.

Table 68:

	Search on TEXT with Fuzzy Score	Search on TEXT with TD/IDF Score	Freestyle Search	Search on Types Other Than TEXT
	Attribute search and textSearch= compare and datatype= TEXT	Attribute search and textSearch=fulltext and datatype=TEXT	Freestyle search (full-text search with TD/IDF score)	Attribute search and datatype <>TEXT
`textSearch=compare`	YES	NO	NO	NO
`textSearch=fulltext`	NO	YES	NO	NO

	Search on TEXT with Fuzzy Score	Search on TEXT with TD/IDF Score	Freestyle Search	Search on Types Other Than TEXT
`similarCalculationMode`	YES (scm=substring-search is only valid for string types. Other types including text types are not supported.)	YES (scm=substring-search is only valid for string types. Other types including text types are not supported.)	YES (scm=substring-search for freestyle search: text columns use scm=compare instead.)	YES (scm=substring-search is only valid for string types. Other types including text types are not supported.)
`interScriptMatching`	YES	YES	YES	YES (Only valid for string types. Other types are not supported.)
`spellCheckFactor`	YES	YES	YES	YES
`composeWords`	YES	YES	YES (Search option has an effect on text columns only. For other column types the option ia ignored.)	NO
`decomposeWords`	YES	YES	YES (Search option has an effect on text columns only. For other column types the option is ignored.)	NO
`compoundWordWeight`	YES	YES	YES (Search option has an effect on text columns only. For other column types the option is ignored.)	NO
`termMappingTable/ListId`	YES	YES	YES (Search option has an effect on text columns only. For other column types the option is ignored.)	YES (Only valid für text and string types.)
`returnAll`	YES	YES	NO	YES
`emptyScore`	YES	NO	NO	YES (Only valid for text, string and date types. Numeric types are not supported.)
`abbreviationSimilarity`	YES	NO	NO	NO
`stopwordTable/ListId`	YES	NO	NO	NO
`andThreshold`	YES	NO	NO	NO
`andSymmetric`	YES	NO	NO	NO
`bestMatchingTokenWeight`	YES	NO	NO	NO

	Search on TEXT with Fuzzy Score	Search on TEXT with TD/IDF Score	Freestyle Search	Search on Types Other Than TEXT
`considerNonMatchingTokens`	YES	NO	NO	NO
`excessTokenWeight`	YES	NO	NO	NO
`minTextScore`	YES	NO	NO	NO
`phraseCheckFactor`	YES	NO	NO	NO
`maxDateDistance`	NO	NO	YES	YES (Only valid for date types. Other types are not supported.)
rank Calculation	fuzzy score	TF/IDF	TF/IDF (text columns) or fuzzy score (other SQL types)	fuzzy score

Legend

YES - The parameter is allowed.

NO - The parameter is not allowed, and an error message is returned if the user sets this option.

4.3.4.5.2 Search Options and CONTAINS()

This section gives several examples of using search options and the CONTAINS() statement.

Example without specifying additional search options

```
SELECT SCORE() AS score, *
FROM documents
WHERE CONTAINS(doc_content, 'Driethanolamyn', FUZZY(0.8))
ORDER BY score DESC;
```

Example with additional search options

You can specify additional search options that change the default behavior of the fuzzy search as an additional string parameter for the FUZZY() function.

```
SELECT SCORE() AS score, *
FROM documents
WHERE CONTAINS(doc_content, 'Driethanolamyn', FUZZY(0.8, 'option1=value1,
option2=value2'))
ORDER BY score DESC;
```

Specify the search options as a comma-separated list of key/value pairs.

Example with EXACT search and additional search options

You can also use search options in combination with an exact search. So, for example, you can use search options for term mappings and stopwords for fuzzy search and for exact search.

```
SELECT SCORE() AS score, *
FROM documents
WHERE CONTAINS(doc_content, 'Driethanolamyn', EXACT('option1=value1,
option2=value2'))
ORDER BY score DESC;
```

4.3.4.6 Fuzzy Search - Options for All Column Types

There are search options that are not only valid for a specific column type such as text or string. These options are described in this chapter.

4.3.4.6.1 Option emptyScore and emptyMatchesNull

These options define the score for empty strings and NULL values when comparing them to non-empty values.

Introduction

Many database tables often contain incomplete data. For example, the first name or the phone number of a customer might be empty, either because the information was not known when the database record was created or because of missing data maintenance.

The search input might therefore contain more information than the database record that the user is looking for. In this case, the user still expects to get the result.

To get the expected behaviour using standard SQL, an application developer writes code as in the following example:

```
SELECT score(), ... FROM ...
WHERE ...
  AND (CONTAINS(firstname, 'Peter', FUZZY(0.8)) OR firstname IS NULL)
  AND ...
```

It is not possible to specify the score for the 'firstname IS NULL' clause, so the overall score() for records with an empty firstname may get an unexpected score() that probably does not match the sort order of other results. SQL statements also become longer and more complex with the additional WHERE clauses.

This is why the 'emptyScore' option has been introduced.

To ensure symmetry, the search works the other way around too. When searching with an empty column, records that contain a value in the column are therefore also returned. This is important for batch processes

for example, where the order of records processed is not known, and results should be the same, regardless of the order of processing.

In the following sections, an empty column value is a column value that is either an empty string ('') or a NULL value. We do not distinguish between these two values.

Supported Data Types

The search option 'emptyScore' supports the following SQL data types:

- VARCHAR
- NVARCHAR
- SHORTTEXT
- TEXT
- DATE
- Columns with a FULLTEXT INDEX

i Note

Numeric types like INTEGER, DECIMAL, FLOAT, and so on are currently not supported.

Examples

```
DROP TABLE test_emptyscore;
CREATE COLUMN TABLE test_emptyscore
(
  id INTEGER  PRIMARY KEY,
  t  TEXT     FUZZY SEARCH INDEX ON
);
INSERT INTO test_emptyscore VALUES ('1', 'eins');
INSERT INTO test_emptyscore VALUES ('2', '');          -- empty string
INSERT INTO test_emptyscore VALUES ('3', '       ');   -- n blanks
INSERT INTO test_emptyscore VALUES ('4', NULL);        -- NULL value
```

Select 'eins' without emptyScore

```
SELECT SCORE() AS score, * FROM test_emptyscore WHERE CONTAINS(T, 'eins',
FUZZY(0.5, 'textSearch=compare')) ORDER BY score DESC, id;
```

Table 69:

SCORE	ID	T
1.0	1	eins

Select 'eins' with emptyScore

```
SELECT SCORE() AS score, * FROM test_emptyscore WHERE CONTAINS(T, 'eins',
FUZZY(0.5, 'textSearch=compare, emptyScore=0.5')) ORDER BY score DESC, id;
```

Table 70:

SCORE	ID	T
1.0	1	eins
0.5	2	
0.5	4	<NULL>

Select empty string without emptyScore

```
SELECT SCORE() AS score, * FROM test_emptyscore WHERE CONTAINS(T, '', FUZZY(0.5,
'textSearch=compare')) ORDER BY score DESC, id;
```

Table 71:

SCORE	ID	T
1.0	2	

Select empty string with emptyScore

```
SELECT SCORE() AS score, * FROM test_emptyscore WHERE CONTAINS(T, '', FUZZY(0.5,
'textSearch=compare, emptyScore=0.5')) ORDER BY score DESC, id;
```

Table 72:

SCORE	ID	T
1.0	2	
1.0	4	<NULL>
0.5	1	eins
0.5	3	

Select empty string with emptyScore and minTextScore

```
SELECT SCORE() AS score, * FROM test_emptyscore WHERE CONTAINS(T, '', FUZZY(0.5,
'textSearch=compare, emptyScore=0.5, mintextscore=0.8')) ORDER BY score DESC, id;
```

Table 73:

SCORE	ID	T
1.0	2	
1.0	4	<NULL>
0.5	1	eins
0.5	3	

> **Note**
>
> When searching with an empty value, for example, records that contain a value in the column are returned and will get the score of the parameter emptyScore. The list will not be shortened, even though the parameter minTextScore is set.

Select 'eins' with emptyMatchesNull

```
SELECT SCORE() AS score, * FROM test_emptyscore WHERE CONTAINS(T, 'eins',
FUZZY(0.5, 'textSearch=compare, emptyMatchesNull=true')) ORDER BY score DESC, id;
```

Table 74:

SCORE	ID	T
1.0	1	eins

Select empty string with emptyMatchesNull

```
SELECT SCORE() AS score, * FROM test_emptyscore WHERE CONTAINS(T, '', FUZZY(0.5,
'textSearch=compare, emptyMatchesNull=true')) ORDER BY score DESC, id;
```

Table 75:

SCORE	ID	T
1.0	2	
1.0	4	<NULL>

Related Information

Finding NULL Values When Searching for an Empty String With emptyMatchesNull [page 179]

4.3.4.6.1.1 Getting Records with Empty Column Values

When the search is called with the emptyScore option as in the following example

```
SELECT score(), ... FROM ...
WHERE ...
  AND CONTAINS(firstname, 'Peter', FUZZY(0.8, 'textSearch=compare,
emptyScore=0.9, ...<otherOptions>...))
  AND ...
```

the records returned by the search are the same as with the following SQL statement (scores differ of course because of the emptyScore parameter)

```
SELECT score(), ... FROM ...
WHERE ...
  AND (    CONTAINS(firstname, 'Peter', FUZZY(0.8,
'textSearch=compare, ...<otherOptions>...))
        OR firstname IS NULL
        OR firstname = ''
      )
  AND ...
```

For records containing a NULL or an empty string in the firstname column, the value of emptyScore (0.9 in this example) is used as firstname score when calculating the overall score. For records containing a non-empty firstname, the fuzzy score is calculated and used.

For columns that do not allow empty strings as values (date types for example) the result of the search is the same as the query

```
SELECT score(), ... FROM ...
WHERE ...
  AND (    CONTAINS(dateofbirth, '2000-01-02', FUZZY(0.8, '...<otherOptions>...))
        OR dateofbirth IS NULL
      )
  AND ...
```

> **Note**

Numeric types like INTEGER, DECIMAL and FLOAT are currently not supported.

4.3.4.6.1.2 Finding Non-Empty Column Values when Searching with an Empty Column Value

When the search is called with an empty string and with option emptyScore as in the following example:

```
SELECT score(), ... FROM ...
WHERE ...
  AND CONTAINS(firstname, '', FUZZY(0.8, 'textSearch=compare,
emptyScore=0.9, ...<otherOptions>...))
  AND ...
```

the result of the search is the same as with the following SQL statement (scores differ, of course, because of the emptyScore parameter):

```
SELECT score(), ... FROM ...
WHERE ...
  AND (    firstname IS NULL
        OR firstname = ''
        OR firstname LIKE '_%'
      )
  AND ...
```

For records that contain a NULL or an empty string in the firstname column, a score of 1.0 is used, because an empty string is considered a 'perfect' match for a search with an empty string. Records with a non-empty firstname get a firstname score of 0.9, which is the value of the emptyScore parameter in this example.

For columns that do not allow empty strings as a value (for example, date types), the result of the search is the same as the query:

```
SELECT score(), ... FROM ...
WHERE ...
  AND (    dateofbirth IS NULL
        OR dateofbirth IS NOT NULL
      )
  AND ...
```

4.3.4.6.1.3 Finding NULL Values When Searching for an Empty String With emptyMatchesNull

In some cases a search application does not distinguish empty strings from null values. It is expected that a search for an empty string returns all empty strings and all null values, but values that are not empty shall not be returned. This behavior can be enabled by setting `emptyMatchesNull=true`.

This option has an effect on the search result if `emptyScore` is not set, otherwise the `emptyMatchesNull` parameter is ignored. If the search term is not empty, the `emptyMatchesNull` option does not change the search result.

Source Code

```
SELECT score(), ... FROM ...
WHERE ...
  AND CONTAINS(firstname, '', FUZZY(0.8, 'textSearch=compare,
emptyMatchesNull=true, ...<otherOptions>...))
  AND ...
```

The result of the search is the same as with the following SQL statement (scores differ of course because of the `emptyScore` parameter):

Source Code

```
SELECT score(), ... FROM ...
WHERE ...
  AND (     firstname IS NULL
       OR firstname = ''
      )
  AND ...
```

Examples

Select 'eins' with emptyMatchesNull

```
SELECT SCORE() AS score, * FROM test_emptyscore WHERE CONTAINS(T, 'eins',
FUZZY(0.5, 'textSearch=compare, emptyMatchesNull=true')) ORDER BY score DESC, id;
```

Table 76:

SCORE	ID	T
1.0	1	eins

Select empty string with emptyMatchesNull

```
SELECT SCORE() AS score, * FROM test_emptyscore WHERE CONTAINS(T, '', FUZZY(0.5,
'textSearch=compare, emptyMatchesNull=true')) ORDER BY score DESC, id;
```

Table 77:

SCORE	ID	T
1.0	2	
1.0	4	<NULL>

4.3.4.6.2 Option returnAll

Option `returnAll` influences the score by returning all values of a column not matching the defined fuzzy search conditions and giving them the score 0.

The default value of the search option returnAll is `returnAll=off`. In this case, the fuzzy search behavior does not change and is as follows:

A CONTAINS() predicate using a fuzzy search returns all values of a table column that meet the fuzzy search conditions as defined in the FUZZY() predicate. All other values of the column are not returned.

With `returnAll=on` the behavior changes, and all other values of the column are also returned with a score of 0. In other words, a CONTAINS() predicate with `returnAll=on` influences the score only and does not behave like a filter condition (it does not remove any rows from the result set).

> **Note**
>
> Option `returnAll` cannot be used for exact searches (example: `'...CONTAINS(col,'xyz',EXACT('returnAll=on'))...'`).

The following example shows a use case of option `returnAll`. This option is usually used with search rule sets in order to avoid creating additional rules.

Example without returnAll

The following example shows the behavior of a fuzzy search without the option `returnAll`.

```
CREATE COLUMN TABLE customer
(
  id           INTEGER        PRIMARY KEY,
  firstname    NVARCHAR(50),
  lastname     NVARCHAR(50),
  streetname   NVARCHAR(50),
  housenumber  NVARCHAR(20),
  postcode     NVARCHAR(20),
  cityname     NVARCHAR(50)
);
INSERT INTO customer VALUES (1, 'Donna', 'Moore', 'Deer Creek Drive', '3475',
'94304', 'Palo Alto, CA');
INSERT INTO customer VALUES (2, 'Donna', 'More',  'Deer Creek Drive', '1809',
'94304', 'Palo Alto, CA');
INSERT INTO customer VALUES (3, 'Donna', 'Moore', 'Deer Creec Drive', '3477',
'94305', 'Palo Alto CA');
```

The user searches for a customer and expects to find all similar entries in the table.

```
SELECT TO_DECIMAL(SCORE(),3,2) score, * FROM customer
WHERE   CONTAINS(firstname,   'Dona',             FUZZY(0.7))
AND     CONTAINS(lastname,    'Moore',            FUZZY(0.7))
AND     CONTAINS(streetname,  'Deep Creek Drive', FUZZY(0.7))
AND     CONTAINS(housenumber, '3475',             FUZZY(0.7))
AND     CONTAINS(postcode,    '94304',            FUZZY(0.7))
AND     CONTAINS(cityname,    'Palo Alto CA',     FUZZY(0.7));
```

The search returns the following:

Table 78:

SCORE	ID	FIRSTNAME	LASTNAME	STREET-NAME	HOUSENUM-BER	POSTCODE	CITYNAME
0.95	1	Donna	Moore	Deer Creek Drive	3475	94304	Palo Alto, CA
0.91	2	Donna	Moore	Deer Creec Drive	3477	94305	Palo Alto, CA

One row is missing because of the wrong house number, but users expect to find this row. Because of the wrong house number, it is given a lower score than the other rows that contain just a few 'minor' spelling errors.

To get the missing row, there are two options:

- Remove the HOUSENUMBER column from the SELECT statement.
- Add a second SELECT statement that returns all rows with a wrong housenumber.

```
SELECT TO_DECIMAL(SCORE(),3,2) score, * FROM customer
WHERE   CONTAINS(firstname,   'Dona',             FUZZY(0.7))
AND     CONTAINS(lastname,    'Moore',            FUZZY(0.7))
AND     CONTAINS(streetname,  'Deep Creek Drive', FUZZY(0.7))
AND     CONTAINS(housenumber, '3475',             FUZZY(0.7))
AND     CONTAINS(postcode,    '94304',            FUZZY(0.7))
AND     CONTAINS(cityname,    'Palo Alto CA',     FUZZY(0.7));
SELECT TO_DECIMAL(SCORE(),3,2) score, * FROM customer
WHERE   CONTAINS(firstname,   'Dona',             FUZZY(0.7))
AND     CONTAINS(lastname,    'Moore',            FUZZY(0.7))
AND     CONTAINS(streetname,  'Deep Creek Drive', FUZZY(0.7))
AND     CONTAINS(housenumber, '3475',             FUZZY(0.7))
AND     CONTAINS(postcode,    '94304',            FUZZY(0.7))
AND     CONTAINS(cityname,    'Palo Alto CA',     FUZZY(0.7))
UNION
SELECT TO_DECIMAL(0.9 * SCORE(),3,2) score, * FROM customer
WHERE   CONTAINS(firstname,   'Dona',             FUZZY(0.7))
AND     CONTAINS(lastname,    'Moore',            FUZZY(0.7))
AND     CONTAINS(streetname,  'Deep Creek Drive', FUZZY(0.7))
AND NOT CONTAINS(housenumber, '3475',             FUZZY(0.7))
AND     CONTAINS(postcode,    '94304',            FUZZY(0.7))
AND     CONTAINS(cityname,    'Palo Alto CA',     FUZZY(0.7));
```

The search statements returns:

Table 79:

SCORE	ID	FIRSTNAME	LASTNAME	STREET-NAME	HOUSENUM-BER	POSTCODE	CITYNAME
0.95	1	Donna	Moore	Deer Creek Drive	3475	94304	Palo Alto, CA
0.93	2	Donna	More	Deer Creek Drive	1809	94304	Palo Alto, CA
0.91	3	Donna	Moore	Deer Creec Drive	3477	94305	Palo Alto, CA

and

Table 80:

SCORE	ID	FIRSTNAME	LASTNAME	STREET-NAME	HOUSENUM-BER	POSTCODE	CITYNAME
0.95	1	Donna	Moore	Deer Creek Drive	3475	94304	Palo Alto, CA
0.91	3	Donna	Moore	Deer Creec Drive	3477	94305	Palo Alto, CA
0.84	2	Donna	More	Deer Creek Drive	1809	94304	Palo Alto, CA

The second statement gives the sort order the user expects. It also runs more slowly than the first statement however, because two searches are executed, and the results are combined to the final result set.

Example with returnAll=on

The following example shows the behavior of a fuzzy search with option `returnAll=on`.

```
CREATE COLUMN TABLE customer
(
  id            INTEGER         PRIMARY KEY,
  firstname     NVARCHAR(50),
  lastname      NVARCHAR(50),
  streetname    NVARCHAR(50),
  housenumber   NVARCHAR(20),
  postcode      NVARCHAR(20),
  cityname      NVARCHAR(50)
);
INSERT INTO customer VALUES (1, 'Donna', 'Moore', 'Deer Creek Drive', '3475',
'94304', 'Palo Alto, CA');
INSERT INTO customer VALUES (2, 'Donna', 'More',  'Deer Creek Drive', '1809',
'94304', 'Palo Alto, CA');
INSERT INTO customer VALUES (3, 'Donna', 'Moore', 'Deer Creec Drive', '3477',
'94305', 'Palo Alto CA');
```

To obtain the expected behavior in a single search statement, search option returnAll can be used.

Because of the high influence of a column with a score of 0 on the overall score, the weight of the column is reduced to 0.2.

```
SELECT TO_DECIMAL(SCORE(),3,2) score, * FROM customer
WHERE   CONTAINS(firstname,   'Dona',             FUZZY(0.7))
AND     CONTAINS(lastname,    'Moore',            FUZZY(0.7))
AND     CONTAINS(streetname,  'Deep Creek Drive', FUZZY(0.7))
AND     CONTAINS(housenumber, '3475',             FUZZY(0.7, 'returnAll=on'),
WEIGHT(0.2))
AND     CONTAINS(postcode,    '94304',            FUZZY(0.7))
AND     CONTAINS(cityname,    'Palo Alto CA',     FUZZY(0.7));
```

The search returns all expected rows:

Table 81:

SCORE	ID	FIRSTNAME	LASTNAME	STREET-NAME	HOUSENUM-BER	POSTCODE	CITYNAME
0.95	1	Donna	Moore	Deer Creek Drive	3475	94304	Palo Alto, CA
0.92	3	Donna	Moore	Deer Creec Drive	3477	94305	Palo Alto, CA
0.89	2	Donna	More	Deer Creek Drive	1809	94304	Palo Alto, CA

Related Information

Search Rules [page 251]

4.3.4.7 Fuzzy Search on String Columns

This topic describes the fuzzy search features that are available for string columns (such as SQL type NVARCHAR). Fuzzy string search also provides a number of special algorithms for content, such as house numbers or postcodes.

String types support a basic fuzzy string search. The values of a column are compared with the user input using the fault-tolerant fuzzy string comparison.

When working with string types, the fuzzy string compare always compares the full strings. When searching with 'SAP' for example, a record like 'SAP Deutschland AG & Co. KG' gets a very low score, because only a very small part of the string is equal (3 of 27 characters match).

A fuzzy search on string types is a replacement for non-fault tolerant SQL statements like

```
SELECT ... FROM products WHERE product_name = 'coffe krisp biscuit' ...
```

which would not return any results because of the spelling errors.

The following SQL data types are supported:

- VARCHAR
- NVARCHAR

It is possible to speed up the fuzzy search by creating additional data structures called 'fuzzy search indexes'. These are used for faster calculation of the fuzzy score. These indexes exist in the memory only, so no additional disk space is needed.

To achieve the best possible response, you should enable the fuzzy search indexes for all database columns with a high load of fuzzy searches and for all database columns that are used in performance-critical queries.

The following search options influence the score calculation:

- Option `spellCheckFactor`: Defines the score of terms that get a fuzzy score of 1.0 but are not equal.
- Option `similarCalculationMode`: Defines how the score is calculated when comparing terms. Defines options to search with substrings of terms.
- Option `emptyScore`: Defines the score of empty column values when searching with non-empty user input (and vice versa).

4.3.4.7.1 Speeding Up the Fuzzy Search with the Fuzzy Search Index

You can speed up the fuzzy search on string types by creating a special data structure called a fuzzy search index.

The additional index will increase the total memory footprint of the loaded table. In unfavourable cases the memory footprint of the column can be doubled.

```
CREATE COLUMN TABLE mytable
(
  id    INTEGER        PRIMARY KEY,
  col1  VARCHAR(100)   FUZZY SEARCH INDEX ON,
  col2  NVARCHAR(100)  FUZZY SEARCH INDEX ON
);
```

Additional performance improvements are possible when creating database indexes on the columns.

```
CREATE INDEX myindex1 ON mytable(col1);
CREATE INDEX myindex2 ON mytable(col2);
```

The state of the fuzzy search index can be changed at a later point in time by using the ALTER TABLE statement.

```
ALTER TABLE mytable ALTER
(
  col1  VARCHAR(100)   FUZZY SEARCH INDEX OFF,
  col2  NVARCHAR(100)
);
```

The view SYS.TABLE_COLUMNS shows the current state of the fuzzy search index. When working with attribute views, this information is also visible in SYS.VIEW_COLUMNS.

```
SELECT column_name, data_type_name, fuzzy_search_index
```

```
FROM table_columns
WHERE table_name = 'MYTABLE';
```

4.3.4.7.2 Substring-Optimized Fuzzy Search on String Columns

String columns (VARCHAR and NVARCHAR) are not suitable for finding small parts of a longer string, for example finding 'test' in 'this is a long test run'. This is more the domain of TEXT fields that have an additional FULLTEXT INDEX. To find a short string in what might be longer values of string columns (VARCHAR and NVARCHAR), a special search mode `scm=substringsearch` has been established. The substring search can be used as a work-around if there is no option to perform the search on a TEXT column. The results will be similar but not identical.

4.3.4.7.2.1 Score Calculation

`search` is a `similarCalculationMode` that is suitable for performing a substring-optimized fuzzy search on string columns. However, it also has the following shortcomings:

- Based on the algorithm, it finds longer texts that are not similar in a human sense with a particularly good score.
- This also impairs the sorting of the result set.

To overcome these issues, a new type of `similarCalculationMode` has been introduced, known as the `subStringSearch`. Search option `scm=substringsearch` returns all rows containing the search input. Very long search strings are truncated after 100 characters. A fault tolerance of up to 8 typographical errors for long search strings limits the result list.

See also the example listed in the topic under Related Links.

Results with similarCalulationMode=search

```
SELECT SCORE(), STRING FROM TABLE WHERE CONTAINS (STRING, 'test', FUZZY(0.5,
'similarCalculationMode=search'))
```

Table 82:

SCORE()	STRING
1.0	test
0.93	long test
0.91	this is a long test run

SCORE()	STRING
0.68	this is a long text column
0.68	this is not a result
0.62	text
0.58	long text

Results with similarCalulationMode=subStringSearch

```
SELECT SCORE(), STRING FROM TABLE WHERE CONTAINS (STRING, 'test', FUZZY(0.7,
'similarCalculationMode=substringsearch'))
```

Table 83:

SCORE()	STRING
1.0	test
0.97	long test
0.95	this is a long test run
0.8	text
0.77	long text
0.76	this is a long text column

As shown in the example, all results containing the search term 'test' are sorted on top, followed by the results containing the term 'text'. The record 'this is not a result' is sorted out.

Related Information

Option similarCalculationMode [page 158]

4.3.4.7.2.2 Comparison of subStringSearch on String Columns And Text Search on Text Columns

Hits and Score

The hits from a `subStringSearch` on string column are determined in a completely different way to the hits from a text search on a text column. The scoring calculation uses a different algorithm too. The results

therefore cannot be the same. They are similar however. The following example shows a comparison of a `subStringSearch` with a very similar text search.

Example:

```
DROP TABLE tab;
CREATE COLUMN TABLE tab (
  id        INTEGER primary key,
  str       NVARCHAR(120) fuzzy search index on,
  txt       TEXT fuzzy search index on);
INSERT INTO tab VALUES (0, 'Albia City', 'Albia City');
INSERT INTO tab VALUES (1, 'Albia', 'Albia');
INSERT INTO tab VALUES (2, 'Albia City Hall', 'Albia City Hall');
INSERT INTO tab VALUES (3, 'City Of Albia', 'City Of Albia');
INSERT INTO tab VALUES (4, 'Albion City Park', 'Albion City Park');
INSERT INTO tab VALUES (5, 'Albiacityhall', 'Albiacityhall');
INSERT INTO tab VALUES (6, 'Albiacity', 'Albiacity');
INSERT INTO tab VALUES (7, 'Cityalbia', 'Cityalbia');
MERGE DELTA OF tab;
SELECT TO_DECIMAL(SCORE(),3,2) score, txt FROM tab WHERE CONTAINS(txt, 'olbia
city',
FUZZY(0.75, 'textsearch=compare, considernonmatchingtokens=input,
composewords=5, decomposewords=5'))
ORDER BY SCORE() DESC;
```

Table 84:

SCORE()	TXT
0.9	Albia City
0.9	Albia City Hall
0.9	City Of Albia
0.81	Albiacity
0.77	Cityalbia

```
SELECT TO_DECIMAL(SCORE(),3,2) score, str FROM tab WHERE CONTAINS(str, 'olbia
city',
FUZZY(0.75,
'similarcalculationmode=substringsearch'))

ORDER BY SCORE() DESC;
```

Table 85:

SCORE()	STR
0.89	Albia City
0.88	City Of Albia
0.88	Albia City Hall
0.82	Albiacity
0.82	Cityalbia
0.78	Albiacityhall

`SubStringSearch` uses a strict AND. Records like "Albia" are therefore not returned, as "City" is missing in the input. As for longer reference entries and additional or missing blanks, it is quite tolerant. Wrong, missing or additional characters result in a moderate deduction.

The main difference to a standard string search is that `subStringSearch` suppresses reference hits that just contain widely spread fragments of the search string which are not considered similar to the input by the human eye.

Response Time

The response time of a `subStringSearch` depends on various factors. Internally, it starts with a fuzzy search on the string column. The results are then filtered, and a score is calculated. The time consumed is depends significantly on the number of hits that the fuzzy search returns. In general, the `subStringSearch` using the fuzzy index can take twice as long as the underlying string search. If the `subStringSearch` runs without fuzzy index, it might be faster than the string search, but consumes more CPU time. Compared to a text search, the `subStringSearch` is a few times slower or worse. It should therefore not be used for large amounts of data, and only when text columns are not an option.

Memory Consumption

The memory consumption of a `subStringSearch` is not different to the string search it is based on. It depends of course on whether or not the fuzzy index is used.

4.3.4.7.3 Content-Specific Fuzzy Search on String Columns

4.3.4.7.3.1 Use Case: Fuzzy Search - Postcodes

Postcodes in almost all countries are ordered by region. This means that if the leading characters of the postcodes of two different addresses are the same, the addresses are near to each other. In Germany, for example, addresses within large cities share the first or even the first two digits of their postcode.

The only exception known to the development team is Cambodia, where postcodes are not ordered by region.

When doing a fuzzy search on addresses, it makes sense to return a higher score for postcodes that are 'near' to a given user input than for postcodes that are 'far away' from the user input. It makes sense to give a higher weight to the leading characters and a lower weight to the trailing characters of the postcode.

Valid addresses may contain a country code in front of the postcode (for example, 'D-12345' or 'DE-12345' for a German address). This is also supported by the fuzzy postcode search.

Score Calculation

Before the fuzzy score is calculated, the postcode strings are standardized.

1. Country codes are separated from the postcode strings. Country codes in this case consist of one to three letters (a-z only, no numbers) at the beginning of the postcode, followed by a minus sign. Longer words are not considered a country code because postal standards do not allow country names in front of the postcode.
2. Country codes are standardized to enable a comparison of different codes for the same country, for example, 'D-', 'DE-' and 'DEU-' for German postcodes. All unknown/invalid country codes are standardized to one special 'dummy' country code.
3. Spaces and dashes are removed from the remaining postcode.
4. All letters are standardized to uppercase.

Table 86:

User Input	Country Code	Remaining Postcode
71691		71691
716 91		71691
D-71691	D	71691
DE-71 691	DE	71691
D 71691		D71691
Germany-71691		GERMANY71691
GB-A1H 2ZU	GB	A1H2ZU
A1H-2ZU		A1H2ZU
gb-A1h 2zu	GB	A1H2ZU
XY-12345	XX	12345
zz-12345	XX	12345
AI-2640	AI	2640

The last example is the only known example where the country code is part of the postcode (AI = Anguilla). The algorithm works here as well, since the country code is also compared. The two examples directly above the AI example show invalid country codes. Both are standardized to the same non-existent 'dummy' country code.

Postcode Comparison

The standardized postcodes are compared using a variation of the fuzzy string comparison. This variation gives a higher weight to the first two characters of the postcode.

Country codes are given the same weight as a single character at the end of the postcode.

- Only postcodes with the same country code can get a score of 1.0.
- If one country code is given and the second country code is empty, the score of the postcode comparison is less than 1.0.

- If both country codes are given and are different, the score of the postcode comparison is also less than 1.0.

Parameter similarCalculationMode

The search option 'similarCalculationMode' with options 'search' and 'symmetricsearch' is available for postcode columns.

When using the search option 'similarCalculationMode', a postcode search with a postcode prefix will find all addresses in a given area.

- A search with '71' returns all postcodes beginning with '71'.
- A search with '1234' returns all postcodes starting with a sequence similar to '1234' and, with a lower score, all postcodes that contain a '1234'.

Parameter spellCheckFactor

Two postcodes may be considered identical by the fuzzy string comparison, but may still be different. In this case, the value of the parameter 'spellCheckFactor' is applied and the score is multiplied by the spellCheckFactor.

Examples of non-equal postcodes that get a score of 1.0 are:

- '123456' and '12 34 56'
- '7070717' and '7071707'

The default value of the search option spellCheckFactor is 0.9. To disable this feature, set 'spellCheckFactor=1.0'.

Example

The following example uses a spellCheckFactor of 1.0, which is not the default value.

Table 87:

Postcode 1	Postcode 2	Score	Remarks
71691	71691	1.0	
71691	71 691	1.0	
71691	81691	0.51	Highest weight on the first digit
71691	72691	0.7	High weight on the sec-ond digit

Postcode 1	Postcode 2	Score	Remarks
71691	71692	0.96	Lower weight on all other digits
71691	D-71691	0.96	Country code missing in one column
D-71691	A-71691	0.96	Country codes are different
71691	D-71692	0.92	
D-71691	A-71692	0.92	
GB-A1H 2ZU	Gb-a1h2zu	1.0	
XX-12345	YY-12345	1.0	Invalid country codes are 'equal'
D-12345	YY-12345	0.96	Valid and invalid country code

SQL Syntax

(N)VARCHAR columns have to be defined as postcode columns to enable the fuzzy postcode search. You do this using the FUZZY SEARCH MODE clause.

You can also improve the performance of the postcode search by activating a fuzzy search index and by creating a database index on the postcode column.

```
CREATE COLUMN TABLE tab
(
  id        INTEGER       PRIMARY KEY,
  postcode  NVARCHAR(20)  FUZZY SEARCH INDEX ON FUZZY SEARCH MODE 'postcode'
);
CREATE INDEX myindex1 ON tab(postcode);
```

You can enable or disable the postcode search at a later point in time with the ALTER TABLE statement. To disable the postcode search, do not specify the FUZZY SEARCH MODE for the postcode column.

```
-- enable postcode search
ALTER TABLE tab ALTER
(
  postcode  NVARCHAR(100)  FUZZY SEARCH MODE 'postcode'
);
-- disable postcode search
ALTER TABLE tab ALTER
(
  postcode  NVARCHAR(100)  FUZZY SEARCH MODE NULL
);
-- do not change the status of the search mode
ALTER TABLE tab ALTER
(
  postcode  NVARCHAR(100)
);
```

You can query the status of the fuzzy search index and the fuzzy search mode from the system view TABLE_COLUMNS.

```
SELECT column_name, data_type_name, fuzzy_search_index, fuzzy_search_mode
FROM table_columns
WHERE table_name = 'TAB';
```

Example

```
CREATE COLUMN TABLE postcodes
(
  postcode NVARCHAR(50) FUZZY SEARCH INDEX ON FUZZY SEARCH MODE 'postcode'
);
INSERT INTO postcodes VALUES ('71691');
INSERT INTO postcodes VALUES ('81691');
INSERT INTO postcodes VALUES ('72691');
INSERT INTO postcodes VALUES ('71692');
INSERT INTO postcodes VALUES ('716 91');
INSERT INTO postcodes VALUES ('A1H 2ZU');
INSERT INTO postcodes VALUES ('A1H2ZU');
INSERT INTO postcodes VALUES ('D-71691');
INSERT INTO postcodes VALUES ('D-71692');
INSERT INTO postcodes VALUES ('A-71691');
INSERT INTO postcodes VALUES ('A-71692');
INSERT INTO postcodes VALUES ('DE-71 691');
INSERT INTO postcodes VALUES ('D 71691');
INSERT INTO postcodes VALUES ('GB-A1H 2ZU');
INSERT INTO postcodes VALUES ('XX-12345');
INSERT INTO postcodes VALUES ('D-12345');
INSERT INTO postcodes VALUES ('71234');
SELECT TO_DECIMAL(SCORE(),3,2), *
FROM postcodes
WHERE CONTAINS(postcode, '71691', FUZZY(0.5, 'spellCheckFactor=1.0'))
ORDER BY SCORE() DESC;
SELECT TO_DECIMAL(SCORE(),3,2), *
FROM postcodes
WHERE CONTAINS(postcode, 'D-71691', FUZZY(0.5, 'spellCheckFactor=1.0'))
ORDER BY SCORE() DESC;
SELECT TO_DECIMAL(SCORE(),3,2), *
FROM postcodes
WHERE CONTAINS(postcode, 'Gb-a1h2zu', FUZZY(0.5, 'spellCheckFactor=1.0'))
ORDER BY SCORE() DESC;
SELECT TO_DECIMAL(SCORE(),3,2), *
FROM postcodes
WHERE CONTAINS(postcode, 'YY-12345', FUZZY(0.5, 'spellCheckFactor=1.0'))
ORDER BY SCORE() DESC;
SELECT TO_DECIMAL(SCORE(),3,2), *FROM postcodes
WHERE CONTAINS(postcode, '71', FUZZY(0.5, 'spellCheckFactor=1.0'))
ORDER BY SCORE() DESC;
SELECT TO_DECIMAL(SCORE(),3,2), *
FROM postcodes
WHERE CONTAINS(postcode, '1234', FUZZY(0.5, 'spellCheckFactor=1.0'))
ORDER BY SCORE() DESC;
```

4.3.4.7.3.2 Use Case: Fuzzy Search - House Numbers

Score Calculation

The house number comparison aims for a 'simple' solution that is easy to understand, gives good results, and works for most countries. The limitations of the algorithm are:

- The algorithm focuses on numeric values - either a single number ('8') or a range of numbers ('8 - 12').
- House number additions (for example, the 'a' in '8a') are either equal or not equal.

When comparing two strings containing house numbers with each other, the score is calculated in accordance with the rules described below.

House number addition. A house number addition in terms of this backlog item is any additional text that is written before or after the numeric value of a house number.

House number ranges. When a string contains at least two numbers and there is a dash between the first and second number, this is treated as a house number range. The first number is the lower bound of the range, the last number is the upper bound.

Multiple numbers. When multiple numbers are part of a house number string that does not define a house number range, the first number is the house number used for the comparison. All remaining information is used as a house number addition.

Whitespace characters. For all rules, whitespace characters are ignored when comparing the house numbers. For the score calculation it does not matter if a house number is given as '8a' or '8 a' or if it is '8-10' or '8 - 10'.

Symmetry. In all examples, the score calculation is symmetric. This means that either string 1 or string 2 can be the user input and the other string is stored in the database table.

Rule 1 - House Numbers or House Number Ranges Are Identical

For identical house numbers, a score of 1.0 is returned. Identical house numbers are house number strings that are equal when whitespace characters are ignored.

Examples:

Table 88:

String 1	String 2	Score
5	5	1.0
5a	5 a	1.0
8-12	8-12	1.0
9 in the backyard	9 in the backyard	1.0

Rule 2 - House Numbers or House Number Ranges Are Very Similar (House Number Additions Are Different)

House numbers or house number ranges are considered very similar when the numerical values are identical but the additional information differs.

Examples:

Table 89:

String 1	String 2	Score
5	5 a	0.9
5a	5 b	0.9
5	Nr. 5	0.9
8-12	8 - 12a	0.9
8-12	Nr. 8-12	0,9
8-12	8 - 12/5	0.9
8 this is a long text -12	8 - 12a	0.9
7	below 7	0.9
9	9 in the backyard	0.9
in the backyard 9	9 in the backyard	0.9

Rule 3 - House Numbers or House Number Ranges Are Less Similar

House numbers and house number ranges are considered less similar in the following cases:

1. A house number is compared to a house number range and the numerical value of the house number equals the lower or upper bound of the range.
2. Two house number ranges are compared and the numerical values of either the lower or upper bounds are equal.

Table 90:

String 1	String 2	Score
8	8-12	0.8
12a	8-12	0.8
8-10	8-12	0.8
8-10	8-12	0.8
10-12a	8-12	0.8
8 in the backyard	8-12	0.8

Rule 4 - Overlapping House Number Ranges

House numbers and house number ranges overlap in the following cases:

1. A house number is compared to a house number range and the numerical value of the house number lies within the range.
2. Two house number ranges are compared and the ranges overlap.

Examples:

Table 91:

String 1	String 2	Score
10	8-12	0.7
10a	8-12	0.7
9	8-12	0.7
8-12	10-14	0.7
8-12a	10b-14	0.7

Last Rule - House Numbers Are Not Equal

Examples:

Table 92:

String 1	String 2	Score
5	6	0.0
8a	9a	0.0
6	8-12	0.0
8-10	12-14	0.0

House Number Columns and Other String Search Options

The following search options available for string column types are not valid for house number columns:

- **SpellCheckFactor**: When comparing house numbers, the search option 'spellCheckFactor' is ignored. So for house numbers, the results are always the same as with 'spellCheckFactor=1.0'.
- **SimilarCalculationMode**: When comparing house numbers, the search option 'similarCalculationMode' is ignored and has no effect on the search result.

Both options are ignored. No error is returned when any of the options is given.

SQL Syntax

To enable the search for house numbers on an (N)VARCHAR column, the FUZZY SEARCH MODE clause is used in a CREATE TABLE statement.

```
CREATE COLUMN TABLE tab
(
  id   INTEGER         PRIMARY KEY,
  col1 NVARCHAR(20)    FUZZY SEARCH MODE 'housenumber'
);
```

To enable or disable the house number search mode at a later point in time, use the ALTER TABLE statement. The fuzzy search mode is not changed if the FUZZY SEARCH MODE clause is omitted.

```
-- enable housenumber search
ALTER TABLE tab ALTER
(
  col1 VARCHAR(20)    FUZZY SEARCH MODE 'housenumber'
);
-- disable housenumber search
ALTER TABLE tab ALTER
(
  col1 VARCHAR(20)    FUZZY SEARCH MODE NULL
);
-- do not change the status of the search mode
ALTER TABLE tab ALTER
(
  col1 VARCHAR(20)
);
```

You can query the state of the fuzzy search mode using the system view TABLE_COLUMNS.

```
SELECT column_name, data_type_name, fuzzy_search_mode
FROM table_columns
WHERE table_name = 'TAB';
```

Note

You cannot use a fuzzy search index in combination with the house number search mode.

Example

The following example creates a table that contains only a single house number column and executes some searches on this column.

```
CREATE COLUMN TABLE housenumbers
(
  housenumber NVARCHAR(50) FUZZY SEARCH MODE 'housenumber'
);
INSERT INTO housenumbers VALUES ('5');
INSERT INTO housenumbers VALUES ('5a');
INSERT INTO housenumbers VALUES ('5 a');
INSERT INTO housenumbers VALUES ('Nr. 5');
INSERT INTO housenumbers VALUES ('8-12');
INSERT INTO housenumbers VALUES ('8 - 12');
```

```
INSERT INTO housenumbers VALUES ('8 - 12a');
INSERT INTO housenumbers VALUES ('Nr. 8-12');
INSERT INTO housenumbers VALUES ('8 - 12/5');
INSERT INTO housenumbers VALUES ('8');
INSERT INTO housenumbers VALUES ('12a');
INSERT INTO housenumbers VALUES ('8-10');
INSERT INTO housenumbers VALUES ('10-12a');
INSERT INTO housenumbers VALUES ('10a');
INSERT INTO housenumbers VALUES ('10-14');
INSERT INTO housenumbers VALUES ('9');
SELECT TO_DECIMAL(SCORE(),3,2), *
FROM housenumbers
WHERE CONTAINS(housenumber, '5', FUZZY(0.8))
ORDER BY TO_DECIMAL(SCORE(),3,2) DESC;
SELECT TO_DECIMAL(SCORE(),3,2), *
FROM housenumbers
WHERE CONTAINS(housenumber, 'Nr. 5', FUZZY(0.8))
ORDER BY TO_DECIMAL(SCORE(),3,2) DESC;
SELECT TO_DECIMAL(SCORE(),3,2), *
FROM housenumbers
WHERE CONTAINS(housenumber, '8a-12', FUZZY(0.8))
ORDER BY TO_DECIMAL(SCORE(),3,2) DESC;
SELECT TO_DECIMAL(SCORE(),3,2), *
FROM housenumbers
WHERE CONTAINS(housenumber, '10-12', FUZZY(0.8))
ORDER BY TO_DECIMAL(SCORE(),3,2) DESC;
SELECT TO_DECIMAL(SCORE(),3,2), *
FROM housenumbers
WHERE CONTAINS(housenumber, '9 in the BACKYARD', FUZZY(0.8))
ORDER BY TO_DECIMAL(SCORE(),3,2) DESC;
```

4.3.4.7.4 Speeding Up the Fuzzy Search on String Columns

Context

Fuzzy search on string types can be sped up by creating a special data structure called a fuzzy search index. The additional index will increase the total memory footprint of the loaded table. In unfavourable cases the memory footprint of the column can be doubled.

```
CREATE COLUMN TABLE mytable
(
  id    INTEGER       PRIMARY KEY,
  col1  VARCHAR(100)  FUZZY SEARCH INDEX ON,
  col2  NVARCHAR(100) FUZZY SEARCH INDEX ON
);
```

Additional performance improvements are possible when creating database indexes on the columns.

```
CREATE INDEX myindex1 ON mytable(col1);
CREATE INDEX myindex2 ON mytable(col2);
```

The state of the fuzzy search index can be changed at a later point in time by using the ALTER TABLE statement.

```
ALTER TABLE mytable ALTER
```

```
(
  col1  VARCHAR(100)  FUZZY SEARCH INDEX OFF,
  col2  NVARCHAR(100)
);
```

The view SYS.TABLE_COLUMNS shows the current state of the fuzzy search index. When working with attribute views, this information is also visible in SYS.VIEW_COLUMNS.

```
SELECT column_name, data_type_name, fuzzy_search_index
FROM table_columns
WHERE table_name = 'MYTABLE';
```

4.3.4.8 Fuzzy Search on Text Columns

This topic describes the fuzzy search features that are available for text columns (such as SQL type SHORTTEXT) and for columns with an additional FULLTEXT INDEX.

Text types support a more sophisticated kind of fuzzy search. Texts are tokenized (split into terms), and the fuzzy comparison is performed term by term. When searching with 'SAP' for example, a record like 'SAP Deutschland AG & Co. KG' gets a high score, because the term 'SAP' exists in both texts. A record like 'SAPPHIRE NOW Orlando' gets a lower score, because 'SAP' is just a part of the longer term 'SAPPHIRE' (3 of 8 characters match).

Fuzzy search on text columns replaces non-fault tolerant statements like

```
SELECT ... FROM documents WHERE doc_content LIKE '% Driethanolamyn %' ...
```

The following SQL data types are supported:

- TEXT
- SHORTTEXT
- fulltext index

A fulltext index is an additional index structure that can be defined for non-text columns to add text search features. Supported column types include NCLOB and NVARCHAR.

It is possible to speed up the fuzzy search by creating data structures called 'fuzzy search indexes', which are used for faster calculation of the fuzzy score. These indexes exist in the memory only, so no additional disk space is needed.

To achieve the best possible response times, you should enable the fuzzy search indexes for all database columns with a high load of fuzzy searches and for all database columns that are used in performance-critical queries.

Fuzzy Search on SQL Type TEXT

A call to contains that references a TEXT column is automatically processed as a text search. In this case, the mode textsearch=compare and all fuzzy search options are allowed:

```
CREATE COLUMN TABLE mytable
(
```

```
  col1 TEXT
);
SELECT score() AS score, * FROM mytable WHERE contains(col1, 'a b', fuzzy(0.8,
'textsearch=compare'));
```

Fuzzy Search on SQL Type SHORTTEXT

When a SHORTTEXT column is created, a column of column store type cs_string and a second hidden text column are created. A call to contains that references the SHORTTEXT column is automatically redirected by the freestyler to the additional hidden TEXT column. In this case, the mode textsearch=compare and all fuzzy search options are allowed:

```
CREATE COLUMN TABLE mytable
(
  col1 SHORTTEXT(200)
);
SELECT score() AS score, * FROM mytable WHERE contains(col1, 'a b', fuzzy(0.8,
'textsearch=compare'));
```

Fuzzy Search on a FULLTEXT INDEX

When a full-text index is created on a column that is not of type TEXT (e.g. NVARCHAR, NCLOB, ...) a hidden text column is added to the table. A call to contains that references the non-TEXT column is automatically redirected by the freestyler to the additional text column. In this case, the mode textsearch=compare and all fuzzy search options are allowed:

```
CREATE COLUMN TABLE mytable
(
  col1 NVARCHAR(2000)
);
CREATE FULLTEXT INDEX myindex ON mytable(col1);
SELECT score() AS score, * FROM mytable WHERE contains(col1, 'a b', fuzzy(0.8,
'textsearch=compare'));
```

Merge Delta for Better Performance

When inserting or loading a large number of rows into a table that has a TEXT or SHORTTEXT column or uses a FULLTEXT INDEX, it is important to merge the delta part of the table in order to ensure satisfactory search performance.

A delta merge can be started manually using the following SQL statement:

```
MERGE DELTA OF mytable;
```

Alternatively, a delta merge can be triggered automatically by the mergedog process.

4.3.4.8.1 Multi-Token Search on Text Columns

When using more than one token in a query, the default content type is AND (for example, ... WHERE CONTAINS (mycolumn, 'software firm', FUZZY(0.5)) ... will return entries that contain a token similar to 'software' and a token similar to 'firm').

Alternatively, you can use OR by adding the key word between the tokens (for example, ... WHERE CONTAINS (mycolumn, 'apple OR 'banana', FUZZY(0.5)) ... will return entries that contain a token similar to 'apple' and entries that contain a token similar to 'banana').

PHRASE is similar to AND, but restricts hits to ones that contain the tokens as a phrase, in other words, in the same order and with nothing between them. A PHRASE is indicated by adding double quotes around the tokens, within the single quotes (for example, ... WHERE CONTAINS (mycolumn, '"day dream"', FUZZY(0.5)) ... will not return an entry containing 'I dream of a day').

The content type AND that is used for full-text searches (default behavior: textSearch=fulltext) is implemented as a logical AND to achieve better performance. A search for 'Miller & Miller AG' for example, with content type AND, matches 'Miller AG'.

For duplicate detection, and for comparing company names, product names and so on, use textSearch=compare. This produces better search results due to the strict AND comparison that is used. In other words, when searching for 'Miller & Miller' with content type AND, only records that contain the term 'Miller' at least twice will be returned.

A strict AND assigns terms from the user input to terms in the database entry just once (and vice versa). For more information, see Partially Matching with Parameter andThreshold [page 206].

Parameters Influencing the Score

Table 93:

Name of Option	Range	Default	Applies to Types
`bestMatchingTokenWeight`	0.0..1.0	0	TEXT
`considerNonMatchingTokens`	max, min, all, input, table	max	TEXT
`excessTokenWeight`	0.0..1.0	1.0	TEXT

Formula for score calculation:

score = bestMatchingTokenWeight x max(tokenScores) + (1-bestMatchingTokenWeight) x $\sqrt{}$(∑(tokenScore^2)/(matchedTokenCount + excessTokenCount x excessTokenWeight))

Recommendations for specific search content types

If you are using an "OR" search (searching for "this or that"), you should set `considerNonMatchingTokens` to table to get a useful score assessment.

Parameters Influencing the Result Set

Table 94:

Option	Range	Default	Applies to Types
andSymmetric	on,off,true,false	off	TEXT
andThreshold	0.0..1.0	1.0	TEXT

For examples, see Symmetric Search with Parameter andSymmetric [page 207] and Partially Matching with Parameter andThreshold [page 206] .

Related Information

Option bestMatchingTokenWeight [page 201]
Option considerNonMatchingTokens [page 202]
Option excessTokenWeight [page 204]

4.3.4.8.1.1 Option bestMatchingTokenWeight

Missing tokens in the search input or tokens with a low score can lower the total score for the field more than desired. The parameter `bestMatchingTokenWeight` allows you to get a better score in such cases. This is done by putting more emphasis on the score of the token that matches best. With this parameter, you can shift the total score value for a field between the root mean square of score values and the best token score value. The lower boundary (root mean square of score values) is reached by setting `bestMatchingTokenWeight` to 0, which is the default value. The upper boundary is reached by setting `bestMatchingTokenWeight` to 1.

Examples

```
DROP TABLE test_table;
CREATE COLUMN TABLE test_table
(
  id INTEGER          PRIMARY KEY,
  t  SHORTTEXT(200)   FUZZY SEARCH INDEX ON
);
INSERT INTO test_table VALUES ('1', 'one');
INSERT INTO test_table VALUES ('2', 'one two');
INSERT INTO test_table VALUES ('3', 'one two three');
INSERT INTO test_table VALUES ('4', 'one two three four');
INSERT INTO test_table VALUES ('5', 'one two three four five');
SELECT TO_DECIMAL(SCORE(),3,2) AS score, * FROM test_table
WHERE CONTAINS(t, 'won zwo tree', FUZZY(0.5, 'textSearch=compare'))
ORDER BY score DESC, id;
```

Table 95:

SCORE	ID	T
0.68	3	one two three
0.59	4	one two three four
0.53	5	one two three four five

```
SELECT TO_DECIMAL(SCORE(),3,2) AS score, * FROM test_table
WHERE CONTAINS(t, 'won zwo tree', FUZZY(0.5,'textSearch=compare,
bestMatchingTokenWeight=0.1'))
ORDER BY score DESC, id
```

Table 96:

SCORE	ID	T
0.7	3	one two three
0.61	4	one two three four
0.56	5	one two three four five

```
SELECT TO_DECIMAL(SCORE(),3,2) AS score, * FROM test_table
WHERE CONTAINS(t, 'won zwo tree', FUZZY(0.5,'textSearch=compare,
bestMatchingTokenWeight=1.0'))
ORDER BY score DESC, id
```

Table 97:

SCORE	ID	T
0.81	3	one two three
0.81	4	one two three four
0.81	5	one two three four five

4.3.4.8.1.2 Option considerNonMatchingTokens

To get the score for the whole field content, we have to compute the mean of the token scores. There are cases in which not every token in the search term has a matching token in the table content, and vice versa. These tokens are called non-matching tokens (or excess tokens).

You can use parameter `considerNonMatchingTokens` to decide how these non-matching tokens affect the calculation of the score for the whole field content.

This is done by specifying what is considered to be the number of tokens relevant for the score calculation (`tokenCount`). The number of matching tokens is subtracted from this number to get the number of non-matching tokens (or excess tokens).

`tokenCount` is determined in accordance with parameter `considerNonMatchingTokens` as follows:

- input: use search term token count
- table: use column value token count

- all : use sum of search term and column value token count divided by 2
- min : use smaller value of token counts from search term and column value
- max : use larger value of token counts from search term and column value (default)

Examples

```
DROP TABLE test_table;
CREATE COLUMN TABLE test_table
(
  id INTEGER         PRIMARY KEY,
  t  SHORTTEXT(200)  FUZZY SEARCH INDEX ON
);
INSERT INTO test_table VALUES ('1', 'one');
INSERT INTO test_table VALUES ('2', 'one two');
INSERT INTO test_table VALUES ('3', 'one two three');
INSERT INTO test_table VALUES ('4', 'one two three four');
INSERT INTO test_table VALUES ('5', 'one two three four five');
```

`considerNonMatchingTokens = input`

```
SELECT TO_DECIMAL(SCORE(),3,2) AS score, * FROM test_table
WHERE CONTAINS(t, 'won zwo tree', FUZZY(0.5,'textSearch=compare,
considerNonMatchingTokens=input'))
ORDER BY score DESC, id
```

Table 98:

SCORE	ID	T
0.68	3	one two three
0.68	4	one two three four
0.68	5	one two three four five

`considerNonMatchingTokens = table`

```
SELECT TO_DECIMAL(SCORE(),3,2) AS score, * FROM test_table
WHERE CONTAINS(t, 'won zwo tree', FUZZY(0.5,'textSearch=compare,
considerNonMatchingTokens=table'))
ORDER BY score DESC, id
```

Table 99:

SCORE	ID	T
0.68	3	one two three
0.59	4	one two three four
0.53	5	one two three four five

`considerNonMatchingTokens = all`

```
SELECT TO_DECIMAL(SCORE(),3,2) AS score, * FROM test_table
WHERE CONTAINS(t, 'won zwo tree', FUZZY(0.5,'textSearch=compare,
considerNonMatchingTokens=all'))
ORDER BY score DESC, id
```

Table 100:

SCORE	ID	T
0.68	3	one two three
0.61	4	one two three four
0.59	5	one two three four five

`considerNonMatchingTokens = min`

```
SELECT TO_DECIMAL(SCORE(),3,2) AS score, * FROM test_table
WHERE CONTAINS(t, 'won zwo tree', FUZZY(0.5,'textSearch=compare,
considerNonMatchingTokens=min'))
ORDER BY score DESC, id
```

Table 101:

SCORE	ID	T
0.68	3	one two three
0.68	4	one two three four
0.68	5	one two three four five

`considerNonMatchingTokens = max`

```
SELECT TO_DECIMAL(SCORE(),3,2) AS score, * FROM test_table
WHERE CONTAINS(t, 'won zwo tree', FUZZY(0.5,'textSearch=compare,
considerNonMatchingTokens=max'))
ORDER BY score DESC, id
```

Table 102:

SCORE	ID	T
0.68	3	one two three
0.59	4	one two three four
0.53	5	one two three four five

4.3.4.8.1.3 Option excessTokenWeight

The parameter `excessTokenWeight` defines the weight of excess (that is, unassigned) tokens. It is set to 1.0 by default.

Excess tokens are tokens that do not have a counterpart token on either the input side or the request side. For example, when searching for "Art Garfunkel", the database entry "Art Garfunkel and Paul Simon" has the excess tokens "and", "Paul", "Simon".

This parameter enables a better sorting by score when the lengths (that is, the number of tokens) of the request entry and the reference entry are different.

Examples

```
DROP TABLE test_table;
CREATE COLUMN TABLE test_table
(
  id INTEGER PRIMARY KEY,
  t  TEXT    FUZZY SEARCH INDEX ON
);
INSERT INTO test_table VALUES (1,'Art Garfunkel');
INSERT INTO test_table VALUES (2,'Art Garfunkel and Paul Simon');
INSERT INTO test_table VALUES (3,'A Heart in New York (Art Garfunkel solo hit)');
INSERT INTO test_table VALUES (4,'Tra Funkelrag');
-- select 1
SELECT TO_DECIMAL(SCORE(),3,2) AS score, * FROM test_table
WHERE CONTAINS(t, 'Art Garfunkel',
FUZZY(0.5,'textSearch=compare,excessTokenWeight=1.0'))
ORDER BY score DESC, id;
-- select 2
SELECT TO_DECIMAL(SCORE(),3,2) AS score, * FROM test_table
WHERE CONTAINS(t, 'Art Garfunkel',
FUZZY(0.5,'textSearch=compare,excessTokenWeight=0.1'))
ORDER BY score DESC, id;
```

Result for select 1

Table 103:

SCORE	ID	T
1.0	1	Art Garfunkel
0.74	4	Tra Funkelrag
0.63	2	Art Garfunkel and Paul Simon
0.47	3	A Heart in New York (Art Garfunkel solo hit)

Result for select 2

Table 104:

SCORE	ID	T
1.0	1	Art Garfunkel
0.93	2	Art Garfunkel and Paul Simon
0.86	3	A Heart in New York (Art Garfunkel solo hit)
0.74	4	Tra Funkelrag

4.3.4.8.2 Multi-Token Search with Soft AND

4.3.4.8.2.1 Partially Matching with Parameter andThreshold

It is possible to specify a 'partial AND' that requires a subset of request tokens only to match the reference tokens. You will then get better results when comparing data like company names.

The 'andThreshold' parameter defines the percentage of tokens that have to match when comparing the user input with a row stored in a TEXT column. In other words, the ratio between the number of matching tokens and the number of input tokens has to be greater than or equal to the given andThreshold.

Table 105:

Option Name	Range	Default	Applies to Types	Description
andThreshold	0.0..1.0	1.0	TEXT	Determines the percentage of tokens that need to match

andThreshold = 1.0 -> all tokens have to match, 'strict AND'

0.0 < andThreshold < 1.0 -> some of the tokens have to match, 'soft AND'

andThreshold = 0.0 -> at least one token has to match, 'OR'

Summary

- The andThreshold parameter is available as a searchOption.
- Allowed values are between 0.0 and 1.0. An SQL error is returned for values outside this range.
- The parameter works on TEXT columns only and is ignored for all other SQL types.
- The parameter is used with content type AND only and is ignored for all other content types.
- The parameter influences performance.

Example

```
DROP TABLE test_soft_and;
CREATE COLUMN TABLE test_soft_and
(
  id INTEGER         PRIMARY KEY,
  t  SHORTTEXT(200)  FUZZY SEARCH INDEX ON
);
INSERT INTO test_soft_and VALUES ('1','eins');
INSERT INTO test_soft_and VALUES ('2','eins zwei');
INSERT INTO test_soft_and VALUES ('3','eins zwei drei');
INSERT INTO test_soft_and VALUES ('4','eins zwei drei vier');
INSERT INTO test_soft_and VALUES ('5','eins zwei drei vier funf');
```

Search with andThreshold

```
SELECT SCORE() AS score, * FROM test_soft_and
WHERE CONTAINS(T, 'eins XXX drei vier',
FUZZY(0.5,'andThreshold=0.75,bestMatchingTokenWeight=0.5,textSearch=compare'))
ORDER BY score DESC, id;
```

Table 106:

SCORE	ID	T
0.933012723922729	4	eins zwei drei vier
0.887298345565796	5	eins zwei drei vier funf

4.3.4.8.2.2 Symmetric Search with Parameter andSymmetric

In addition to the parameter andThreshold, it is possible to specify a 'symmetric AND' that also returns a record when all tokens of a database entry are part of the user input. The parameter 'andSymmetric' was added to the fuzzy search and activates the symmetric AND comparison when comparing the user input with a row stored in a TEXT column.

Table 107:

Option Name	Range	Default	Applies to Types	Short Description
andSymmetric	on,off,true,false	off	TEXT	Activates a symmetric AND content search

Example

When using content type AND, a search with 'SAP Deutschland AG' returns 'SAP Deutschland AG & Co KG' but not 'SAP Deutschland' or 'SAP Walldorf'. When using the symmetric and, the search with 'SAP Deutschland AG' returns 'SAP Deutschland AG & Co KG' and 'SAP Deutschland', but not 'SAP Walldorf'.

Summary

- The parameter andSymmetric is available as a searchOption.
- Allowed values are [on,true] and [off, false]. An SQL error is returned for other values.
- The parameter works on TEXT columns only and is ignored for all other SQL types.
- The parameter is used with content type AND only and is ignored for all other content types.
- The parameter works in combination with andThreshold to activate a symmetric 'soft AND'.
- The parameter influence the performance.

Example

```
DROP TABLE test_soft_and;
CREATE COLUMN TABLE test_soft_and
(
  id INTEGER          PRIMARY KEY,
  t  SHORTTEXT(200)   FUZZY SEARCH INDEX ON
);
INSERT INTO test_soft_and VALUES ('1','one');
INSERT INTO test_soft_and VALUES ('2','one two');
INSERT INTO test_soft_and VALUES ('3','one two three');
INSERT INTO test_soft_and VALUES ('4','one two three four');
INSERT INTO test_soft_and VALUES ('5','one two three four five');
```

Search with andSymmetric=off

```
SELECT SCORE() AS score, * FROM test_soft_and
WHERE CONTAINS(T, 'one two three', FUZZY(0.5,
'andSymmetric=off,bestMatchingTokenWeight=0.5,textSearch=compare'))
ORDER BY score DESC, id;
```

Table 108:

SCORE	ID	T
1	3	one two three
0.933012723922729	4	one two three four
0.887298345565796	5	one two three four five

Search with andSymmetric=on

```
SELECT SCORE() AS score, * FROM test_soft_and
WHERE CONTAINS(T, 'one two three',
FUZZY(0.5,'andSymmetric=on,bestMatchingTokenWeight=0.5,textSearch=compare'))
ORDER BY score DESC, id;
```

Table 109:

SCORE	ID	T
1	3	one two three
0.933012723922729	4	one two three four
0.90824830532074	2	one two
0.887298345565796	5	one two three four five
0.788675129413605	1	one

4.3.4.8.3 Option abbreviationSimilarity

The **abbreviation similarity** option allows you to search for and with initial characters. You can thus find abbreviations when searching with long strings or find long strings when searching with initial characters.

Table 110:

Option Name	Range	Default	Applies to Types	Description
abbreviationSimilarity	0.0..1.0	0.0	TEXT	Defines the similarity that is returned for a matching initial character.

> **Note**
>
> abbreviationSimilarity = 0.0 disables the abbreviation search.

The abbreviationSimilarity option works for TEXT fields only. It is available for all term actions available in SQL, like EXACT, SIMILAR, LINGUISTIC. It accepts values between 0.0 and 1.0 . If the given value for abbreviationSimilarity is out of this range, the system returns a SQL error.

Search for inital characters:

"Peter" finds "P." and "P"

"P." always finds "P" with similarity 1.0

"Hans-Peter" finds "Hans P", "H. Peter" and "H.-P."

Search with initial characters:

"P." and "P" find "Peter"

"P." always finds "P" with similarity 1.

"Hans P" and "H P" find "Hans-Peter"

Example

The abbreviationSimilarity option is used to search for and find a word using its first character and vice versa with a given score. With abbreviationSimilarity = 0.9, a SELECT retrieves “word” with SCORE = 0.9 if you search with “w” (and vice versa).

```
CREATE COLUMN TABLE abbrev
(
  id   INTEGER         PRIMARY KEY,
  name SHORTTEXT(200)  FUZZY SEARCH INDEX ON
);
INSERT INTO abbrev VALUES ('1','Peter');
INSERT INTO abbrev VALUES ('2','Hans');
INSERT INTO abbrev VALUES ('3','H.');
INSERT INTO abbrev VALUES ('4','P.');
INSERT INTO abbrev VALUES ('5','Hans-Peter');
INSERT INTO abbrev VALUES ('6','H.-P.');
INSERT INTO abbrev VALUES ('7','HP');
INSERT INTO abbrev VALUES ('8','G Gerd');
INSERT INTO abbrev VALUES ('9','G');
INSERT INTO abbrev VALUES ('10','Gerd');
```

Search one token with abbreviationSimilarity

```
SELECT SCORE() AS score, id, name FROM abbrev
WHERE CONTAINS(name, 'HP', FUZZY(0.5,
'abbreviationSimilarity=0.80,textSearch=compare'))
ORDER BY score DESC, id;
```

Table 111:

SCORE	ID	NAME
1	7	HP
0.800000011920929	3	H.
0.565685451030731	6	H.-P.

Search two tokens with abbreviationSimilarity

```
SELECT SCORE() AS score, id, name FROM abbrev
WHERE CONTAINS(name, 'Hans Peter',FUZZY(0.5,
'abbreviationSimilarity=0.80,textSearch=compare'))
```

```
ORDER BY score DESC, id;
```

Table 112:

SCORE	ID	NAME
1	5	Hans-Peter
0.800000011920929	6	H.-P.

Search two tokens with abbreviationSimilarity

```
SELECT SCORE() AS score, id, name FROM abbrev
WHERE CONTAINS(name, 'Hans Peter',FUZZY(0.5,
'abbreviationSimilarity=0.80,textSearch=compare'))
ORDER BY score DESC, id;
```

Table 113:

SCORE	ID	NAME
0.905538558959961	8	G Gerd

4.3.4.8.4 Option minTextScore

The *minTextScore* option allows you to set the score a text field has to reach to be a match.

A text field can contain more than one token. Each token is evaluated against the fuzzySimilarity parameter which defines the minimum score a single token has to reach. The overall score of a text field differs from the fuzzySimilarity because the overall Text score is computed from the fuzzySimilarity of all tokens and parameters found, like bestMatchingTokenWeight, considerNonMatchingTokens and term mapping weights.

minTextScore defines the minimum score the whole content of a text field has to reach.

Table 114:

Option Name	Range	Default	Applies to Types	Description
minTextScore	0.0..1.0	0.0	TEXT	Minimum score that all records in the result have to reach.

A minTextScore value of 0.0 means that all records that match the conditions defined by fuzzySimilarity and the search options are returned. The result list will not be cut.

Note

- If you use a *fuzzySimilarity* of 0.0, the minTextScore parameter becomes redundant.
- minTextScore does not work with a freestyle or fulltext search.
- When using minTextScore in combination with the emptyScore parameter, rows found because of the emptyScore parameter are returned even if emptyScore is less than minTextScore. See the code example for emptyScore.

```
DROP TABLE tab_mintextscore;
CREATE COLUMN TABLE tab_mintextscore
```

```
(
  id INTEGER   PRIMARY KEY,
  t  TEXT      FAST PREPROCESS ON FUZZY SEARCH INDEX ON
);
INSERT INTO tab_mintextscore VALUES ('1','Bert');
INSERT INTO tab_mintextscore VALUES ('2','Berta');
INSERT INTO tab_mintextscore VALUES ('3','Bart');
```

Search on a text column

```
SELECT SCORE() AS score, id, t FROM tab_mintextscore
WHERE CONTAINS(t, 'Ernie OR Bert', FUZZY(0.100, 'textSearch=compare,
bestMatchingTokenWeight=0,minTextScore=0.70'))
ORDER BY score DESC, id;
```

Table 115:

SCORE	ID	T	Description
0.7	1	Bert	<< rank value of 0.7 is reached, this is a match
0.65	1	Berta	<< minTextScore not reached - cut off
0.52	1	Bart	<< minTextScore not reached - cut off

4.3.4.8.5 Option textSearch

Option `textSearch` is used to select the search algorithm for TEXT columns:

- **textSearch=fulltext** (default value): A full-text search is performed on a TEXT column. IDF calculation or specialOrRanking (depending on search flags) is used. This is the 'old' NewDB behavior.
- **textSearch=compare**: A search similar to a Fuzzy Double search is performed. Additional search options are enabled.
  The value of the `textSearch` search option defines which other search options are allowed for a search.

Rank calculation

When 'fulltext' is specified, the search is performed using IDF or specialOrRanking (depending on the search flags). The fuzzy score is not calculated.

When 'compare' is specified, the fuzzy score is calculated using `bestMatchingTokenWeight` and `considerNonMatchingTokens`. The user does not have to set these options. In this case, the default values are used.

Example

```
SELECT SCORE() AS score, * FROM test_soft_and
WHERE CONTAINS(T, 'eins', FUZZY(0.5, 'textSearch=compare'))
ORDER BY score DESC, id;
```

Select with 'Fuzzy Score' (`textSearch=compare`)

```
SELECT SCORE() AS score, * FROM test_soft_and
WHERE CONTAINS(T, 'eins', FUZZY(0.5, 'textSearch=compare'))
ORDER BY score DESC, id;
```

Table 116:

SCORE	ID	T
1.0	1	eins
0.7071067690849304	2	eins zwei
0.5773502588272095	3	eins zwei drei

Select with 'IDF Score' (`textSearch=fulltext`)

```
SELECT SCORE() AS score, * FROM test_soft_and
WHERE CONTAINS(T, 'eins', FUZZY(0.5, 'textSearch=fulltext'))
ORDER BY score DESC, id;
```

Table 117:

SCORE	ID	T
0.888888955116272	1	eins
0.6666666865348816	2	eins zwei
0.5333333611488342	3	eins zwei drei

4.3.4.8.6 Option phraseCheckFactor

Option `phraseCheckFactor` defines the score of a search when the terms in a text field are not in the same order as in the user input.

Option `phraseCheckFactor` defines the score of a search when the terms in a text field are not in the same order as in the user input (without any additional terms). The value of this option is multiplied with the overall fuzzy score of a text column if the search terms do not appear in the correct order. Search results returning terms in the exact term order as the search input receive get a higher score (see example below). The default value of this option is 1.0, so it has no influence on the score.

Search option `phraseCheckFactor` is available for TEXT, SHORTTEXT and FULLTEXT INDEXES only. It is allowed for attribute searches only (no freestyle search).

Search option `phraseCheckFactor` is available for CONTAINS() only.

ℹ Note

Search rule sets are not supported at present.

Stopwords in the search input and in the database are ignored when comparing the order of the terms. Stopwords still influence the overall score however.

Example

```
CREATE COLUMN TABLE TAB_TEST (ID INT PRIMARY KEY, TEXT     TEXT FAST PREPROCESS
ON) ;
  INSERT INTO TAB_TEST VALUES ('1','International Business Machines');
  INSERT INTO TAB_TEST VALUES ('2','Business Machines International');
SELECT to_decimal(score(),10,2) as SCORE, ID, TEXT
FROM TAB_TEST
WHERE CONTAINS (TEXT, 'International Business Machines', FUZZY(0.8,
'textSearch=compare, phrasecheckfactor=0.9'))
ORDER BY SCORE DESC, ID;
DROP TABLE TAB_TEST;
```

Table 118:

Score	ID	Text
1.0	1	International Business Machines
0.9	2	Business Machines International (phraseCheckFactor got used)

4.3.4.8.7 Stopwords

4.3.4.8.7.1 Usage

Stopwords are terms that are less significant for a search and are therefore not used to generate the result set. In other words, the search is carried out as though the stopwords did not exist (either in the user input or in the database column).

Stopwords do influence the score that is calculated however. A record with stopwords identical to the user input gets a higher score than a record with differing or missing stopwords.

Stopwords can be defined either as single terms or as stopword phrases consisting of multiple terms. Stopword phrases are only applied when all terms of the stopword appear in exactly the given order.

Use case example: When searching for company names, the legal form (Ltd, SA, AG, and so on) is less significant and less selective than the other parts of the name.

Stopwords are stored in a column-store table with the following format:

```
CREATE COLUMN TABLE mystopwords
(
stopword_id VARCHAR(32) PRIMARY KEY,
list_id VARCHAR(32) NOT NULL,
language_code CHAR(2),
term NVARCHAR(200) NOT NULL
);
```

Stopwords are language-dependent. It is possible to define the language that a stopword is valid for. You can also define stopwords for all languages by not setting a language.

As with term mappings, stopwords can be grouped together in multiple groups. Groups of stopwords are identified by the value of the list_id column that is part of the stopword table.

```
INSERT INTO mystopwords VALUES (1, 'legalform', '', 'Ltd');
INSERT INTO mystopwords VALUES (2, 'legalform', 'de', 'GmbH');
INSERT INTO mystopwords VALUES (3, 'legalform', 'de', 'Gesellschaft mit
beschränkter Haftung');
INSERT INTO mystopwords VALUES (4, 'legalform', 'de', 'AG');
INSERT INTO mystopwords VALUES (5, 'legalform', 'de', 'Aktiengesellschaft');
```

To activate stopwords for a search on a TEXT column, you need to provide two search options (similar to the options used for term mappings):

- stopwordListId=mylist1,mylist2,mylist3
- stopwordTable=[<schemaname>.]<tablename>

```
SELECT TO_DECIMAL(SCORE(),3,2) as score, company FROM mydata
WHERE CONTAINS(company, 'xy gmbh', FUZZY(0.7, 'textsearch=compare,
stopwordTable=MYSTOPWORDS, stopwordListId=legalform'))
ORDER BY score DESC;
```

To activate **language-specific stopwords**, you have to provide the language parameter:

```
SELECT TO_DECIMAL(SCORE(),3,2) as score, company FROM mydata
WHERE CONTAINS(company, 'xy gmbh', FUZZY(0.7, 'textsearch=compare,
stopwordTable=MYSTOPWORDS, stopwordListId=legalform'), language('de'))
ORDER BY score DESC;
```

> **Note**
>
> In this case, all stopwords where language_code is set to 'de' or empty will be used. Any stopwords with other language codes will be ignored.

> **Note**
>
> As a prerequisite the column to be searched in has to have the LANGUAGE parameter set. This requires the parameter FAST PREPROCESS set to OFF for this column.

Stopwords are removed from the search term first. In this example, the result set of the search is the same as for the search '... WHERE CONTAINS(company, 'xy', ...'.

When calculating the score, the fuzzy scores of the non-stopword terms have the biggest influence on the resulting score. Stopwords in the user input and in the database records are also given less weight than the

non-stopword terms, so records with matching stopwords get a higher score than records with differing or missing stopwords.

The result of the above example is as follows:

Table 119:

Score	Company	Comment
1.00	XY GmbH	
0.95	XY	Missing stopword
0.95	XY Aktiengesellschaft	Differing stopword
0.92	XY Gesellschaft mit beschränkter Haftung	Many differing stopwords
0.78	XY Company	Additional non-matching term, no stopword

The value given for stopwordTable can be any valid SQL identifier as defined in the SQL reference manual. If no schema is given, the current one is used. The following examples all reference the same stopword table.

```
SET SCHEMA schema1;
SELECT ... WHERE CONTAINS(c, 'xy', FUZZY(0.7,
'stopwordTable=schema1.mystopwords, ...'))...;
SELECT ... WHERE CONTAINS(c, 'xy', FUZZY(0.7,
'stopwordTable="SCHEMA1"."MYSTOPWORDS", ...'))...;
SELECT ... WHERE CONTAINS(c, 'xy', FUZZY(0.7,
'stopwordTable=mystopwords, ...'))...;
```

Using Multiple Stopword Lists

It is possible to use more than one stopword list in a single call to `CONTAINS()`. The names of the stopword lists have to be specified as a comma-separated list enclosed in double quotation marks.

The following example shows how to specify one or more stopword lists:

```
/* use one stopword list */
SELECT TO_DECIMAL(SCORE(),3,2) as score, company FROM mydata
WHERE CONTAINS(company, 'xy gmbh', FUZZY(0.7, 'textsearch=compare,
stopwordTable=MYSTOPWORDS, stopwordListId=list01'))
ORDER BY score DESC;
/* or */
SELECT TO_DECIMAL(SCORE(),3,2) as score, company FROM mydata
WHERE CONTAINS(company, 'xy gmbh', FUZZY(0.7, 'textsearch=compare,
stopwordTable=MYSTOPWORDS, stopwordListId="list01"'))
ORDER BY score DESC;
/* use multiple stopword lists */
SELECT TO_DECIMAL(SCORE(),3,2) as score, company FROM mydata
WHERE CONTAINS(company, 'xy gmbh', FUZZY(0.7, 'textsearch=compare,
stopwordTable=MYSTOPWORDS, stopwordListId="list01,list02,list03"'))
ORDER BY score DESC;
/* use all stopword lists */
SELECT TO_DECIMAL(SCORE(),3,2) as score, company FROM mydata
WHERE CONTAINS(company, 'xy gmbh', FUZZY(0.7, 'textsearch=compare,
stopwordTable=MYSTOPWORDS, stopwordListId=*'))
ORDER BY score DESC;
```

4.3.4.8.7.2 Stopword Example

```
CREATE COLUMN TABLE stopwords
(
stopword_id VARCHAR(32) PRIMARY KEY,
list_id VARCHAR(32) NOT NULL,
language_code CHAR(2) NOT NULL,
term NVARCHAR(200) NOT NULL
);
CREATE COLUMN TABLE companies
(
id INTEGER PRIMARY KEY,
 companyname  SHORTTEXT(200)  FUZZY SEARCH INDEX ON LANGUAGE COLUMN language
FAST PREPROCESS OFF,
);
INSERT INTO companies VALUES ('1', 'SAP AG');
INSERT INTO companies VALUES ('2', 'SAP Aktiengesellschaft');
INSERT INTO stopwords VALUES ('1', '01', 'de', 'AG');
INSERT INTO stopwords VALUES ('2', '01', 'de', 'Aktiengesellschaft');
INSERT INTO stopwords VALUES ('3', '01', 'de', 'blub');
```

Query 1: User input without stopwords. Stopwords in the database table only.

```
SELECT TO_DECIMAL(SCORE(), 3, 2) AS score, *
FROM companies
WHERE CONTAINS(companyname, 'sap', FUZZY(0.8, 'stopwordTable=stopwords,
stopwordListId=01, textSearch=compare'))
ORDER BY score DESC, ID;
```

Table 120:

SCORE	ID	COMPANYNAME
0.95	1	SAP AG
0.95	2	SAP Aktiengesellschaft

Query 2: User input with stopword. Other stopwords in the database table.

```
SELECT TO_DECIMAL(SCORE(), 3, 2) AS score, *
FROM companies
WHERE CONTAINS (companyname, 'sap blub', FUZZY(0.8, 'stopwordTable=stopwords,
stopwordListId=01, textSearch=compare'))
ORDER BY score DESC, ID;
```

Table 121:

SCORE	ID	COMPANYNAME
0.95	1	SAP AG
0.95	2	SAP Aktiengesellschaft

Query 3: User input with stopword. One record with matching stopword, one record with differing stopword.

```
SELECT TO_DECIMAL(SCORE(), 3, 2) AS score, *
FROM companies
WHERE CONTAINS (companyname, 'sap aktiengesellschaft', FUZZY(0.8,
'stopwordTable=stopwords, stopwordListId=01, textSearch=compare'))
ORDER BY score DESC, ID;
```

Table 122:

SCORE	ID	COMPANYNAME
1	2	SAP Aktiengesellschaft
0.95	1	SAP AG

Query 4: User input with two stopwords. Database records with one matching stopword.

```
SELECT TO_DECIMAL(SCORE(), 3, 2) AS score, *
FROM companies
WHERE CONTAINS (companyname, 'sap ag aktiengesellschaft', FUZZY(0.8,
'stopwordTable=stopwords, stopwordListId=01, textSearch=compare'))
ORDER BY score DESC, ID;
```

Table 123:

SCORE	ID	COMPANYNAME
0.97	1	SAP AG
0.97	2	SAP Aktiengesellschaft

4.3.4.8.7.3 Stopwords Combined With Term Mappings

When stopwords and term mappings are combined in a single query, term mappings are applied first. Stopwords are then applied to all variations of the search term created by the term mappings.

Let us assume that you have defined the following term mapping:

Table 124:

Term 1	Term 2
Incredible Busy Machines	IBM
Ltd	Limited

Now you search for "Incredible Busy Machines Ltd".

The search would be carried out for all possible search terms:

Table 125:

Search Terms
Incredible Busy Machines Ltd
Incredible Busy Machines Limited
IBM Ltd
IBM Limited

Let us assume that you have defined the following stopwords:

Table 126:

Stopword
busy machines
ltd

The stopwords will not be searched, so the resulting search terms would be:

Table 127:

Search Terms
Incredible Busy Machines Ltd
Incredible Busy Machines Limited
IBM Ltd
IBM Limited

4.3.4.8.8 Term Mappings

Facts About Term Mappings

Term mappings have the following characteristics:

- Term mappings can be used to extend the search by adding additional search terms to the user input.
  When the user enters a search term, the search term is expanded, and synonyms, hypernyms, hyponyms, and so on are added. The result that is returned to the user contains additional records or documents related to the search term that could be useful to the user.
- Term mappings are defined in a column table and can be changed at any time.
  The current term mapping definition is applied when a search is started. The definition of term mappings does not change the data that is stored in the database tables (unlike the definition of synonyms in Text Analysis, where a change of synonyms requires the text data to be reloaded or reindexed).
- Term mappings can be grouped.
  Each group of term mappings is identified by a list_id that is stored in the term mapping table. By grouping term mappings, it is possible to apply different sets of term mappings to different columns of a table. You may want to use certain term mappings when searching company names for example, and other term mappings when searching documents. When starting a search, it is possible to specify which term mapping list_ids to apply to each column.
- Term mappings can be assigned a weight.
  In this case, records that are found because of a term mapping will receive a lower score than records found with the original user input. From the user's view, the sorting of the result list is more meaningful.
- Term mappings are defined as a unidirectional replacement.
  For a term mapping definition of 'term1' -> 'term2', 'term1' is replaced with 'term2', but 'term2' is not replaced with 'term1'. This is helpful if you want a search with a hypernym to find all hyponyms, but not the other way round. If a bidirectional replacement is needed (as for synonyms), both directions have to be added to the term mapping table.
- Term mappings are language-dependent.
  It is possible to define the language that a term mapping is valid for. You can also define term mappings for all languages by not setting a language.

Use Cases

Synonyms

If you have a large database of company names, you might want to map the legal forms of companies.

For example:

Table 128:

Searching for	You would also like to find	With a weight of
AG	Aktiengesellschaft	1.0
Ltd	Limited	1.0

Since these are synonyms, the term mappings have to be added to the term mapping table in both directions, as shown in the example below.

Usually, synonym definitions get a weight of 1.0, because records found when the term mapping is applied are as good as records found with the original user input.

Hypernyms, Hyponyms

If you search with a hypernym, you might also find other documents related to this topic.

For example:

Table 129:

Searching for	You would also like to find	With a weight of
car	VW Golf	0.8

Since these are not synonyms, and a search with 'VW Golf' should not return all documents about cars, the term mapping is added to the term mapping table in this direction only.

Format of the Term Mapping Table

Table 130:

Column Name	Type	Primary Key	Description	Comment
MAPPING_ID	VARCHAR(32)	x	Primary key	For example, a GUID
LIST_ID	VARCHAR(32)		Term mapping list ID	Used to group term mappings
LANGUAGE_CODE	VARCHAR(2)		Language code (ISO2)	NULL: term mapping is valid for all languages
TERM_1	NVARCHAR(200)		Term 1, the term to be replaced	
TERM_2	NVARCHAR(200)		Term 2, the term that replaces term 1	

Column Name	Type	Primary Key	Description	Comment
WEIGHT	DECIMAL, DECI-MAL(n,m)		Weight, 0.0 <= Weight <= 1.0	

The definition of the term mapping table is checked, so tables with other column names or data types cannot be used for a fuzzy search. Nevertheless, the table may contain additional columns that are ignored by the fuzzy search engine.

Example code for creating a term mapping table via SQL:

```
CREATE COLUMN TABLE termmappings
(
    mapping_id    VARCHAR(32)   PRIMARY KEY,
    list_id       VARCHAR(32)   NOT NULL,
    language_code VARCHAR(2),
    term_1        NVARCHAR(200) NOT NULL,
    term_2        NVARCHAR(200) NOT NULL,
    weight        DECIMAL       NOT NULL
);
```

i Note

You cannot create a DECIMAL column using the ABAP DDIC. Instead you have to use the SQL type DECIMAL(n,m) or the DDIC type DEC(n,m), for example DEC(3,2).

Support of Client-Dependent Term Mappings

Since ABAP applications support multiple clients, and each individual client should only see their own data, Fuzzy Search provides the option of using client-specific term mappings.

To limit term mappings to the current client in ABAP, you have to create an extra view on top of your custom term mapping table (including the client information) and limit the data to the current client with a WHERE condition (`WHERE CLNT = SESSION_CONTEXT('CLIENT')`).

Basic Example

The value given for the termMappingTable parameter can be any valid SQL identifier as defined in the SQL reference manual. If no schema is specified, the current schema is used.

```
CREATE COLUMN TABLE termmappings
(
    mapping_id    VARCHAR(32)    PRIMARY KEY,
    list_id       VARCHAR(32)    NOT NULL,
    language_code VARCHAR(2),
    term_1        NVARCHAR(200)  NOT NULL,
    term_2        NVARCHAR(200)  NOT NULL,
    weight        DECIMAL        NOT NULL
);
CREATE COLUMN TABLE companies
(
```

```
    id          INTEGER          PRIMARY KEY,
    companyname SHORTTEXT(200)   FUZZY SEARCH INDEX ON LANGUAGE COLUMN language
FAST PREPROCESS OFF,
    language    VARCHAR(2)
);
INSERT INTO companies VALUES ('1','SAP AG', 'de');
INSERT INTO companies VALUES ('2','SAP Aktiengesellschaft', 'de');
INSERT INTO termmappings VALUES ('1','01','de','AG','Aktiengesellschaft','0.9');
INSERT INTO termmappings VALUES ('2','01','de','Aktiengesellschaft','AG','0.9');
SELECT TO_DECIMAL(SCORE(),3,2) AS score, *
FROM companies
WHERE CONTAINS(companyname, 'sap aktiengesellschaft',
               FUZZY(0.8,
'termMappingTable=termmappings,termMappingListId=01,textSearch=compare'))
ORDER BY score DESC, id;
```

Table 131:

SCORE	ID	COMPANYNAME
1.0	2	SAP Aktiengesellschaft
0.9	1	SAP AG

```
SELECT TO_DECIMAL(SCORE(),3,2) AS score, * FROM companies
WHERE CONTAINS(companyname, 'sap ag',
FUZZY(0.8,'termMappingTable=TERMMAPPINGS,termMappingListId=01,textSearch=compare'
))
ORDER BY score DESC, id;
```

Table 132:

SCORE	ID	COMPANYNAME
1.0	1	SAP AG
0.9	2	SAP Aktiengesellschaft

To activate **language-specific term mappings**, you must provide the language parameter:

```
SELECT TO_DECIMAL(SCORE(),3,2) AS score, * FROM companies
WHERE CONTAINS(companyname, 'sap ag',
FUZZY(0.8,'termMappingTable=TERMMAPPINGS,termMappingListId=01,textSearch=compare'
), language('de'))
ORDER BY score DESC, id;
```

> **i Note**
>
> In this case, all term mappings where language_code is set to 'de' or is empty are used. Any term mappings with other language codes are ignored.

> **i Note**
>
> As a prerequisite the column to be searched in (here: companyname) has to have the `LANGUAGE` parameter set. This requires the parameter `FAST PREPROCESS` set to OFF for this column.

Using Multiple Term Mapping Lists

It is possible to use more than one term mapping list in a single call to CONTAINS(). To do this, specify the names of the term mapping lists as a comma-separated list enclosed in double quotes.

The following example shows how to specify one or more term mapping lists.

```
/* use one term mapping list */
SELECT TO_DECIMAL(SCORE(),3,2) AS score, * FROM companies
WHERE CONTAINS(companyname, 'sap ag',
FUZZY(0.8,'termMappingTable=TERMMAPPINGS,termMappingListId=01,textSearch=compare'
), language('de'))
ORDER BY score DESC, id;
/* or */
SELECT TO_DECIMAL(SCORE(),3,2) AS score, * FROM companies
WHERE CONTAINS(companyname, 'sap ag',
FUZZY(0.8,'termMappingTable=TERMMAPPINGS,termMappingListId="01",textSearch=compar
e'), language('de'))
ORDER BY score DESC, id;
/* use multiple term mapping lists */
SELECT TO_DECIMAL(SCORE(),3,2) AS score, * FROM companies
WHERE CONTAINS(companyname, 'sap ag',
FUZZY(0.8,'termMappingTable=TERMMAPPINGS,termMappingListId="01,02,03",textSearch=
compare'), language('de'))
ORDER BY score DESC, id;
/* use all term mapping lists */
SELECT TO_DECIMAL(SCORE(),3,2) AS score, * FROM companies
WHERE CONTAINS(companyname, 'sap ag',
FUZZY(0.8,'termMappingTable=TERMMAPPINGS,termMappingListId=*,textSearch=compare')
, language('de'))
ORDER BY score DESC, id;
```

Term Mappings and String Columns

It is also possible to use term mappings with string-type columns. In this case the term mapping replacement is **not** done on a token-by-token basis. Instead, the complete string is repaced if it matches the term mapping definition.

The following example shows how to use term mappings for string-type columns.

```
CREATE COLUMN TABLE termmappings
(
    mapping_id    VARCHAR(32)   PRIMARY KEY,
    list_id       VARCHAR(32)   NOT NULL,
    language_code VARCHAR(2),
    term_1        NVARCHAR(200) NOT NULL,
    term_2        NVARCHAR(200) NOT NULL,
    weight        DECIMAL       NOT NULL
);

INSERT INTO termmappings VALUES ('1', 'colors', NULL, 'gray', 'light gray', 0.9);
INSERT INTO termmappings VALUES ('2', 'colors', NULL, 'grey', 'gray', 0.9);

CREATE COLUMN TABLE products
(
  id INTEGER PRIMARY KEY,
  product NVARCHAR(100) NOT NULL,
  color  NVARCHAR(100)
);
```

```
INSERT INTO products VALUES (1, 'a product', 'gray');
INSERT INTO products VALUES (2, 'another product', 'light gray');
INSERT INTO products VALUES (3, 'new product', 'grey');
INSERT INTO products VALUES (4, 'latest product', 'light grey');

SELECT TO_DECIMAL(SCORE(),3,2), * FROM products WHERE CONTAINS(color, 'gray',
EXACT('termMappingTable=termmappings, termMappingListId=colors'));
SELECT TO_DECIMAL(SCORE(),3,2), * FROM products WHERE CONTAINS(color, 'light
grey', EXACT('termMappingTable=termmappings, termMappingListId=colors'));
```

The following tables show the search results:

Table 133:

SCORE	ID	PRODUCT	COLOR
1	2	a product	gray
0.9	1	another product	light gray

Table 134:

SCORE	ID	PRODUCT	COLOR
1	4	latest product	light grey

```
/* add fulltext index to see the differences between string-type column and text-
type column */
CREATE FULLTEXT INDEX products_color ON products(color) SYNC;
SELECT TO_DECIMAL(SCORE(),3,2), * FROM products WHERE CONTAINS(color, 'gray',
EXACT('termMappingTable=termmappings, termMappingListId=colors'));
SELECT TO_DECIMAL(SCORE(),3,2), * FROM products WHERE CONTAINS(color, 'light
grey', EXACT('termMappingTable=termmappings, termMappingListId=colors'));
```

Table 135:

SCORE	ID	PRODUCT	COLOR
1	1	a product	gray
0.9	2	another product	light gray

Table 136:

SCORE	ID	PRODUCT	COLOR
1.0	4	latest product	light grey
0.9	2	another product	light gray

4.3.4.8.9 Compound Words

Introduction

A search on a TEXT column is very sensitive to compound spelling. This means that a search for a compound word (written without blanks) might not find a document if there is an additional blank character between the

components. In some cases, compound words (for example in Finnic and Germanic languages) may be a combination of more than two words.

Examples of compound words:

- "Hans-Peter" vs. "Hanspeter"
- "Abfahrt Zeit" vs. "Abfahrtszeit"
- "Post Office" vs. "Postoffice"
- "soft ball" vs. "softball"
- "Example-Cola" vs. "ExampleCola"
- "hätäuloskäytävä" vs. "hätä ulos käytävä" (emergency exit in finnish)

To mitigate this, it is possible to generate compositions and decompositions for a query on the fly. Each (de)composition can be thought of as a term mapping (see Fuzzy Search - Term Mappings)

In this context, a **word** is any sequence of at least two characters.

The following options are offered:

- `composeWords`
  For more information, see Option composeWords [page 226].
- `decomposeWords`
  For more information, see Option decomposeWords [page 227].
- `compoundWordWeight`
  For more information, see Option compoundWordWeight [page 228].

Differences to Term Mappings

Generally, (de)compositions work like term mappings with a WEIGHT of `compoundWordWeight`. However, to limit the number of (mostly redundant) candidates, compound words and decomposition phrases are always searched with a fixed `similarCalculationMode` and `fuzziness` of the maximum of 0.7 and the value given by the user.

Stopwords

From a user perspective, it does not make any difference whether (de)composition terms are augmented or entered manually. (De)compositions therefore apply to stopwords as well as non-stopwords. Consequently, the following statement finds "SAPAG" regardless of whether "AG" is a stopword or not, and vice versa (with decomposeWords=2):

```
CONTAINS(COMPANY, 'SAP AG', FUZZY(0.8, 'composeWords=2'))
```

Performance Considerations

Setting **`decomposeWords>=3`**, a large number of search terms and very long words can have a negative impact on performance, due to the large number of resulting combinations. This can be mitigated by raising

minTextScore and lowering compoundWordWeight, as all combinations with an effective weight (product of all term mapping/compound word weights) less than minTextScore can be skipped during the search.

For more information, see Option minTextScore [page 210]

Setting **composeWords>=2** should not have a significant performance impact.

Let's have a look at impact factors in different data domains

> **Note**
>
> Factor: An impact factor of 2 means that your runtime might be doubled.

Table 137:

Data Domain	Table Rows	Dictionary Size	Request Baseline	composeWords Factor	decomposeWords Factor	Comment
lastnames	10 000 000	540 000	50ms	1	1.6	
full-text documents	130 000	1 200 000	70ms	2	6	IDF scoring
music tracks (short text documents)	6 400 000	1 250 000	70ms	5	7	textSearch=compare

Examples

Creating a table:

```
CREATE COLUMN TABLE SNACKS (TEXT TEXT FUZZY SEARCH INDEX ON);
INSERT INTO SNACKS VALUES('Mini-Maxi YadaThing Comp.');
```

```
SELECT TO_DECIMAL(SCORE(),3,2) AS SCORE, TEXT FROM SNACKS
 WHERE CONTAINS(TEXT, 'minimaxi yada thing', FUZZY(0.7, 'ts=compare'));
```

Table 138:

Score	Text

```
SELECT TO_DECIMAL(SCORE(),3,2) AS SCORE, TEXT FROM SNACKS
 WHERE CONTAINS(TEXT, 'mini-maxi yadathing', FUZZY(0.7, 'ts=compare'));
```

Table 139:

Score	Text
0.86	Mini-Maxi YadaThing Comp.

```
SELECT TO_DECIMAL(SCORE(),3,2) AS SCORE, TEXT FROM SNACKS
 WHERE CONTAINS(TEXT, 'minimaxi yada thing', FUZZY(0.7,
'ts=compare,composeWords=2,decomposeWords=2'));
```

Table 140:

Score	Text
0.7	Mini-Maxi YadaThing Comp.

```
SELECT TO_DECIMAL(SCORE(),3,2) AS SCORE, TEXT FROM SNACKS
 WHERE CONTAINS(TEXT, 'minimaxi yada thing', FUZZY(0.7,
'ts=compare,composeWords=2,decomposeWords=2,compoundWordWeight=1.0'));
```

Table 141:

Score	Text
0.86	Mini-Maxi YadaThing Comp.

```
SELECT TO_DECIMAL(SCORE(),3,2) AS SCORE, TEXT FROM SNACKS
 WHERE CONTAINS(TEXT, 'minimaxi yada thing', FUZZY(0.7,
'ts=compare,composeWords=2,decomposeWords=2,compoundWordWeight=0.8'));
```

Table 142:

Score	Text
0.55	Mini-Maxi YadaThing Comp.

4.3.4.8.9.1 Option composeWords

The option `composeWords` defines how words in the user input are combined into compound words.

Table 143:

Option	Range	Default	Description
`composeWords`	1..5	1 (off)	Each consecutive series of at most composeWords words in the user input is concatenated into a compound word.

Examples

Table 144:

User Input	composeWords Value	Generated Compound Words
gene rat ion	=> 2	generat ion
		gen eration
	=> 3	generat ion
		gen eration
		generation

4.3.4.8.9.2 Option decomposeWords

The option `decomposeWords` defines how words in the user input are split into separate words, building a decomposition phrase.

Table 145:

Option	Range	Default	Description
`decomposeWords`	1..5	1 (off)	Each word in the user input is split into at most `decomposeWords` words .

Examples

Table 146:

User Input	decomposeWords Value	Generated Compound Words
generation	=> 2	ge neration
		gen eration
		gene ration
		gener ation
		genera tion
		generat ion
	=> 3	all of the above and
		ge ne ration
		ge ner ation
		ge nera tion
		ge nerat ion
		ge nerati on
		gen er ation
		gen era tion
		gen erat ion
		gen erati on
		gene ra tion
		gene rat ion
		gene rati on
		gener at ion
		gener ati on
		genera ti on

4.3.4.8.9.3 Option compoundWordWeight

The option `compoundWordWeight` defines how compound word hits affect the score of a document.

Table 147:

Option	Range	Default	Description
`compoundWordWeight`	0.0..1.0	0.9	Compound mapping weight.

For each applied (de)composition, the score of a document is multiplied by `compoundWordWeight`.

Examples

Table 148:

User Input	compoundWord-Weight	composeWords	decomposeWords	Document	Score Multiplier
minimaxi yada yada company	0.8	2	2	mini-maxi ya-dayada company	0.8*0.8 = 0.64
minimaxi yada yada company	0.8	2	1	mini-maxi ya-dayada company	0.8 (if found)

4.3.4.8.10 Speeding Up the Fuzzy Search on Text Columns

Context

You can speed up the fuzzy search for all SQL types except DATE by creating a special data structure known as a fuzzy search index. The additional data structures increase the total memory footprint of the loaded table, in some cases even doubling it.

TEXT Columns

TEXT columns offer the 'FUZZY SEARCH INDEX' option to enable and disable the indexes when creating a table:

```
CREATE COLUMN TABLE mytable
(
  id    INTEGER        PRIMARY KEY,
  col1  TEXT           FUZZY SEARCH INDEX ON
);
```

You can change this later using the ALTER TABLE command.

```
ALTER TABLE mytable ALTER
(
  col1  TEXT           FUZZY SEARCH INDEX OFF
);
```

The SYS.TABLE_COLUMNS view shows the current state of the fuzzy search indexes. When working with attribute views, this information is also visible in SYS.VIEW_COLUMNS.

SHORTTEXT Columns

SHORTTEXT columns offer the 'FUZZY SEARCH INDEX' option to enable and disable the indexes when creating a table:

```
CREATE COLUMN TABLE mytable
(
  id     INTEGER         PRIMARY KEY,
  col2   SHORTTEXT(100) FUZZY SEARCH INDEX ON
);
```

You can change this later using the ALTER TABLE command.

```
ALTER TABLE mytable ALTER
(
  col2   SHORTTEXT(100) FUZZY SEARCH INDEX OFF
);
```

i Note

The following limitation applies: Since SHORTTEXT creates a hidden column of type Text, the state of the fuzzy search index is not visible in SYS.TABLE_COLUMNS and SYS.VIEW_COLUMNS.

SYS.FULLTEXT_INDEXES can be used to query the state of the fuzzy search indexes.

```
SELECT index_name, internal_column_name, fuzzy_search_index
FROM fulltext_indexes
WHERE table_name = 'MYTABLE';
```

Full-Text Indexes

A full-text index offers the 'FUZZY SEARCH INDEX' option to enable and disable the indexes when creating a full-text index:

```
CREATE COLUMN TABLE mytable
(
  col3 NVARCHAR(2000)
);
CREATE FULLTEXT INDEX myindex ON mytable(col3)
  FUZZY SEARCH INDEX ON;
```

You can change this later using the ALTER FULLTEXT INDEX command:

```
ALTER FULLTEXT INDEX myindex FUZZY SEARCH INDEX OFF;
```

The SYS.FULLTEXT_INDEXES view shows the current state of the fuzzy search indexes.

```
SELECT index_name, internal_column_name, fuzzy_search_index
FROM fulltext_indexes
WHERE table_name = 'MYTABLE';
```

4.3.4.9 Fuzzy Search on DATE Columns

This topic describes the fuzzy search features that are available for date columns (SQL type `DATE`).

Fuzzy search on `DATE` columns supports two different types of searches:

- The search can be based on the fuzzy similarity between dates. This means that search handles date-specific typographical errors and finds dates lying within a user-defined maximum distance.
- Alternatively, the search can be based on a linear or gaussian score function that calculates the score depending on the distance between the dates.

Related Information

DATE Columns with Fuzzy Similarity [page 230]
DATE Columns with Score Functions [page 233]

4.3.4.9.1 DATE Columns with Fuzzy Similarity

Fuzzy search on `DATE` columns supports two types of errors: date-specific typographical errors and dates lying within a user-defined maximum distance.

It is not possible to create additional data structures for `DATE` columns to speed up the search. The performance of queries is perfectly satisfactory without the need for database tuning.

> **Note**
>
> Fuzzy search for dates works with valid dates only. A search with an invalid date does not return any results.
>
> The following example does not return any results, as '31' is not a valid month:
>
> ```
> ...WHERE CONTAINS(mydate, '2012-31-01', FUZZY(0.8))...
> ```

Score Calculation for Typographical Errors

Instead of using Levenshtein distance or other string comparison algorithms, the following date-specific typographical errors and errors are defined as similar:

1. One wrong digit at any position (for example, 2011-08-15 instead of 2011-08-25). This type of error gets a score of 0.90.
2. Two digits interchanged within one component (day, month, or year) (for example, 2001-01-12, 2010-10-12, or 2010-01-21 instead of 2010-01-12). This type of error gets a score of 0.85.
3. Month and day interchanged (US versus European date format) (for example, 2010-10-12 instead of 2010-12-10). This type of error gets a score of 0.80.

Only one of these errors is allowed. Dates with more than one error are not considered similar, so the score is 0.0.

Dates with a score less than the `fuzzySimilarity` parameter are not returned.

Example:

```
SELECT TO_DECIMAL(SCORE(),3,2) AS score, * FROM dates
WHERE CONTAINS(dateofbirth, '2000-01-10', FUZZY(0.8))
ORDER BY score DESC;
```

For this example we get:

2000-01-09 -> 0.00 (not returned to the user)

2000-01-10 -> 1.00

2000-01-11 -> 0.90

2000-01-12 -> 0.90

...

2000-01-21 -> 0.00 (not returned to the user)

...

2000-10-01 -> 0.80

Score Calculation for Date Distance

The maximum allowed distance between dates can be defined using the search option `maxDateDistance`, which defines a number of days.

The default for this option is 0, meaning that the feature is disabled. This is shown in the following example:

```
SELECT TO_DECIMAL(SCORE(),3,2) AS score, * FROM dates
WHERE CONTAINS(dateofbirth, '2000-01-10', FUZZY(0.95, 'maxDateDistance=5'))
ORDER BY score DESC;
```

This query returns all dates between 2000-01-05 and 2000-01-15.

The fuzzy score for dates is calculated as follows:

1. The identical date gets a score of 1.0.
2. The date that is `maxDateDistance` days away from the search input gets a score that equals the `fuzzySimilarity` parameter (0.95 in the example above).
3. The score of dates between the identical date and `maxDateDistance` is calculated as a linear function between the two dates defined above. In other words, for each day, the score is reduced by ((1-`fuzzySimilarity`) / `maxDateDistance`).
4. For dates outside the range of maxDateDistance, the score is 0.0.

With the example above, we therefore get:

2000-01-04 -> 0.00

2000-01-05 -> 0.95

2000-01-06 -> 0.96

...

2000-01-09 -> 0.99

2000-01-10 -> 1.0

2000-01-11 -> 0.99

2000-01-12 -> 0.98

...

2000-01-15 -> 0.95

2000-01-16 -> 0.00

The distance between dates is calculated following the rules of the Gregorian calendar.

The special case `fuzzySimilarity = 1.0` and `maxDateDistance=n` is allowed and returns all dates within a range of n days with a rank of 1.0.

Dates That Meet Both Conditions

If a date meets the conditions of a typo and the conditions of the `maxDateDistance` parameter, two scores are calculated for the same date. In this case, the `score()` function returns the highest of both scores. This is shown in the following example:

```
SELECT TO_DECIMAL(SCORE(),3,2) AS score, * FROM dates
WHERE CONTAINS(dateofbirth, '2000-01-10', FUZZY(0.8, 'maxDateDistance=5'))
ORDER BY score DESC;
```

This query returns the following:

2000-01-04 -> 0.00

2000-01-05 -> 0.80

2000-01-06 -> 0.84

2000-01-07 -> 0.88

2000-01-08 -> 0.92

2000-01-09 -> 0.96

2000-01-10 -> 1.0

2000-01-11 -> 0.96

2000-01-12 -> 0.92

2000-01-13 -> 0.90

2000-01-14 -> 0.90

2000-01-15 -> 0.90

2000-01-16 -> 0.90

4.3.4.9.2 DATE Columns with Score Functions

As an alternative to the fuzzy search for dates, score functions are available that calculate the score based on the distance between search term and dates stored in the column.

SQL types for dates (`DATE`, `SECONDDATE`, and `TIMESTAMP`) support score functions that return a score based on the difference between the search term and the value stored in the date column. Functions for linear and gaussian scores are available.

Linear Score Function

The linear score function for dates is similar to the linear score function for numeric columns.

The only difference is that for dates or datetime types the distance between two values is calculated as a number of days. This means that two dates that are two weeks apart have a distance of 14. Two timestamps that are 12 hours apart have a distance of 0.5.

The scale and offset parameters of the function are also interpreted as a number of days. To specify, for example, a scale of four weeks, `scoreFunctionScale=28` has to be given. For a scale of six hours, `scoreFunctionScale=0.25` has to be defined.

Gaussian Score Function

The gaussian score function for dates is also similar to the gaussian score function for numeric columns.

As for linear score functions, the only difference is that all distance, scale, and offset values are interpreted as a number of days.

Search Options for Score Functions

The following table shows the search options that are available for the score functions described above.

Table 149:

Search Option	Possible Values	Default Value	Description	Linear	Gaussian
`scoreFunction`	linear, gaussian	n/a	Name of the score function	yes	yes

Search Option	Possible Values	Default Value	Description	Linear	Gaussian
`scoreFunctionScale`	scale > 0	n/a	Value of the score function parameter '`scale`', given in days	yes	yes
`scoreFunctionDecay`	0 <= decay < 1 (linear), 0 < decay < 1 (gaussian)	0.5	Value of the score function parameter '`decay`'	yes	yes
`scoreFunctionOffset`	offset >= 0	0	Value of the score function parameter '`offset`', given in days	yes	yes

Example

Sample Code

Linear and Gaussian Score Functions for Dates

```
CREATE COLUMN TABLE datvalues
(
  id INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
  val DATE
);

INSERT INTO datvalues (val) VALUES ('2017-04-12');
INSERT INTO datvalues (val) VALUES ('2017-01-01');
INSERT INTO datvalues (val) VALUES ('2016-04-12');
INSERT INTO datvalues (val) VALUES ('2016-01-01');
INSERT INTO datvalues (val) VALUES ('2015-04-12');
INSERT INTO datvalues (val) VALUES ('2015-01-01');
INSERT INTO datvalues (val) VALUES ('2018-04-12');
INSERT INTO datvalues (val) VALUES ('2000-01-01');

/* linear score */
SELECT SCORE(), val FROM datvalues WHERE CONTAINS(val, '2017-04-12',
FUZZY(0.4, 'scoreFunction=linear,scoreFunctionScale=365')) ORDER BY val;
SELECT SCORE(), val FROM datvalues WHERE CONTAINS(val, '2017-04-12',
FUZZY(0.4,
'scoreFunction=linear,scoreFunctionScale=365,scoreFunctionDecay=0.75')) ORDER
BY val;
SELECT SCORE(), val FROM datvalues WHERE CONTAINS(val, '2017-04-12',
FUZZY(0.4,
'scoreFunction=linear,scoreFunctionScale=365,scoreFunctionDecay=0.75,scoreFunct
ionOffset=182')) ORDER BY val;
SELECT SCORE(), val FROM datvalues WHERE CONTAINS(val, '2017-04-12',
FUZZY(0.4, 'sf=linear,sfs=365,sfd=0.75,sfo=182')) ORDER BY val;

/* gaussian score */
```

```
SELECT SCORE(), val FROM datvalues WHERE CONTAINS(val, '2017-04-12',
FUZZY(0.4, 'scoreFunction=gaussian,scoreFunctionScale=365')) ORDER BY val;
SELECT SCORE(), val FROM datvalues WHERE CONTAINS(val, '2017-04-12',
FUZZY(0.4,
'scoreFunction=gaussian,scoreFunctionScale=365,scoreFunctionDecay=0.75'))
ORDER BY val;
SELECT SCORE(), val FROM datvalues WHERE CONTAINS(val, '2017-04-12',
FUZZY(0.4,
'scoreFunction=gaussian,scoreFunctionScale=365,scoreFunctionDecay=0.75,scoreFun
ctionOffset=182')) ORDER BY val;
SELECT SCORE(), val FROM datvalues WHERE CONTAINS(val, '2017-04-12',
FUZZY(0.4, 'sf=gaussian,sfs=365,sfd=0.75,sfo=182')) ORDER BY val;

DROP TABLE datvalues;
```

With the search term '2017-04-12', the example above returns the following scores.

Table 150:

Value	linear, scale=365	linear, scale=365, decay=0.75	linear, scale=365, decay=0.75, offset=182	gaussian, scale=365	gaussian, scale=365, decay=0.75	gaussian, scale=365, decay=0.75, offset=182
2000-01-01						
2015-01-01		0.43	0.555			0.402
2015-04-12		0.5	0.624			0.522
2016-01-01		0.68	0.805		0.624	0.839
2016-04-12	0.5	0.75	0.875	0.5	0.75	0.93
2017-01-01	0.862	0.931	1	0.948	0.978	1
2017-04-12	1	1	1	1	1	1
2018-04-12	0.5	0.75	0.875	0.5	0.75	0.93

4.3.4.10 Fuzzy Search on Numeric Columns

For numeric columns, several score functions are available that calculate the score based on the distance between the search term and numbers stored in the column.

Without any further search options, a fuzzy search on numeric columns normally calculates scores based on the edit distance (Levenshtein distance) between the string representation of the search term and the string representations of the numbers stored in the column.

Score Functions

Numeric SQL types (`TINYINT`, `SMALLINT`, `INTEGER`, `BIGINT`, `DECIMAL`, `SMALLDECIMAL`, `REAL`, `DOUBLE`, and `FLOAT`) support score functions that return a score based on the difference between the numerical value of the search term and the value stored in the numeric column.

You can use functions for linear, Gaussian, and logarithmic scores.

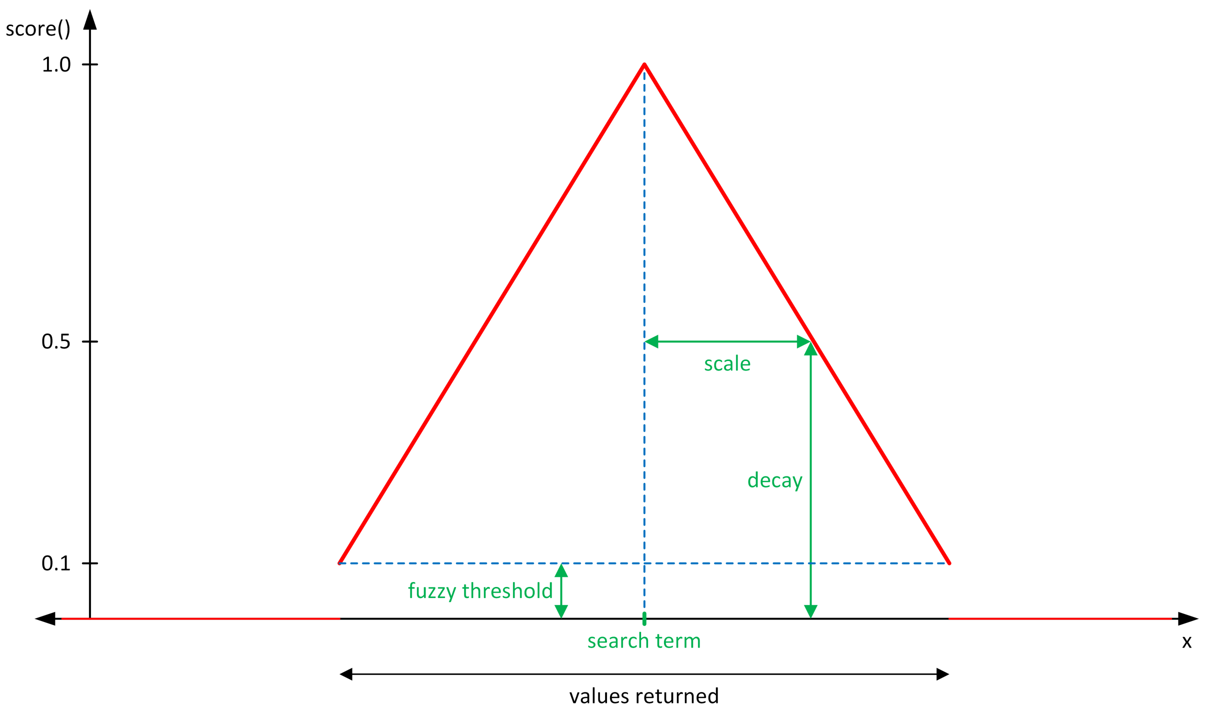
Linear Score Function

The linear score function has its highest point with `SCORE()=1` at the given search term. The parameters `scale` and `decay` define a second point of the score function and the slope of the score function is defined as:

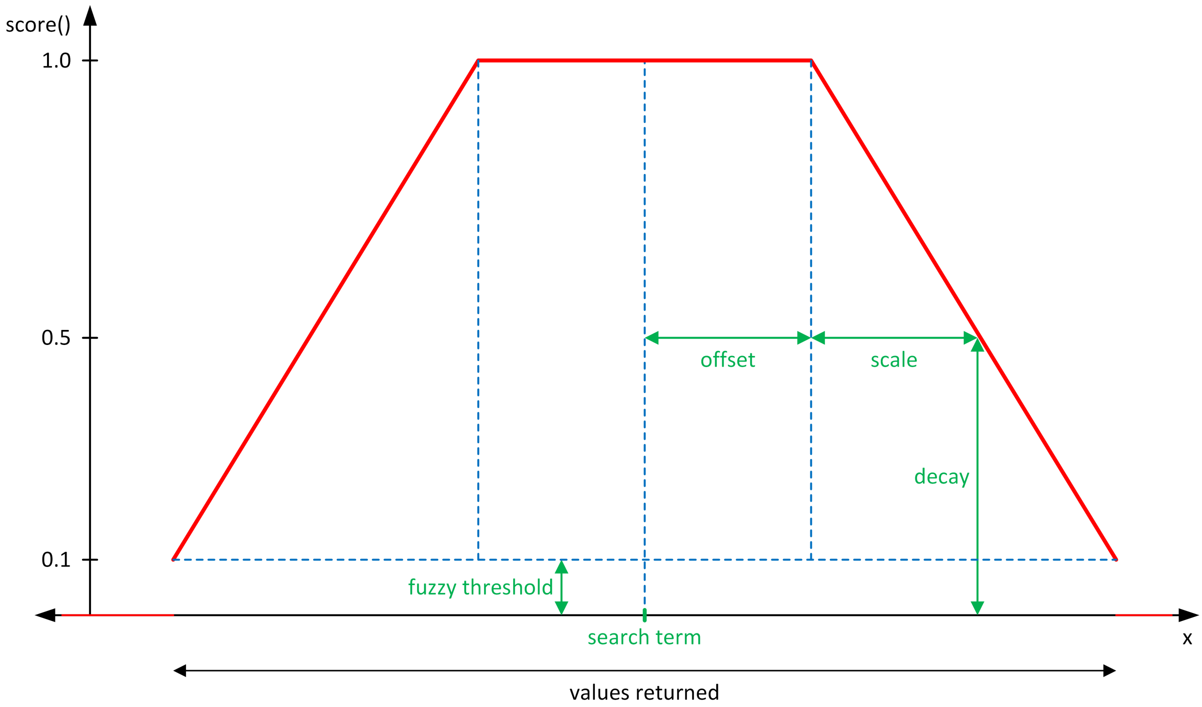
```
slope = (1 - decay) / scale
```

The fuzzy threshold given in the `CONTAINS()` predicate, defines the range of values that is returned for a search. All values with a score lower than the fuzzy threshold are not returned.

The following image shows the linear score function:



The optional parameter `offset` defines a range of values that gets a score of 1. All values in the range between (`search_term - offset`) and (`search_term + offset`) get a score of 1 and the linear score function is shifted left and right by the given offset. This is shown in the image below:



For a set of parameters `offset`, `scale`, `decay`, `fuzzy_threshold` and `search_term`, the linear score of a value `x` stored in the column is calculated as follows:

```
slope = (1 - decay) / scale
```

```
h(x) = max(0, abs(x - search_term) - offset)
```

```
linear_score(x) = max(1 - slope * h(x), 0)
```

The score function returns:

- a score based on the linear score function, if the linear score is greater than or equal to the given fuzzy threshold;
  `score(x) = linear_score(x)`
- a score of 0, if the linear score function is lesser than the given fuzzy threshold;
  `score(x) = 0.`
  A score of 0 means that the value is not included in the search result.

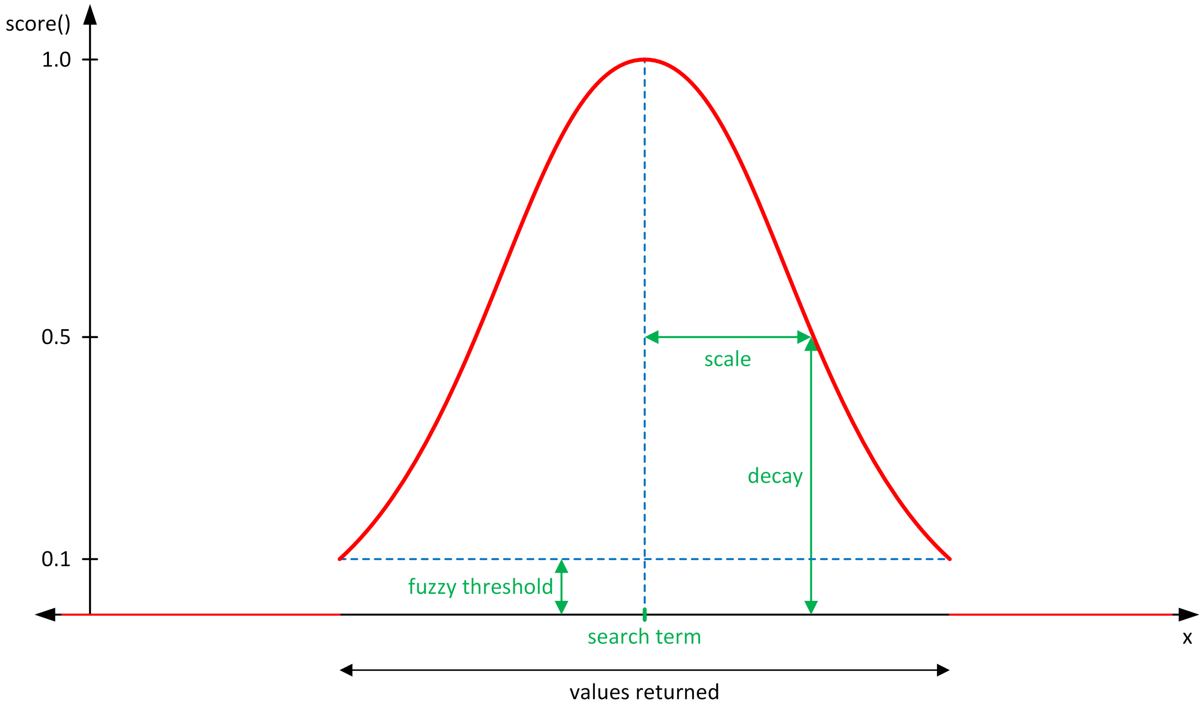
Gaussian Score Function

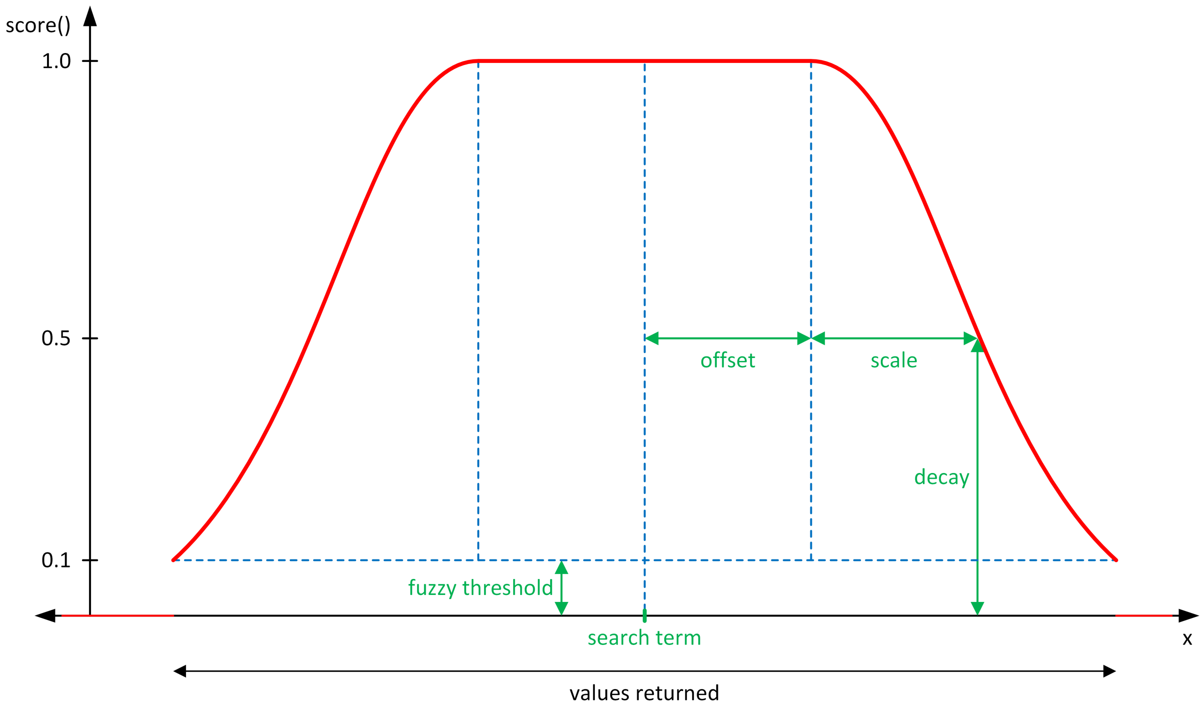
The Gaussian score function has its highest point with `SCORE()=1` at the given search term. The parameters `scale` and `decay` define a second point of the score function and the variance of the score function is defined as:

```
variance = - scale^2 / (2 * log(decay))
```

The fuzzy threshold given in the `CONTAINS()` predicate, defines the range of values that is returned for a search. All values with a score lower than the fuzzy threshold are not returned.

The following images show the Gaussian score function with and without the optional parameter `offset`.





For a set of parameters `offset`, `scale`, `decay`, `fuzzy_threshold` and `search_term`, the Gaussian score of a value `x` stored in the column is calculated as follows:

```
variance = - scale^2 / (2 * log(decay))
```

```
h(x) = max(0, abs(x - search_term) - offset)
```

```
gaussian_score(x) = exp( -h(x)^2 / (2 * variance))
```

The score function returns:

- a score based on the Gaussian score function, if the Gaussian score is greater than or equal to the given fuzzy threshold;
  `score(x) = gaussian_score(x)`
- a score of 0, if the Gaussian score function is lesser than the given fuzzy threshold;
  `score(x) = 0.`
  A score of 0 means that the value is not included in the search result.

Logarithmic Score Function

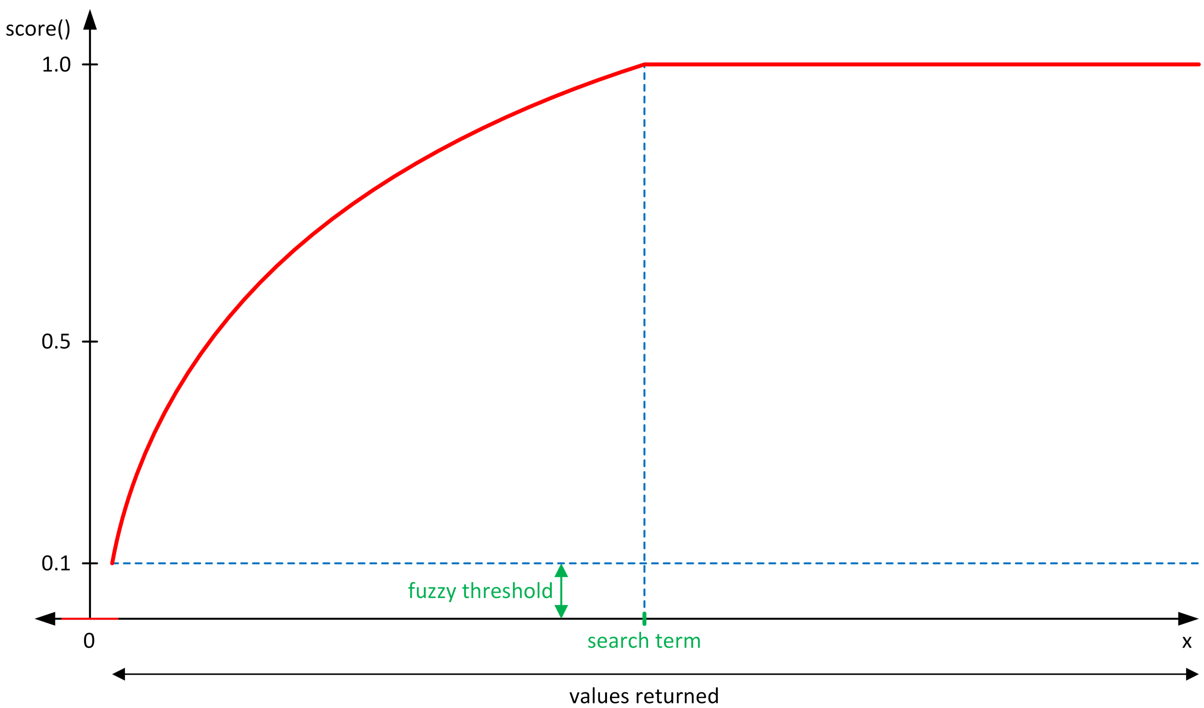
The logarithmic score function calculates scores using a logarithm function with a base that is given as a parameter or, if the parameter is not given, that is defined by the given search term:

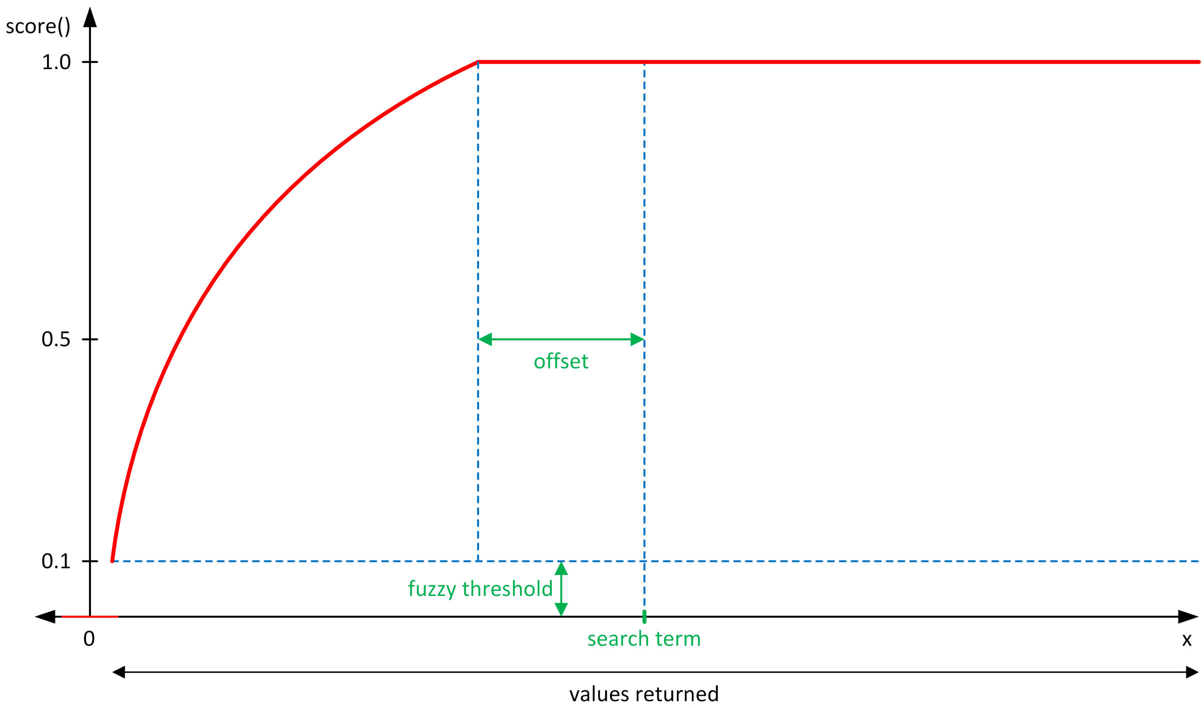
```
base = 1 + search_term
```

or, if the optional parameter `offset` is given;

```
base = (1 + Search_term - offset).
```

The images below show the logarithmic score function without and with optional `offset` parameter:





For a set of parameters offset, base, fuzzy_threshold and search_term, the logarithmic score of a value x stored in the column is calculated as follows:

`base = (1 + Search_term - offset)` (only if parameter `base` is not given)

`factor = (base - 1) / (search_term - offset)`

`logarithmic_score(x) = log(base, 1 + factor * x)`

The score function returns:

- a score based on the logarithmic score function, if the logarithmic score is greater than or equal to the given fuzzy threshold and if `x` is lesser than or equal to the given search term minus the given offset;
  `score(x) = logarithmic_score(x)`
- a score of 1, if x is greater than the given search term minus the given offset;
  `score(x) = 1`
- a score of 0, if x is lesser than 0 or the logarithmic score function is lesser than the given fuzzy threshold;
  `score(x) = 0.`
  A score of 0 means that the value is not included in the search result.

> **Note**
>
> When using a logarithmic score function, the search term has to be greater than 0. If the offset is given, (`search_term - offset`) also has to be greater than 0.

Search Options for Score Functions

The following table shows the search options that are available for the score functions.

Table 151:

Search Option	Possible Values	Default Value	Description	Linear	Gaussian	Logarithmic
`scoreFunction`	linear, Gaussian, logarithmic	n/a	Name of the score function	yes	yes	yes
`scoreFunctionScale`	scale > 0	n/a	Value of the score function parameter `scale`	yes	yes	no
`scoreFunctionDecay`	0 <= decay < 1 (linear), 0 < decay < 1 (Gaussian)	0.5	Value of the score function parameter `decay`	yes	yes	no
`scoreFunctionBase`	base > 1	1 + `(searchterm - offset)`	Value of the score function parameter `base`	no	no	yes
`scoreFunctionOffset`	offset >= 0	0	Value of the score function parameter `offset`	yes	yes	yes

Examples

Sample Code

Linear and Gaussian Score Functions

```
CREATE COLUMN TABLE numvalues
(
  id INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
  val INTEGER
);

INSERT INTO numvalues (val) VALUES (400);
INSERT INTO numvalues (val) VALUES (500);
INSERT INTO numvalues (val) VALUES (510);
INSERT INTO numvalues (val) VALUES (550);
INSERT INTO numvalues (val) VALUES (600);
INSERT INTO numvalues (val) VALUES (601);
INSERT INTO numvalues (val) VALUES (650);
INSERT INTO numvalues (val) VALUES (651);

/* linear score */
SELECT SCORE(), val FROM numvalues WHERE CONTAINS(val, '500', FUZZY(0.1,
'scoreFunction=linear,scoreFunctionScale=50')) ORDER BY val;
SELECT SCORE(), val FROM numvalues WHERE CONTAINS(val, '500', FUZZY(0.1,
'scoreFunction=linear,scoreFunctionScale=50,scoreFunctionDecay=0.6')) ORDER BY
val;
```

```
SELECT SCORE(), val FROM numvalues WHERE CONTAINS(val, '500', FUZZY(0.1,
'scoreFunction=linear,scoreFunctionScale=50,scoreFunctionDecay=0.6,scoreFunctio
nOffset=50')) ORDER BY val;
SELECT SCORE(), val FROM numvalues WHERE CONTAINS(val, '500', FUZZY(0.1,
'sf=linear,sfs=50,sfd=0.6,sfo=50')) ORDER BY val;

/* gaussian score */
SELECT SCORE(), val FROM numvalues WHERE CONTAINS(val, '500', FUZZY(0.1,
'scoreFunction=gaussian,scoreFunctionScale=50')) ORDER BY val;
SELECT SCORE(), val FROM numvalues WHERE CONTAINS(val, '500', FUZZY(0.1,
'scoreFunction=gaussian,scoreFunctionScale=50,scoreFunctionDecay=0.6')) ORDER
BY val;
SELECT SCORE(), val FROM numvalues WHERE CONTAINS(val, '500', FUZZY(0.1,
'scoreFunction=gaussian,scoreFunctionScale=50,scoreFunctionDecay=0.6,scoreFunct
ionOffset=50')) ORDER BY val;
SELECT SCORE(), val FROM numvalues WHERE CONTAINS(val, '500', FUZZY(0.1,
'sf=gaussian,sfs=50,sfd=0.6,sfo=50')) ORDER BY val;

DROP TABLE numvalues;
```

With the search term '500', the example above returns the following scores:

Table 152:

Value	linear, scale=50	linear, scale=50, decay=0.6	linear, scale=50, decay=0.6, offset=50	Gaussian, scale=50	Gaussian, scale=50, decay=0.6	Gaussian, scale=50, decay=0.6, offset=50
400		0.2	0.6		0.13	0.6
500	1	1	1	1	1	1
510	0.9	0.92	1	0.973	0.98	1
550	0.5	0.6	1	0.5	0.6	1
600		0.2	0.6		0.13	0.6
601		0.192	0.592		0.124	0.59
650			0.2			0.13
651			0.192			0.124

The following example shows the logarithmic score function, as it could possibly be used for a column containing a click counter.

Sample Code

Logarithmic Score Function

```
CREATE COLUMN TABLE numvalues
(
  id INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
  val INTEGER
);
```

```
INSERT INTO numvalues (val) VALUES (0);
INSERT INTO numvalues (val) VALUES (1);
INSERT INTO numvalues (val) VALUES (5);
INSERT INTO numvalues (val) VALUES (10);
INSERT INTO numvalues (val) VALUES (15);
INSERT INTO numvalues (val) VALUES (100);
INSERT INTO numvalues (val) VALUES (105);
INSERT INTO numvalues (val) VALUES (150);
INSERT INTO numvalues (val) VALUES (1000);
INSERT INTO numvalues (val) VALUES (1005);
INSERT INTO numvalues (val) VALUES (10000);
INSERT INTO numvalues (val) VALUES (100000);
INSERT INTO numvalues (val) VALUES (200000);

SELECT SCORE(), val FROM numvalues WHERE CONTAINS(val, '10000', FUZZY(0.05,
'scoreFunction=logarithmic')) ORDER BY val;

DROP TABLE numvalues;
```

With the search term '10000', the example above returns the following scores:

Table 153:

Value	logarithmic
1	0.075
5	0.195
10	0.26
15	0.301
100	0.501
105	0.506
150	0.545
1000	0.75
1005	0.751
10000	1
100000	1
200000	1

4.3.4.11 Language Specifics

This section includes information about language-specific behaviour of fuzzy search.

4.3.4.11.1 Limitations of Fuzzy Search when Using 'Non-Whitespace' Languages

Some languages do not use a space character to separate words. Examples are East-Asian languages like Chinese and Japanese. When using fuzzy search with these languages, there are limitations that are described below.

Limitations of NVARCHAR Columns

- Substring Search
  Substring search uses word boundaries to search for results and to calculate the scores. Since there are no space characters between words in a 'non-whitespace' language, substring search cannot locate the word boundaries.
  As a work-around, users can insert space characters between words when they type in the search term. Nevertheless, scores will not be as accurate as with 'whitespace' languages like English or French, because the word boundaries in the database entries are not known.

Limitations of Text Columns and Full-Text Indexes

- Fast Preprocessing and Language Detection
  Since there are no space characters between words, language specific dictionaries are used to tokenize text. As a consequence, a language column has to be defined or automatic language detection has to be enabled. It is not possible to use fast preprocessing for text columns or full-text indexes.
- Spelling Errors Change Number of Tokens
  The tokenizer uses a dictionary to split a text into single tokens. Spelling errors in the text lead to other matching dictionary entries, and as a result the length and number of tokens may be different to the same text without spelling errors. In some cases, no matching dictionary entries are found, so parts of the text may be split into very short tokens or even single characters (for example, with Chinese text).
  Text search and fuzzy search search the user input, token by token. If the number of tokens is different because of spelling errors, fuzzy search does not find the results expected by the user. The user gets the expected results only if the spelling errors do not change the output of the tokenizer.

  > Example
  >
  > Let's use two names of provinces of the People's Republic of China: 黑龙江 in Pinyin: Heilongjiang 浙江. In Pinyin: Zhejiang. If we concatenate the two names, we get '黑龙江浙江'. The tokenizer is able to split this string correctly into two tokens: 黑龙江 and 浙江. Imagine a misspelling in Heilongjiang and we type '嘿笼奖' which has the same Pinyin representation. If we concatenate the misspelled first province name and the original name of the second province, we get '嘿笼奖浙江', And the tokenizer returns four tokens: 嘿, 笼, 奖, and 浙江. The difference in the tokenization makes it impossible for fuzzy search to find the entry containing the correctly spelled name.

- Defining Term Mappings and Stopwords
  The language specific dictionaries are not used to tokenize the terms that are stored in a term-mapping table or in a stopword table. You therefore have to add space characters between the tokens of a 'non-

whitespace' language when adding data to the `term_1` and `term_2` columns of a term mapping table or to the `term` column of a stopword table. Otherwise the term mappings and stopwords will not be applied.

Related Information

Example for the Limitation on Defining Term Mappings and Stopwords [page 245]

4.3.4.11.1.1 Example for the Limitation on Defining Term Mappings and Stopwords

First we create a table and insert two records. Both records refer to the same company name 'Sample' - as we do not matter about the extension 'limited partnership'.

```
                CREATE COLUMN TABLE chinesetext(
                id INTEGER PRIMARY KEY,
                text_zh TEXT FAST PREPROCESS OFF LANGUAGE DETECTION ('ZH','EN'),
                text_en TEXT);
                INSERT INTO chinesetext VALUES (101, '样品有限合伙', 'Sample
limited partnership');
                INSERT INTO chinesetext VALUES (102, '样品', 'Sample');
```

The following SELECT statement retrieves only the record with id 102.

```
                SELECT SCORE(), * FROM chinesetext
                WHERE CONTAINS(text_zh, '样品', FUZZY(0.8, 'textSearch=compare,
minTextScore=0.6'));
```

Therefore we create a stopword table with an entry for 'limited partnership'.

```
            CREATE COLUMN TABLE mystopwords
            (
            stopword_id    VARCHAR(32)    PRIMARY KEY,
            list_id        VARCHAR(32)    NOT NULL,
            language_code  VARCHAR(2),
            term           NVARCHAR(200)  NOT NULL
            );
            INSERT INTO mystopwords VALUES (1, 'list1', '', '有限合伙'');
```

The following SELECT statement still retrieves only the record with id 102.

```
            SELECT SCORE(), * FROM chinesetext
            WHERE CONTAINS(text_zh, '样品'', FUZZY(0.8, 'textSearch=compare,
minTextScore=0.6, stopWordTable=mystopwords, stopWordListID=list1'));
```

We need to INSERT another entry into the stopword table having a blank between 'limited' and 'partnership'.

```
INSERT INTO mystopwords VALUES (2, 'list2', '', '有限 合伙'');
```

The following SELECT statement retrieves both records.

```
            SELECT SCORE(), * FROM chinesetext
            WHERE CONTAINS(text_zh, '样品'', FUZZY(0.8, 'textSearch=compare,
minTextScore=0.6, stopWordTable=mystopwords, stopWordListID=list2'));
```

4.3.4.11.2 Fuzzy Search - Chinese

This chapter describes the features of fuzzy search which are specific to Chinese script, meaning all strings or texts written with Chinese characters.

When comparing strings containing Chinese characters, fuzzy search does not just compare the Chinese characters. Fuzzy search also takes account of the Pinyin representation of the Chinese characters. Thus fuzzy search does not just return hits based on the identical Chinese characters. It also returns hits which sound similar and have an identical or similar Pinyin representation. This allows fuzzy search to deal with errors based on the most commonly used input method for Chinese characters: to type in the Pinyin representation and to pick one of the suggestions.

When a user enters a Chinese word, s/he might enter a wrong pinyin transcription or select a wrong Chinese word from the proposed choices. The following table shows an example of Chinese words that are considered to be similar by fuzzy search:

Table 154:

Chinese Term	Pinyin Representation
北京	beijing
背景	beijing
被禁	beijin

4.3.4.11.2.1 Fuzzy Search on Chinese Strings

Fuzzy search on NVARCHAR columns compares the complete content of the search input against the complete string of the database field. This includes all characters including spaces and punctuation. At this point, there is no difference to fuzzy search on non-Chinese scripts.

> **Note**
>
> Including the Pinyin representation for Chinese strings is active as soon as there are Chinese characters either in the search input or in a database entry.

4.3.4.11.2.2 Fuzzy Search on Chinese Texts

> **Note**
>
> The Chinese text search feature described below is active as soon as the language of the search is set to Simplified Chinese (language code 'zh'). This is done either by setting a default language, by automatic language detection, or by defining the language in the CONTAINS() predicate.

If the language of the search is set to Simplified Chinese (language code 'zh'), the search options composeWords, decomposeWords and compoundWordWeight are ignored.

When comparing Chinese text values, each token of each text is compared as described above, taking the Pinyin representation into account.

Chinese script is a non-whitespace script. That means that tokens cannot easily be recognized by surrounding whitespaces. SAP HANA therefore has a sophisticated mechanism to recognize word boundaries and to split Chinese texts into words. Spelling errors can result in the wrongly entered character indicating the end of a word however, whereas the correct character would not. Alternatively, a wrongly entered character results in concatenation of subsequent characters.

A Chinese text containing a single spelling error might therefore be differently tokenized than the original text. The number of tokens and the length of the tokens can vary. In this case, it is very likely that you will get a lot of mismatches when you compare the tokens. Fuzzy search has to deal with these different tokenizations to find useful results.

For performance reasons, fuzzy search assumes that most of the spelling errors will be in the search input. Fuzzy search therefore assumes that the database entries are tokenized correctly and it handles varying tokenization in the search input only. This is done by creating alternative sets of tokens for the search input. An alternative set of tokens is built through concatenation of two or more adjacent tokens of the recognized tokens. The score for the comparison is determined by the best matching set of tokens. This behavior is similar to the behavior which could be achieved by using the search option composeWords.

Example

'Heilongjiang' is the name of a province of the People's Republic of China. '黑龙江' is the correct spelling in Chinese script. Let us assume that this is the value of TEXT field in a database table. If you perform fuzzy search with search input '黑隆江', fuzzy search has to deal with the following issue:

The search input contains a spelling error. The second character is incorrect, but has the same Pinyin representation as the correct character. Based on the spelling error, a standard tokenizer splits the search input into three tokens, whereas the correctly spelled word in the database is considered to be one token.

Table 155:

Token Set of Search Input	Token Set of Database Entry
'黑' AND '隆' AND '江'	'黑龙江'

Since every search token has to match a counterpart token in order to get a hit, the search input would never match the correctly spelled word. Only one of the three tokens of the search input has a chance of being

matched to the one token in the database field. To overcome this, fuzzy search creates additional sets of tokens to be compared with the database content. These additonal token sets are shown in the table below:

Table 156:

Additional Token Sets
‘黑隆’ AND ‘江’
‘黑’ AND ‘隆江’
‘黑隆江’

The last set of tokens – consisting of only one token – will make the match.

4.3.4.11.2.3 Using the Search Option interScriptMatching

Use the search option interScriptMatching to search records that contain Simplified Chinese and, for example, English characters in a single column.

Context

You can search with Pinyin data (Latin characters) and find records written in Chinese characters and the other way round, so that you do not have to care about the character set used in the database records. As a default behavior, you can search with a character set (for example, Latin), and do not find data that uses a different character set (for example, Chinese or Greek), so that you do not get any search results that you cannot read.

If you use `interScriptMatching=on`, each of the three user inputs 'Shanghai 市', '上海市', and 'Shanghai Shi' finds all three strings as a result.

If you use `interScriptMatching=off`:

- A search with only Latin characters does not search the Pinyin transliteration
- A search with only Chinese characters does not use the Pinyin transliteration to search for Latin strings
- A search with mixed input finds only mixed strings

Related Information

Option interScriptMatching [page 166]

4.3.4.12 Support of Tables with Client Column

Fuzzy search supports the client column (e.g. 'MNDT', 'CLNT', or 'CLIENT') in tables of SAP business applications. Thus ensures that each user gets only those search results that belong to its own client only.

4.3.4.12.1 Adding the Client Information to a WHERE Condition

For SELECT statements that use the `CONTAINS()` predicate, you can limit the search results to the current client.

Context

Procedure

Add either the client condition directly or by using the session parameter `CLIENT` to the `WHERE` clause, as shown in the following examples.

> **Note**
>
> For ABAP applications it may not be necessary to add the additional `AND` condition in the SQL statements as this is handled by the OpenSQL implementation for SAP HANA.

> **Example**

```
SELECT SCORE(), ... FROM tab
WHERE CONTAINS((col1, col2, col3), 'a b c', FUZZY(0.8))
AND CONTAINS(col4, 'x y z', FUZZY(0.7))
AND MNDT = '100'
ORDER BY SCORE() DESC;
```

```
SELECT SCORE(), ... FROM tab
WHERE CONTAINS((col1, col2, col3), 'a b c', FUZZY(0.8))
AND CONTAINS(col4, 'x y z', FUZZY(0.7))
AND MNDT = SESSION_CONTEXT('CLIENT');
ORDER BY SCORE() DESC;
```

4.3.4.12.2 Enabling Client-dependent Term Mappings and Stopwords

By defining an additional column to your term mapping or stoppword table you can use them for multiple clients.

Context

As with any other SQL feature, the implementation of term mappings and stopwords inside SAP HANA is not aware of a client column. It is not possible to specify the client column in the `SELECT` statement as part of the fuzzy search options. Nevertheless, it is possible to use a term mapping or stopword table with multiple clients as shown in the following example.

Procedure

1. Create a term mapping table with an additional client column.

```
CREATE COLUMN TABLE termmappings
(
client        NVARCHAR(3)    NOT NULL,
mapping_id    NVARCHAR(32)   NOT NULL,
list_id       NVARCHAR(32)   NOT NULL,
language_code NVARCHAR(2),
term_1        NVARCHAR(200) NOT NULL,
term_2        NVARCHAR(200) NOT NULL,
weight        DECIMAL       NOT NULL,
PRIMARY KEY (client, mapping_id)
);
```

2. To add data to the term mapping table, you have to specify the client column in addition to the other columns of the table.

```
INSERT INTO termmappings VALUES ('100',
'1','01','','AG','Aktiengesellschaft','0.9');
INSERT INTO termmappings VALUES ('100',
'2','01','','Aktiengesellschaft','AG','0.9');
```

3. To use the term mappings in a search, define a view that selects the term mappings of a given client from the table.

```
CREATE VIEW termmappingsview AS
SELECT mapping_id, list_id, language_code, term_1, term_2, weight
FROM termmappings
WHERE client = SESSION_CONTEXT('CLIENT');
```

4. Instead of using a SQL view, you can also define an attribute view in the SAP HANA Studio. You can also define a join view using the `CREATE COLUMN VIEW` statement. In this case you have to define a filter condition (also called constraint) that uses the session variable `$$client$$` and that filters for `client eq '$$client$$'`.

5. When the client information is defined in the session context, you can use the term mapping view in a `CONTAINS()` predicate.

```
CREATE COLUMN TABLE companies
(
id          INTEGER          PRIMARY KEY,
companyname SHORTTEXT(200)   FUZZY SEARCH INDEX ON
);
INSERT INTO companies VALUES ('1','SAP AG');
INSERT INTO companies VALUES ('2','SAP Aktiengesellschaft');

SELECT TO_DECIMAL(SCORE(),3,2) AS score, * FROM companies
WHERE CONTAINS(companyname, 'sap ag',
FUZZY(0.8,'termMappingTable=termmappingsview,termMappingListId=01,textSearch=c
ompare'))
ORDER BY score DESC, id;
```

4.3.4.13 Search Rules

Overview

The fuzzy search in SAP HANA allows you to search for structured database content that is similar to the user input. In this case, the user input and the records in the database are nearly the same but differ in their spelling (typing errors for example) or contain additional information (additional or missing terms for example).

One of the use cases for the fuzzy search on structured data is the prevention of duplicate records. New database records can be checked for similar and existing records in real time, just before the new record is saved.

Before saving a new customer to the database for example, the application checks for similar customers that might be duplicates of the new customer that has been entered. The application performs a number of searches and then presents to the user any existing customers that are similar to the user input. The user then decides whether to create a new customer (because the records presented are similar, but not really duplicates) or to accept one of the existing customers and continue with this customer record.

The searches performed by the application are defined by business rules that define when two customers are similar. For example, the application might consider two customers to be similar if one of the following conditions is true:

1. The customers' names and addresses are similar.
2. The customers' last names and addresses are identical, but the first names are different (could be persons living in the same household).
3. The customers' names are similar, and the dates of birth are identical.

These rules can be hard-coded in the application by writing three SELECT statements that perform the three searches defined above. Whenever the requirements for the search rules change, the application code has to be changed, tested, and deployed to the productive system. This can be costly in terms of time and the development resources needed.

Alternatively, the application can use search rules to store the rules as a configuration object in the SAP HANA database. Instead of embedding the rules in SELECT statements that are part of the application code, the application has to call a database procedure to process all rules defined in the configuration object.

If the business requirements change, only the search rule definition in the configuration object has to be changed, tested, and deployed. The interface to the database procedure that is called by the application remains unchanged. Without any change to the application code, the definition of the search rules is therefore changed, and the user immediately receives search results according to the new requirements. This results in much less effort and more flexibility when changing search rules.

Search Rule Sets Stored in the Application

Applications that do not use the SAP HANA repository or XS Advanced do not want search rule sets to be the only configuration object that is stored in SAP HANA and that has to be transported from system to system in a different way than all other application data. These applications may store the search rule set configuration in their own database and pass it as a dynamic search rule set to every call of the search rule set procedure.

Batch Processing

Search rules support a batch processing mode. This batch mode allows to compare a set of records given in an input table with a reference set of records with a single call to the search rule set procedure to find any duplicates within these two sets of data.

Supported Database Objects

You can search on attribute views, column views of type Join, and SQL views.

Attribute Views have to be modeled using the SAP HANA studio and have to be stored as objects in the SAP HANA repository. Column views of type Join and SQL views can be created using SQL statements.

Other database objects, such as row store tables, column store tables, calculation views, or analytic views, are not supported.

Important Terms

A **search rule set** is the configuration object that is stored in the SAP HANA repository and that contains the definition of the search rules. When the database procedure is called in order to perform a search, a search rule set is executed. This means that all rules that are defined in the search rule set are executed.

A **search rule** defines a condition when two records – the user input and a record in the database – are considered similar. Each rule in a search rule set is converted to a SELECT statement and is executed when the search rule set is processed.

4.3.4.13.1 Creating Search Rule Sets

Prerequisites

Before you can create a search rule set in the SAP HANA studio, you have to create a workspace and a project there. In this workspace, you create a package that will contain your rule set.

To set up the project in the SAP HANA studio, use the connection of the user who modeled the attribute view (for example, the user from the `MODELOWNER` tutorial) and add the package that you want to contain the search rule sets to the project (for example, the package from the `apps.customer` tutorial).

Procedure

1. In the SAP HANA modeler, open the *Project Explorer* view and navigate to your package.
2. From the context menu of your package, choose *New* > *Other* > *Search Rule Set*.
3. Enter a file name for your rule set. The file has to have the extension **`.searchruleset`**.
4. Open and edit the search rule set in the search rule set editor. See Working with the Search Rule Set Editor [page 254]

> **Note**
>
> It is also possible to edit search rule sets in the SAP HANA Web-based Development Workbench even if there is no editor specific to search rule sets. A standard text editor has to be used to create the search rule set xml files.
>
> When a `*.searchruleset` file is opened in the editor, it is possible to insert a code snippet that shows the xml structure of a search rule set. This can be used as a template to create a new search rule set.

5. Define the attribute view, key columns, score selection parameter, stopwords, and term mappings.
6. From the context menu of your package or search rule, choose *Team* > *Activate*.

Results

You can now execute a search with the rule set. See Executing a Search With a Rule Set [page 265]

Related Information

Tutorial: Create and Use Search Rules [page 272]

4.3.4.13.2 XML Structure of a Search Rule Set

Search rule sets are stored in XML files in the local file system. The search rule set editor reflects the structure of these files and displays the search rule set in a tree-like structure. To edit search rule sets, you need some basic knowledge of the structure of the XML files. The XML structure of a search rule set configuration is as follows:

```
+Document Root
    +Rule Set
        +Attribute View
            +Key Column 1
            +...
            +Key Column n
        +Rule 1
            +Column 1
                +optional: Column Options (text, string or date options)
            +...
            +Column n
                +optional: Column Options
        +...
        +Rule n
            +Column 1
                +optional: Column Options
            +...
            +Column n
                +optional: Column Options
```

4.3.4.13.3 Working with the Search Rule Set Editor

Context

To open a rule set in the editor, double-click a rule set file or, choose *Open With* > *Search Rule Set Editor* in the context menu.

Table 157:

Action	Description
Adding new nodes, children or siblings	To add new nodes, children or siblings, you can use the context menu of each node. You can a new key column node in the following ways for example: • Select the attribute view node and, choose *New Key Column* in the context menu. • Select a key column node and choose *New Key Column* in the context menu.

Action	Description
Deleting a node	To delete a node from the search rule set, select the node and choose *Delete* in the context menu. This deletes the node together with all its child nodes.
Changing the order of nodes	To change the order of nodes or to move nodes to other parent nodes, you can drag and drop them. With the SAP HANA studio running on Microsoft Windows, you can copy nodes below the same parent node or even to another parent node by pressing `CTRL` while dragging.
Changing the node properties	To change properties, click on the value of the property in the *Properties* view and enter the new value. Each node contains a set of properties that define the behavior of the search rules. Node properties are displayed in the *Properties* view in the SAP HANA studio when a node is selected in the tree. If the *Properties* view is not displayed, you can open it by choosing *Window* > *Show View* > *Properties*. **Note** Some node properties refer to database objects or to column names. These properties are case-sensitive, so be sure to enter all names in the correct notation.

4.3.4.13.4 Configuring Search Rule Sets

Procedure

1. Define the attribute view.

   The search rule set can be used with different types of views. For more information, see Search Rules [page 251].

   The view to be used is defined in the view node of the search rule set.

The name property contains the name of the view. The 'syntax' of the name depends on the type of the view:

Table 158:

View Type	Syntax	Example
Repository objects: ○ Attribute View	FQN notation (fully qualified name)	`apps.customer::CUSTOMER`
Catalog objects: ○ Join View ○ SQL View	"Schema"."View_Name"	`"SAP_CRM"."CUSTOMERS_VIEW"`

There are two ways of setting the name of the view:

- You can type in the name of the view in the "Properties" view using the correct syntax for catalog and repository objects.
- You can use drag & drop. Drag the catalog and repository objects from the "SAP HANA Systems", view and drop them on the view node. You can also drag tables from the catalog and drop them on the "Stopwords (table-based)" and "Term Mappings (table-based)" nodes.

Once you have defined the view, and your project is shared with a HANA repository workspace, you do not need to enter all the column names and the names of the key columns manually. The "Property View" of the corresponding node (column or key column) provides you with a combobox containing all available fields.

Example attribute view name: `apps.customer::CUSTOMER`

2. Define the key columns and the score selection parameter.

A search might return the same record more than once, as it matches more than one rule.

To enable the search to return each record just once, key columns must be defined in a way that makes records in the result set unique. For an attribute view, there is no definition of key columns available, so the key columns have to be defined for each search rule set.

The key columns are a set of one or more columns of the attribute view that clearly identify a record in the view. The key columns are therefore similar to the primary key columns of a database table.

As for primary keys, LOB types (BLOB, CLOB, NCLOB, TEXT) are not allowed as key columns.

> **Note**
>
> It is possible to create an invalid key column definition that does not make the result set unique. In this case, when running a search, an error is raised when records returned by a single rule are not unique.

By default, each search rule set contains one key column node below the attribute view node. If more columns are needed to make records unique, more key column nodes can be added below the attribute view node.

In each key column node, enter the name of the attribute view column in the properties panel.

In addition to the key columns, you have to define how the result set should be made unique. Records returned by more than one rule usually have different scores assigned. Only one combination of score and rule name can be returned in the result set.

The score selection parameter defines whether the values with the highest score or the values found with the first matching rule are returned.

You define the score selection parameter in the properties panel of the rule set node.

3. Optional: Define stopwords and term mappings.

   To use stopwords and term mappings in a search rule set, the configuration options have to be added to the rule set.

   First, open the context menu for the *Search Rule Set* node and choose *New Stopwords (table-based)* or *New Term Mappings (table based)*.

   In the properties of the new nodes, you can define the stopword table and term mapping table that is used.

   On the *Stopwords (table-based)* or *Term Mappings (table-based)* node, select *New Column* to enable the stopwords or term mappings on a column. In the properties panel, you can define the name of the column where you want stopwords and term mappings to be applied.

   Below the *Column* nodes, create new *List ID* nodes (open the context menu and choose *New List ID*). In each list ID node, you can configure a stopword or term mapping list that is applied on the column.

   The stopword and term mapping settings are valid for a given column in all rules.

4. Add rules.

   To add a new rule to the search rule set, open the context menu on the *Search Rule Set* node and choose *New Rule*.

   You can change the order of the rules by dragging a rule with the mouse to a new position.

4.3.4.13.5 Configuration Settings and Properties

Search Rule Set Properties

The only property of a search rule set is the `Score Selection`. `Score selection` defines which rule number is returned for a record if it was found by multiple rules. `highestScore` will choose the rule that gives the highest score for the record (used for non-hierarchical views or together with `Rule Cut`) and `firstRule` will pick the first rule (top down) that found this record (used for hierarchical rules).

Attribute View Properties

A search rule set needs an attribute view on which the rules are executed. To connect the search rule set with the attribute view, the node `Attribute View` with the property `Name` is used. `Name` takes the full name consisting of package and attribute view, separated by a double colon. `_SYS_RT` needs SELECT granted on this view.

Key Columns

When searching with multiple search rules, it is likely that the same row is returned by more than one rule. In this case, the search rules runtime has to make a decision on what score, rule ID, and rule number are returned for this record because `EXECUTE_SEARCH_RULE_SET` returns every row only once.

To make the results unique, the `EXECUTE_SEARCH_RULE_SET` function needs a definition of the 'key columns' that identify each row. This is done by adding a `Key Column` node for each column the key consists of. Its property `Name` takes the column's name. If the key consists of multiple columns, the same number of `Key Column` nodes have to be defined. LOB types (CLOB, NCLOB, TEXT) are not allowed for key columns.

Stopwords and Term Mapping Properties

To use stopwords, the node `Stopwords (table-based)` has to be added. It takes `Schema` and `Table` names of the stopword table as properties. `_SYS_RT` needs SELECT granted on this table. For each column that uses stopwords, a `Column` node has to be added. Its property `Name` takes the column name for which stopwords are to be used. Multiple `List ID` nodes can be defined under the `Column` node. Their property `Id` indicates a list ID of the stopword table that is used for the given column.

The same is valid for term mappings with the node `Term Mappings (table-based)`.

Rule Properties

For each rule that shall be defined, a node `Rule` has to be added. The order of the rules (top down) in the resource set also defines the order of the execution. You name the rule by setting the `Name` property.

`Min Fuzziness` defines the minimum score that the rule's columns have to result in to trigger this rule. All records with a lower score will not show in the result. The property `Weight` is a factor that lowers the rule's score (for example, a score of 0.9 with a weight of 0.9 will result in 0.81). This is used to show that the rule's hits have a lower value. `Rule Cut` is used to stop the processing before a rule is used. Whenever the total number of hits of earlier rules exceeds the value of `Rule Cut`, no more rules are processed. A value of `0` (zero) deactivates `Rule Cut` for this rule.

`Rule Cut` is used for the following reasons:

- To separate rules (or groups of rules) that have a different hierarchy by putting a `Rule Cut = 1` into the next rule (or first rule of the next group). If you do this, the next rule (or next group) will not be executed if the previous rule (or group of rules) results in hits.
- To speed up processing. This is done by putting a `Rule Cut` with the desired number of hits into every rule. As soon as this number is reached, no more rules are processed.

Column Properties

A rule needs at least one column to search on. The column's `Name` property points to a column of the `Attribute View` defined above. `Min Fuzziness` corresponds to the fuzziness of the fuzzy() function within the contains() statement. The property `Weight` is a factor that lowers the column's score (for example, a

score of 0.9 with a weight of 0.9 will result in 0.81). This is used to show that the column's hits have a lower value.

The column of the attribute view has a certain type in the table it is based on. The fuzzy search allows certain column options for each type. These translate directly into column properties:

Table 159:

Column Type	Column Option Node	Available Properties
TEXT, SHORTTEXT, any STRING column with fulltext index	Text Column Options	• `abbreviationSimilarity` • `andSymmetric` • `andThreshold` • `bestMatchingTokenWeight` • `considerNonMatchingTokens` • `emptyScore` • `enableStopwords` • `enableTermMappings` • `minTextScore` • `similarCalculationMode` • `spellCheckFactor`
Other STRING columns	String Column Options	• `emptyScore` • `similarCalculationMode` • `spellCheckFactor`
DATE, TIMESTAMP	Date Column Options	• `emptyScore` • `maxDateDistance`

The column option nodes are optional; if not specified, default values for their properties will be used.

TEXT columns (as well as SHORTTEXT and any STRING columns with fulltext index) always get the fuzzy search option `textsearch=compare` set. This is to ensure that the fuzzy score will be retrieved instead of the TF/IDF score.

The two properties `enableStopwords` and `enableTermMappings` are not standard fuzzy() parameters. However, since stopwords and term mappings are defined for a column and are independent of the rules, these parameters are needed to individually switch them off (false) for single rules. The default value of `enableStopwords` and `enableTermMappings` is `true`.

4.3.4.13.5.1 Column Conditions

Each column in a rule can have three different column conditions. These conditions steer the behavior of the column and/or the rule at runtime. Depending on how the queried rule is filled, either the column will be skipped from the where clause, or the whole rule will be skipped. It is also possible to change the requested value of the column.

The table below shows the various condition types, together with the properties that can be set for them:

Table 160:

Condition	Action	Value	Replace By	Description

`If Equals`	`skipColumn`
	`skipRule`
	`replace`
		-
		-
		value to compare to
			-
			-
			new value
				Column is given in the query to the `EXECUTE_SEARCH_RULE_SET` function. The condition matches if there is a request value given that equals the value of the column condition.

`If Not Empty`	`skipColumn`
	`skipRule`
	`replace`
		-
		-
		-
			-
			-
			new value
				Column is given in the query to the `EXECUTE_SEARCH_RULE_SET` function. The condition matches if there is any request value given.

`If Missing`	`skipColumn`
	`skipRule`
	`replace`
		-
		-
		-
			-
			-
			new value
				If no column condition is given, the behavior is as follows: Missing columns in a rule cause the rule to be skipped. Empty values will be requested as empty.

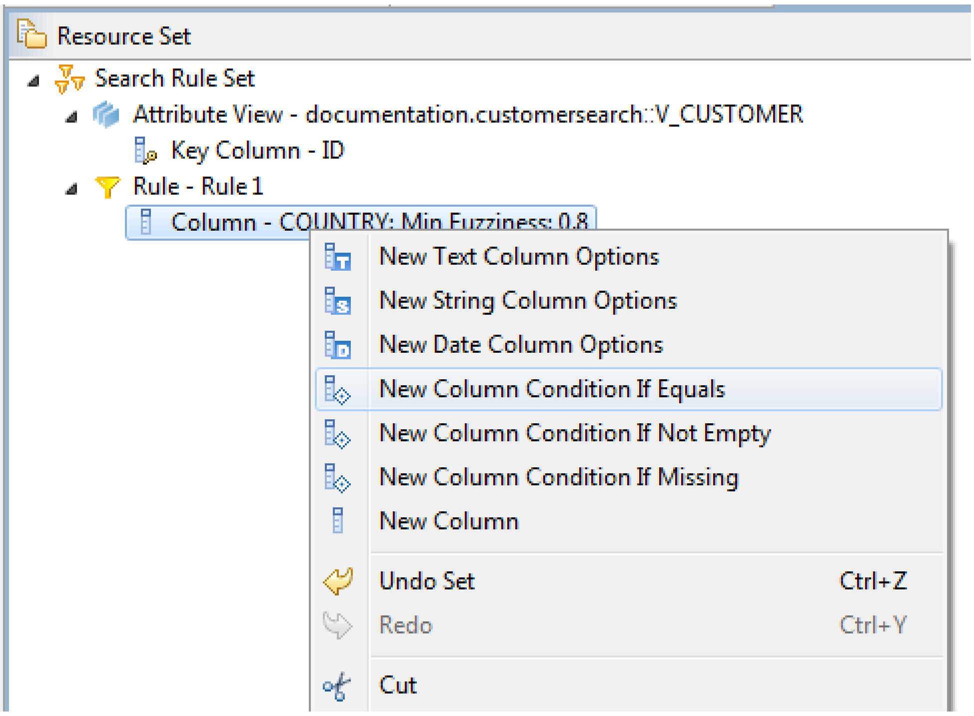
If the `if Equals` condition matches the value, the `if Not Empty` condition is also true. In this case, only the action for the `If Equals` condition will be performed.

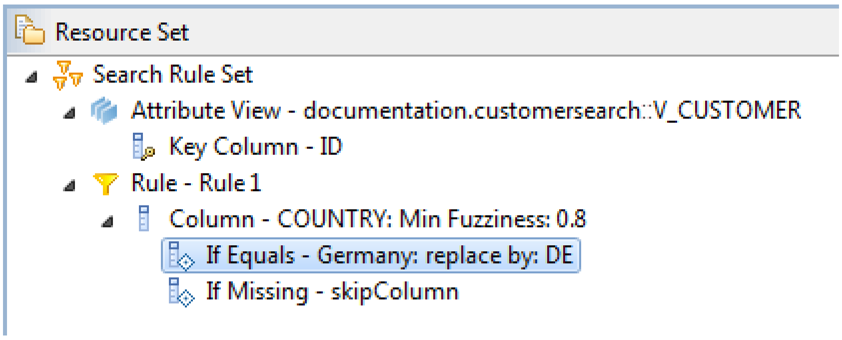
The `If Equals` condition can be given more than once per column.

If no column condition is given, the behavior is as follows: Missing columns in a rule cause the rule to be skipped. Empty values will be requested as empty.

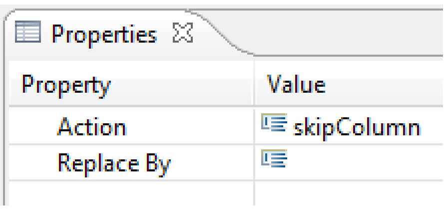
Configuring Column Conditions in the Search Rule Set Editor

1. Add a new column condition to your rule. Open the context menu on a column and choose a condition.

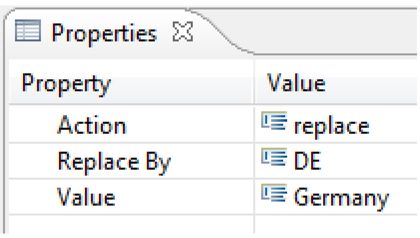




2. Set the properties for the column condition.
Example properties for a condition:



Example properties for the `If Equals` condition:



4.3.4.13.5.2 Non-Unique Keys: Using a View with an 'Anchor Table'

Sometimes, an application needs to search a view with one or more 1:n joins but is interested in the base objects (or 'anchor objects') only. For example, an application searches business partners and all of their addresses, but in the search result each business partner (the 'anchor object') should occur only once.

To get this behavior with search rule sets, only the columns of the primary key of the business partner are configured as key columns in the search rule set. Additional columns of the primary key (from the 1:n join) are not configured as key columns.

In addition, the property `nonUniqueKeys` of the `View` node in the Search Rule Set editor is set to `true`.

> **Note**
>
> If `nonUniqueKeys` is set to `false` a duplicate-keys error is returned in the following example.

Example

In this example, we assume that in the application a search for business partners (in this case companies) is done. Each business partner will be returned only once even if there are multiple addresses of the same business partner in the search result.

The business partner view contains the following two companies with three addresses in total. It is a 1:n join between a business partner table with primary key `Business Partner ID` and an address table with primary key `Address ID`.

The column **Business Partner ID** is configured as a key column in the search rule set. The **Address ID**, which is the second column of the view's primary key, is not configured as a key column in the search rule set. Therefore, the key in a search result may not be unique.

Table 161:

Business Partner ID	Company Name	Address ID	Street Name	House Number	Postcode	City Name	Country
1	Manufacturing Company	1	Norton Street	75	10016	New York	US
1	Manufacturing Company	2	Morton Street	74	10017	New York	US
2	Manufacturing Comp.	3	Norton Street	12	10020	New York	US

A search rule set with `nonUniqueKeys` set to `true` is called with the following search input:

Table 162:

Company Name	Street Name	City Name	Country
Manufacturing Company	Morton Street	New York	US

For the first business partner there are two rows found. The temporary internal search result ordered by score is shown below.

Table 163:

_Score	Business Partner ID	Company Name	Address ID	Street Name	House Number	Postcode	City Name	Country
1.0	1	Manufac-turing Company	2	Morton Street	74	10017	New York	US
0.9	1	Manufac-turing Company	1	Norton Street	75	10016	New York	US
0.8	2	Manufac-turing Comp.	3	Norton Street	12	10020	New York	US

As the `nonUniqueKeys` parameter is set to `true`, only one of the rows with **Business Partner ID**=1 will be returned to the caller.

For each rule in the search rule set only the row with the highest score is selected. If needed, conflict resolution between rules would work as usual using the **Score Selection setting** (either `highestScore` or `firstRule`).

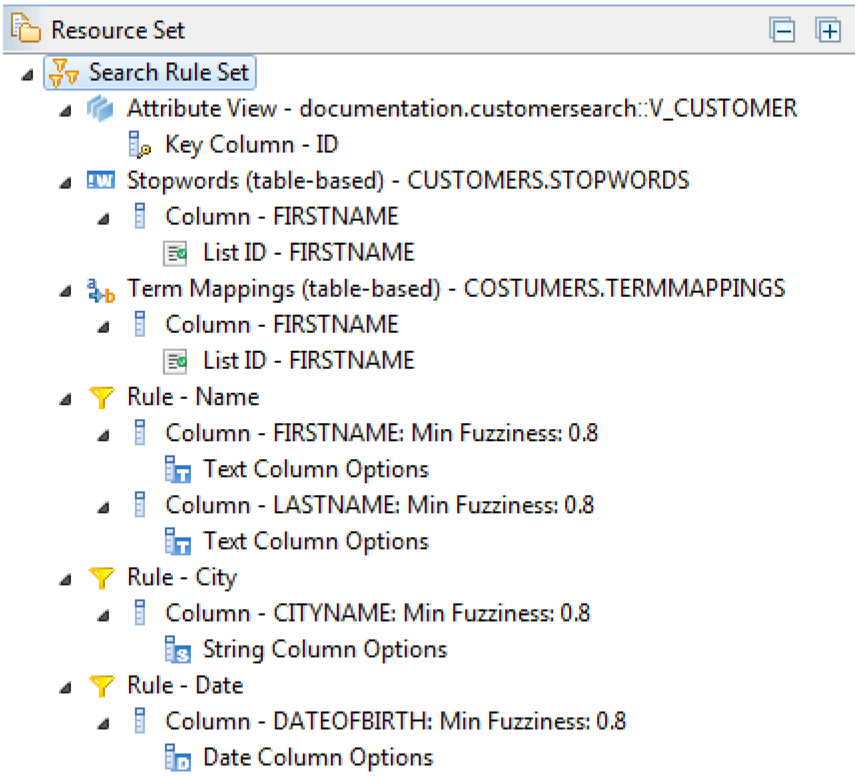
The final search result in this case is shown below:

Table 164:

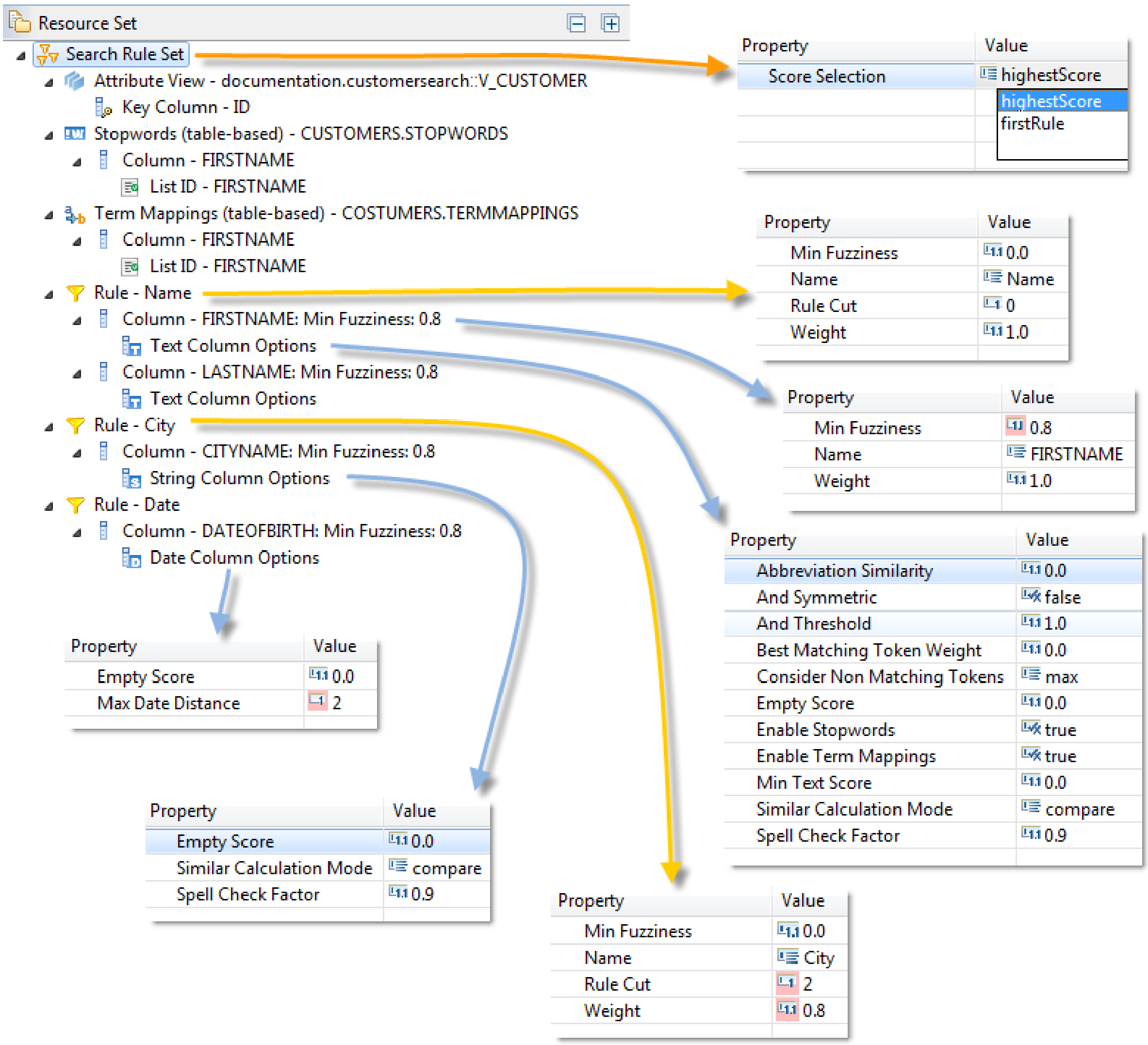
_Score	Business Partner ID	Company Name	Address ID	Street Name	House Number	Postcode	City Name	Country
1.0	1	Manufac-turing Company	2	Morton Street	74	10017	New York	US
0.8	2	Manufac-turing Comp.	3	Norton Street	12	10020	New York	US

4.3.4.13.5.3 Example Search Rule Set

Let us look at this example of a search rule set:



This rule set includes the following properties:



4.3.4.13.6 Executing a Search With a Rule Set

SAP HANA exports a new built-in function, `SYS.EXECUTE_SEARCH_RULE_SET`, that you can use to execute a previously-defined search rule set.

The function has one parameter, which is an XML string to run the search. In the XML, you have to specify the following:

- The ruleset to be executed
- How the result will be returned
- The limitation of the result
- The input values

By default, the `EXECUTE_SEARCH_RULE_SET` method returns a ResultSet object. This object contains all the columns from the referenced attribute view, plus the additional columns `_SCORE` and `_RULE_ID`. Alternatively, the `EXECUTE_SEARCH_RULE_SET` can write the result into a results table that needs to be created by the user.

Transaction Isolation Level

The `EXECUTE_SEARCH_RULE_SET` function creates one SELECT statement for each of the rules and runs the statements independently of each other. The statements are executed in the transaction context of the calling application, and use the same isolation level as the application. The isolation level influences the results of the `EXECUTE_SEARCH_RULE_SET` function if other transactions that change the contents of the database tables are running in parallel.

When the isolation level 'READ COMMITTED' is used, each of the SELECT statements of the search rule set sees all changes that had been committed when execution of the SELECT statement begins. The second rule of a rule set might see a new record that had not been committed when the first rule was executed, for example. In this case, the new record might be returned by the 'wrong' rule, and the user obtains an incorrect result.

If the isolation levels 'REPEATABLE READ' or 'SERIALIZABLE' are used, all SELECT statements see the same state of the database. The results returned by `EXECUTE_SEARCH_RULE_SET` are therefore always correct.

Available XML Tags and Parameters

Table 165:

XML Tag	Occurence	Parameter	Description
`query`	1	`limit`	Defines the maximum number of rows that are returned
	1	`offset`	Defines the number of rows skipped **Note** `limit` and `offset` work in the same way as the SQL options `limit` and `offset`
`ruleset`	0..1	`name`	Name of the ruleset to be executed
`resulttableschema`	0..1	`name`	Schema name of the resulttable
`resulttablename`	1..n	`name`	Table name of the resulttable
`column`	0..n	`name`	Input column name
`resultsetcolumn`		`name`	Defines the column to be returned. Columns are returned in the order defined by these tags. If no resultsetcolumn is defined, all columns that are defined in the attribute view plus `_SCORE` and `_RULE_ID`are returned. **Note** This tag is not valid if you use a resulttable for the search result. In this case, the resulttable defines the structure of the result.
`filter`	0..1		SQL WHERE clause limiting the search to a subset of the view contents

Predefined Columns

Table 166:

Column	SQL Type	Description
_SCORE		The Fuzzy Score for each hit in the result. It can be used in combination with a resulttable and the result-set.
_RULE_ID		The name of the rule that provided this hit. It can be used in combination with a resulttable and the result-set.
_RULE_NUMBER		The number of the rule. The first rule is given _RULE_NUMBER 1. Note The _RULE_NUMBER column is only returned if the user specifies this. Can be used in combination with a resulttable and the resultset.

Related Information

Code Examples: Searching with a Rule Set [page 267]

4.3.4.13.7 Code Examples: Searching with a Rule Set

This topic contains code examples for search rule sets.

Executing a Search and Returning the Result as a Result Set

```
CALL SYS.EXECUTE_SEARCH_RULE_SET('
<query>
  <ruleset name="documentation.customersearch:Search.searchruleset" />     --
specifies the SearchRuleSet
  <column name="FIRSTNAME">Herbert</column>   -- specifies the input value for
column FIRSTNAME
  <column name="LASTNAME">Hofmann</column>    -- specifies the input value for
column LASTNAME
</query>
');
```

Executing a Search and Writing the Result to a Column Table Provided by the User

```
--First create the result table
set schema MY_SCHEMA;
CREATE COLUMN TABLE MY_RESULT_TABLE (
_SCORE FLOAT,
_RULE_ID VARCHAR(255),
"FIRSTNAME" TEXT,
"LASTNAME" TEXT
};
-- Afterwards you can execute the search using the created result table.
CALL EXECUTE_SEARCH_RULE_SET('
<query>
  <ruleset name="documentation.customersearch:Search.searchruleset" />     --
specifies the SearchRuleSet
  <resulttableschema name="MY_SCHEMA"/>       -- specifies the schema of the
result table
  <resulttablename name="MY_RESULT_TABLE"/>   -- specifies the name of the
result table
  <column name="FIRSTNAME">Herbert</column>   -- specifies the input value for
column FIRSTNAME
  <column name="LASTNAME">Hofmann</column>    -- specifies the input value for
column LASTNAME
</query>
');
-- get the result
select * from MY_RESULT_TABLE;
```

Limiting the Number of Rows Returned by a Search

i Note

When calling the system procedure EXECUTE_SEARCH_RULE_SET, the application can define the maximum number of rows that are returned by setting a limit parameter.

In the default setting, this parameter is undefined, meaning that an unlimited number of rows is returned. The limitation takes place after each rule and at the end, when all rules are performed. In the following example, a maximum number of 100 rows will be returned.

```
-- run the search
CALL EXECUTE_SEARCH_RULE_SET('
<query limit="10" offset="100">
  <ruleset name="documentation.customersearch:Search.searchruleset" />
  <column name="FIRSTNAME">billy</column>
  <column name="LASTNAME">smith</column>
</query>
');
```

You can use this parameter with the result set object and with the custom result table.

Getting a Paged Result List by Using the Parameter Offset

When calling the system procedure `EXECUTE_SEARCH_RULE_SET`, the application can return a paged result list by using the parameters `limit` and `offset`.

- `limit` defines the number of rows returned
- `offset` defines the number of rows skipped

The following example returns a result list starting from row number 101 and ending with row number 110:

```
-- run the search
CALL EXECUTE_SEARCH_RULE_SET('
<query limit="10" offset="100">
  <ruleset name="documentation.customersearch:Search.searchruleset" />
  <column name="FIRSTNAME">billy</column>
  <column name="LASTNAME">smith</column>
</query>
');
```

The parameter `offset` is only allowed together with the parameter `limit`.

Getting a User-Defined Result Set

When calling the system procedure `EXECUTE_SEARCH_RULE_SET`, the application can return a user-defined result set instead of all columns defined in the attribute view.

```
-- run the search
CALL EXECUTE_SEARCH_RULE_SET('
<query limit="10" offset="100">
  <ruleset name="documentation.customersearch:Search.searchruleset" />
  <column name="FIRSTNAME">billy</column>
  <column name="LASTNAME">smith</column>
  <resultsetcolumn name="_SCORE" />
  <resultsetcolumn name="_RULE_ID" />
  <resultsetcolumn name="_RULE_NUMBER" />
  <resultsetcolumn name="FIRSTNAME" />
  <resultsetcolumn name="LASTNAME" />
</query>
');
```

In this case, the application returns a result set with the columns `_SCORE`, `_RULE_ID`, `_RULE_NUMBER`, `FIRSTNAME`, and `LASTNAME`.

Using Custom Names for Predefined Columns

When calling the system procedure `EXECUTE_SEARCH_RULE_SET`, the application can return custom names for internal columns `_SCORE`, `_RULE_ID`, `_RULE_NUMBER`.

```
-- run the search
CALL EXECUTE_SEARCH_RULE_SET('
<query limit="10" offset="100">
  <ruleset name="documentation.customersearch:Search.searchruleset" />
```

```
  <column name="FIRSTNAME">billy</column>
  <column name="LASTNAME">smith</column>
  <scorecolumn name="score" />
  <ruleidcolumn name="ruleid" />
  <rulenumbercolumn name="rulenumber" />
  <resultsetcolumn name="score" />
  <resultsetcolumn name="ruleid" />
  <resultsetcolumn name="rulenumber" />
  <resultsetcolumn name="FIRSTNAME" />
  <resultsetcolumn name="LASTNAME" />
</query>
');
```

In this case the application returns a resultset with the columns `_SCORE`, `_RULE_ID`, `_RULE_NUMBER`, `FIRSTNAME` and `LASTNAME`.

Adding Columns with a Constant Value to the Result Set

When calling the system procedure `EXECUTE_SEARCH_RULE_SET`, the application can return a result set with additional columns, defined in the call of the procedure.

```
-- run the search
CALL EXECUTE_SEARCH_RULE_SET('
  <query limit="10" offset="100">
  <ruleset name="documentation.customersearch:Search.searchruleset" />
  <column name="FIRSTNAME">billy</column>
  <column name="LASTNAME">smith</column>
  <constantcolumn name="USER_NAME">Max</constantcolumn>
  <constantcolumn name="USER_ID">42</constantcolumn>
</query>
');
```

In this case, the application returns a result set with the columns `_SCORE`, `_RULE_ID`, all the view columns, plus `USER_NAME`, and `USER_ID`. The last two columns have the values "Max" and "42", respectively, for all result rows.

This feature is also available for the result table. The additional columns have to be created in the result table with any SQL type. The columns have to be defined in the call to `EXECUTE_SEARCH_RULE_SET` as shown in the example above.

Example

```
--First create the result table
set schema MY_SCHEMA;
CREATE COLUMN TABLE MY_RESULT_TABLE (
"_SCORE" FLOAT,
"_RULE_ID" VARCHAR(255),
"FIRSTNAME" TEXT,
"LASTNAME" TEXT,
"USER_NAME" NVARCHAR(100),
"USER_ID" INTEGER
};
-- Afterwards you can execute the search using the created result table.
CALL EXECUTE_SEARCH_RULE_SET('
<query>
  <ruleset name="apps.customer:Search.searchruleset" />     -- specifies the
SearchRuleSet
  <resulttableschema name="MY_SCHEMA"/>       -- specifies the schema of the
result table
```

```
  <resulttablename name="MY_RESULT_TABLE"/>   -- specifies the name of the
result table
  <column name="FIRSTNAME">Herbert</column>   -- specifies the input value for
column FIRSTNAME
  <column name="LASTNAME">Hofmann</column>    -- specifies the input value for
column LASTNAME
  <constantcolumn name="USER_NAME">Max</constantcolumn>
  <constantcolumn name="USER_ID">42</constantcolumn>
</query>
');
-- get the result
select * from MY_RESULT_TABLE;
```

This feature is also available in conjunction with `<resultsetcolumn>`. You need to define the constant columns as resultsetcolumns to specify the order. You also need to define the constant columns as shown above to specify the constant values.

```
-- run the search
CALL EXECUTE_SEARCH_RULE_SET('
<query limit="10" offset="100">
  <ruleset name="documentation.customersearch:Search.searchruleset" />
  <column name="FIRSTNAME">billy</column>
  <column name="LASTNAME">smith</column>
  <resultsetcolumn name="USER_ID" />
  <resultsetcolumn name="_SCORE" />
  <resultsetcolumn name="_RULE_ID" />
  <resultsetcolumn name="USER_NAME" />
  <resultsetcolumn name="FIRSTNAME" />
  <resultsetcolumn name="LASTNAME" />
  <constantcolumn name="USER_NAME">Max</constantcolumn>
  <constantcolumn name="USER_ID">42</constantcolumn>
</query>
');
```

Limiting the Search to a Subset of the View Contents

```
-- run the search
CALL EXECUTE_SEARCH_RULE_SET('
<query>
  <ruleset name="apps.customer:Search.searchruleset" />
  <column name="FIRSTNAME">billy</column>
  <column name="LASTNAME">smith</column>
  <resultsetcolumn name="_SCORE" />
  <resultsetcolumn name="_RULE_ID" />
  <resultsetcolumn name="_RULE_NUMBER" />
  <resultsetcolumn name="FIRSTNAME" />
  <resultsetcolumn name="LASTNAME" />
  <filter>"ORDER_DATE" BETWEEN '2012-09-01' AND '2012-12-31'</filter>
</query>
');
```

This filter limits the search to documents with an `ORDER_DATE` in the specified range. The filter condition is evaluated in every rule, before evaluating the rule cut option and the query parameters `limit` and `offset`.

The filter condition must be a valid SQL WHERE clause with some extra rules:

- Comparative operators are allowed, for example: =, <, <=, >, >=
- Only a limited set of SQL keywords is allowed: NOT, AND, OR, BETWEEN, IN, LIKE, ESCAPE, IS, NULL, UPPER, LOWER, SUBSTRING

- The `CONTAINS()` function is not allowed, since this would affect the score() that is returned
- Only columns defined in the attribute view are allowed
- All column identifiers must be enclosed in double quotes (")
- All string values must be enclosed in a pair of two single quotes ('')

```
<filter>"CITY_NAME" = ''Walldorf''</filter>
```

Searching Views with a Client Column

The filter condition described above can be used to search views that have been created by SAP business applications. Usually these applications define tables and views with a client column. This column is, for example, called `MNDT`, `CLNT`, or `CLIENT`. Limit search results to rows belonging to the same client as the current user. This can be done by adding an appropriate filter condition to the call to `EXECUTE_SEARCH_RULE_SET`, as shown in the following example.

```
-- run the search
CALL EXECUTE_SEARCH_RULE_SET('
<query>
  <ruleset name="apps.customer:Search.searchruleset" />
  <column name="FIRSTNAME">billy</column>
  <column name="LASTNAME">smith</column>
  <filter>"MNDT" = '100'</filter>
</query>
');
```

4.3.4.13.8 Tutorial: Create and Use Search Rules

This tutorial documentation describes the development of an application that uses the search rule sets to search on a table containing a company's customers.

Prerequisites

As a developer who wants to create search rule sets, you are familar with the SAP HANA development environment.

The development tools for search rule sets are part of the development environment for native SAP HANA applications. Nevertheless, search rule sets can be used in all types of SAP HANA applications.

The Scenario

This tutorial documentation describes the development of an application that uses the search rule sets to search on a table containing a company's customers. The data model used for this tutorial is very simple. Let us assume that all customers are people, so no company information can be stored in the table.

Users and Roles

When building an application based on SAP HANA, different database users are created for modeling content and for running the application. Privileges have to be granted to give each user a minimum set of access rights needed to perform tasks corresponding to his or her role.

The scenario in this documentation uses the following database users:

- The data stored in the column tables is owned by the first user called `TABLEOWNER`.
- Attribute views and search rules are modeled by a second user called `MODELOWNER`.
- Finally, the application runs with a third user called `APPOWNER`.

Instead of granting privileges to users directly, you can use roles.

4.3.4.13.8.1 Creating the Application Package

Context

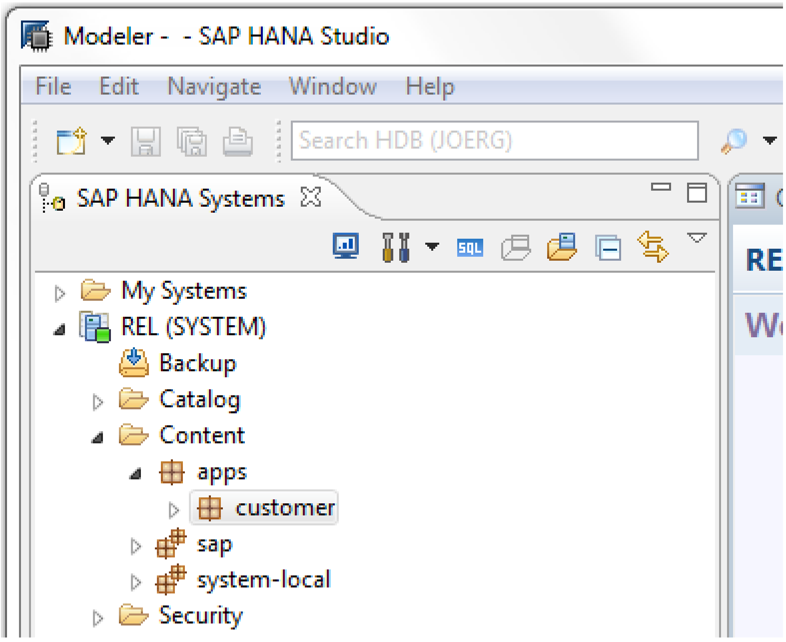
In our example, the database content (the attribute view and the search rule set) will be stored in a package called "apps.customer".

Procedure

In the SAP HANA studio, create a new package named *customer*.

You can create the package in the SAP HANA Systems view of the SAP HANA studio (either in the Modeler perspective or in the SAP HANA Development perspective).

Use the SYSTEM user or any other user with sufficient privileges.



4.3.4.13.8.2 Creating the Users

You have to create three users and and assign a set of privileges to them. The users are created by the administration user SYSTEM or by any other user with sufficient privileges.

Procedure

1. Create the TABLEOWNER

```
CREATE USER tableowner PASSWORD "********";
-- privilege needed to create the application schema
GRANT CREATE SCHEMA TO tableowner;
```

2. Create the MODELOWNER

```
CREATE USER modelowner PASSWORD "********";
-- Allow repository access from SAP HANA studio
GRANT EXECUTE ON _sys_repo.repository_rest TO modelowner;
-- Privileges needed to access packages in the HANA repository
-- read access to packages and designtime objects (native and imported
objects)
GRANT REPO.READ                          ON "apps.customer" TO modelowner;
-- all kinds of inactive changes to design-time objects in native packages
GRANT REPO.EDIT_NATIVE_OBJECTS       ON "apps.customer" TO modelowner;
-- activate / reactivate design-time objects in native packages
GRANT REPO.ACTIVATE_NATIVE_OBJECTS   ON "apps.customer" TO modelowner;
-- create, update or delete native packages, or create subpackages of native
packages
GRANT REPO.MAINTAIN_NATIVE_PACKAGES ON "apps.customer" TO modelowner;
-- permission go grant and revoke privileges on activated content (like, for
example, attribute views)
```

```
GRANT EXECUTE ON GRANT_PRIVILEGE_ON_ACTIVATED_CONTENT TO modelowner;
GRANT EXECUTE ON REVOKE_PRIVILEGE_ON_ACTIVATED_CONTENT TO modelowner;
-- A user needs an analytical privilege to access activated attribute views.
--
-- The predefined privilege _SYS_BI_CP_ALL gives access to all activated
content.
-- Create your own analytical privilege if a limited access to activated
content is needed.
CALL _SYS_REPO.GRANT_ACTIVATED_ANALYTICAL_PRIVILEGE('_SYS_BI_CP_ALL',
'MODELOWNER');
-- privileges needed to run the search rules
GRANT EXECUTE ON sys.execute_search_rule_set TO modelowner;
```

> **Note**
>
> _SYS_RT.SEARCH_RULE_SETS Table
>
> The _SYS_RT.SEARCH_RULE_SETS must not be queried in applications using search rule sets.

If you do not want to assign a large number of privileges to each modeling user, you can create a new role containing all required privileges and assign this role to the users. This is the preferred option, but alternatively you can also use the predefined MODELING or CONTENT_ADMIN roles to grant the privileges.

3. Create the APPOWNER

```
CREATE USER appowner PASSWORD "********";
-- A user needs an analytical privilege to access activated attribute views.
CALL _SYS_REPO.GRANT_ACTIVATED_ANALYTICAL_PRIVILEGE('_SYS_BI_CP_ALL',
'APPOWNER');
-- privileges needed to run the search rules
GRANT EXECUTE ON sys.execute_search_rule_set TO appowner;
```

> **Note**
>
> _SYS_RT.SEARCH_RULE_SETS Table
>
> The _SYS_RT.SEARCH_RULE_SETS must not be queried in applications using search rule sets.

4.3.4.13.8.3 Creating the Database Schema and Tables

Context

You have to create a new database schema called 'CUSTOMERS' to store the database tables.

Then, you create the table 'CUSTOMER' in the new schema to store information about customers such as name, address, phone number, and date of birth. The example uses a simple table definition only, but more complex table definitions including 1:n relationships between names and addresses are possible.

The schema and table are created by the user TABLEOWNER.

Procedure

Creating the Database Schema and Tables

```
CREATE SCHEMA customers;
SET SCHEMA customers;
-- create sample table
CREATE COLUMN TABLE customer
(
  id            INTEGER          PRIMARY KEY,
  firstname     SHORTTEXT(100)   FUZZY SEARCH INDEX ON,
  lastname      SHORTTEXT(100)   FUZZY SEARCH INDEX ON,
  streetname    NVARCHAR(100)    FUZZY SEARCH INDEX ON,
  housenumber   NVARCHAR(20)                          FUZZY SEARCH MODE
'housenumber',
  postcode      NVARCHAR(20)     FUZZY SEARCH INDEX ON FUZZY SEARCH MODE
'postcode',
  cityname      NVARCHAR(100)    FUZZY SEARCH INDEX ON,
  countrycode   NVARCHAR(2),
  phone         NVARCHAR(20),
  dateofbirth   DATE
);
-- needed to model an attribute view on top of the table
GRANT SELECT ON customer TO modelowner;
-- needed to allow activation of attribute views that use this table
GRANT SELECT ON customer TO _sys_repo WITH GRANT OPTION;
-- for better performance, database indexes should be created
-- on all NVARCHAR columns that are used in the search rules
-- example:
CREATE INDEX customer_cityname ON customer(cityname);
-- insert a sample record:
INSERT INTO customer VALUES(1, 'Billy', 'Smith', 'Summerset Drv', '1001',
'123456789', 'Littleton', 'US', '555-1234', '1950-12-01');
-- to be able to use stopwords a stopword table is needed:
CREATE COLUMN TABLE stopwords
(
  stopword_id    VARCHAR(32)    PRIMARY KEY,
  list_id        VARCHAR(32)    NOT NULL,
  language_code  VARCHAR(2),
  term           NVARCHAR(200)  NOT NULL
);
GRANT SELECT ON stopwords TO _sys_repo WITH GRANT OPTION;
INSERT INTO stopwords VALUES('1', 'firstname', '', 'Dr');
-- and for term mappings another table:
CREATE COLUMN TABLE termmappings
(
  mapping_id    VARCHAR(32)   PRIMARY KEY,
  list_id       VARCHAR(32)   NOT NULL,
  language_code VARCHAR(2),
  term_1        NVARCHAR(255) NOT NULL,
  term_2        NVARCHAR(255) NOT NULL,
  weight        DECIMAL       NOT NULL
);
GRANT SELECT ON termmappings TO _sys_repo WITH GRANT OPTION;
INSERT INTO termmappings VALUES('1', 'firstname', '', 'William', 'Will',
'1.0');
INSERT INTO termmappings VALUES('2', 'firstname', '', 'William', 'Bill',
'0.9');
INSERT INTO termmappings VALUES('3', 'firstname', '', 'William', 'Billy',
'0.9');
INSERT INTO termmappings VALUES('7', 'firstname', '', 'Will',    'William',
'1.0');
INSERT INTO termmappings VALUES('8', 'firstname', '', 'Will',    'Bill',
'0.9');
INSERT INTO termmappings VALUES('9', 'firstname', '', 'Will',    'Billy',
'0.9');
```

```
INSERT INTO termmappings VALUES('4', 'firstname', '', 'Bill',    'William',
'0.9');
INSERT INTO termmappings VALUES('5', 'firstname', '', 'Bill',    'Will',
'0.9');
INSERT INTO termmappings VALUES('6', 'firstname', '', 'Bill',    'Billy',
'1.0');
```

4.3.4.13.8.4 Defining the Attribute View

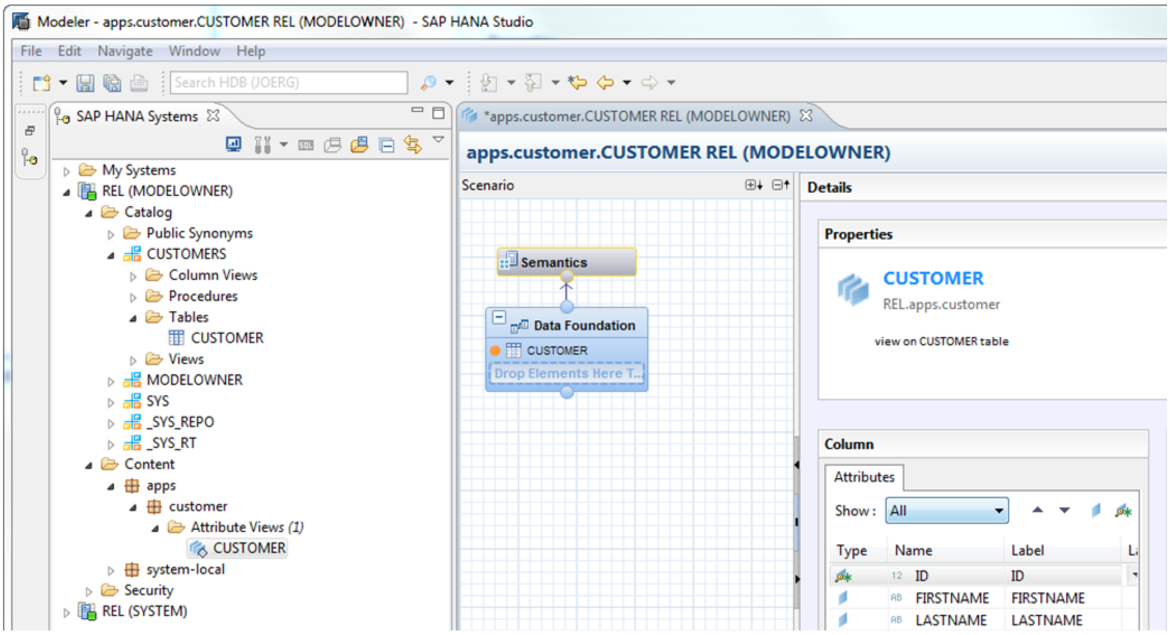
As the user MODELOWNER, you create the attribute view that is used for the search.

Context

The attribute view is created in the SAP HANA Systems view that is part of the SAP HANA studio (either in the Modeler perspective or in the SAP HANA Development perspective).

Procedure

1. In the package 'apps.customer', create a new attribute view called CUSTOMER (containing all columns from table CUSTOMERS.CUSTOMER).
2. Save and activate the attribute view.



A SELECT privilege is required for a user to see the contents of the activated attribute view.

```
-- allow users MODELOWNER and APPOWNER to see the contents of the attribute
view
```

```
CALL GRANT_PRIVILEGE_ON_ACTIVATED_CONTENT('SELECT',
'"apps.customer::CUSTOMER"', 'MODELOWNER');
CALL GRANT_PRIVILEGE_ON_ACTIVATED_CONTENT('SELECT',
'"apps.customer::CUSTOMER"', 'APPOWNER');
```

The contents of the view are now visible to users MODELOWNER and APPOWNER.

```
-- test the view
SELECT * FROM "apps.customer::CUSTOMER";
```

4.3.4.13.8.5 Creating the Search Rule Set

Read about the steps needed to define a search rule set and to run the final application.

Procedure

1. Preparing the Development Environment

   The SAP HANA studio is used to create, modify, and activate search rule sets.

   Before creating your first search rule set, you have to set up a native SAP HANA Development project in the SAP HANA studio:

   1. Open the SAP HANA Development perspective.
   2. Create a new repository workspace in the SAP HANA Repositories view.
      Use the database connection of user MODELOWNER for the repository workspace.
   3. Create a new project called 'project' in package 'apps.customer'.
      - Select 'General/Project' for a project that contains search rule sets only.
      - Select 'SAP HANA Development/XS Project' for native SAP HANA projects.
      - Use 'Team/Share Project' to move the project to the repository package 'apps.customer'.
2. Defining the Search Rule Set

   The next step for user MODELOWNER is to define the search rule set that will be used by the application. This is done in the Project Explorer view in the SAP HANA studio.

   The steps to define the search rule set are as follows:

   1. In your project, create a new search rule set 'Search.searchruleset'.
      The search rule set is created in the package 'apps.customer.project'.
   2. Edit the search rule set.
      - Define view "apps.customer::CUSTOMER" as the view that will be searched.
      - Define at least one rule (for example, use columns FIRSTNAME and LASTNAME in the rule).
   3. Save the search rule set.
   4. Commit and activate your changes.

   Now you can call the search in the SAP HANA studio:

```
-- run the search
CALL SYS.EXECUTE_SEARCH_RULE_SET('
```

```
<query>
  <ruleset name="apps.customer:Search.searchruleset" />
  <column name="FIRSTNAME">Dr. bill</column>
  <column name="LASTNAME">smiths</column>
  <column name="CITYNAME">littleton</column>
  <column name="DATEOFBIRTH">1950-12-02</column></query>
');
```

3. Running the Application

   The application user (APPOWNER) can now build an application that uses the attribute view and the search rules defined previously.

   In the application, you can call the following statements:

```
-- show contents of attribute view
SELECT * FROM "apps.customer::CUSTOMER";
-- run the search
CALL SYS.EXECUTE_SEARCH_RULE_SET('
<query>
  <ruleset name="apps.customer:Search.searchruleset" />
  <column name="FIRSTNAME">Dr. bill</column>
  <column name="LASTNAME">smiths</column>
  <column name="CITYNAME">littleton</column>
  <column name="DATEOFBIRTH">1950-12-02</column></query>
');
```

4.3.4.13.9 Information Views

Applications can query information about search rule sets to allow their UI to be aligned with the current configuration.

The following views provide the information needed by the search applications:

- `SYS.SEARCH_RULE_SETS`
- `SYS.SEARCH_RULE_SET_CONDITIONS`

SYS.SEARCH_RULE_SETS

Applications need to know which search rule sets are defined for a given database object. Example: When a database object is searched, the user can select from a list of available search rule sets valid for the database object.

The database objects used by the search rule sets can be either catalog objects like 'normal' SQL views, column tables or column views created using SQL, or design time objects like attribute views or column tables that are activated using the SAP HANA repository.

Catalog objects are referenced as `SEARCHED_SCHEMA_NAME` and `SEARCHED_OBJECT_NAME` in the view.

For design time objects, developers do not need to know the name of the resulting catalog object and it the search rule set the name of the design time object is used. Design time objects therefore are referenced as `SEARCHED_REPOSITORY_PACKAGE_ID` and `SEARCHED_REPOSITORY_OBJECT_NAME`.

When a search rule set is activated, no catalog objects are created. All information is written to the `_SYS_REPO.SEARCH_RULE_SETS`. table The search rule set can therefore be referenced as a design time object only.

Table 167:

Column	Type	Description
`SEARCH_RULE_SET_PACKAGE_ID`	NVARCHAR(256)	The name of the repository package of the search rule set.
`SEARCH_RULE_SET_NAME`	NVARCHAR(256)	The name of the search rule set.
`SEARCHED_SCHEMA_NAME*`	NVARCHAR(256)	The schema of the catalog object that is searched by the given search rule set.
`SEARCHED_OBJECT_NAME*`	NVARCHAR(256)	The name of the catalog object that is searched by the given search rule set.
`SEARCHED_REPOSITORY_PACKAGE_ID*`	NVARCHAR(256)	The name of the repository package of the repository object that is searched by the given search rule set.
`SEARCHED_REPOSITORY_OBJECT_NAME*`	NVARCHAR(256)	The name of the repository object that is searched by the given search rule set.

*: Either `SEARCHED_SCHEMA_NAME` and `SEARCHED_OBJECT_NAME` or `SEARCHED_REPOSITORY_PACKAGE_ID` and `SEARCHED_REPOSITORY_OBJECT_NAME` are given.

SYS.SEARCH_RULE_SET_CONDITIONS

Applications need to know which columns are accepted by a search rule set as input columns. This information is used to align the UI with the search rule set (for example by hiding columns that are part of the view that is searched, but are not used by the search rule set).

A rule in a search rule set contains conditions that define if a rule is executed or not. A rule may be skipped if a column is empty for example.

An application can use the definition of column conditions to inform the user of the minimal input, so that at least one rule is executed.

Table 168:

Column	Type	Description
`SEARCH_RULE_SET_PACKAGE_ID`	NVARCHAR(256)	The name of the repository package of the search rule set.
`SEARCH_RULE_SET_NAME`	NVARCHAR(256)	The name of the search rule set.
`RULE_NUMBER`	INTEGER	The number of the rule that contains the column.
`RULE_ID`	NVARCHAR(256)	The ID of the rule that contains the column.

Column	Type	Description
SEARCHED_COLUMN_NAME	NVARCHAR(256)	The name of a column that is used by a search rule set (a 'searchable column').
SEARCHED_COLUMN_CONDITION	NVARCHAR(20)	The condition. Possible values are 'EQUALS', 'NOT EMPTY', 'MISSING'.
SEARCHED_COLUMN_VALUE	NVARCHAR(5000)	The value to compare if condition is 'EQUALS'.
ACTION	NVARCHAR(20)	The action that is performed if the condition is true. Possible values are 'SKIP COLUMN', 'SKIP RULE', 'REPLACE'.

4.3.4.13.10 Search Rule Sets in Batch Mode

The batch mode is introduced to provide the possibility to do mass data processing with a high degree of parallelization.

The batch mode can be used with dynamic search rule sets as well. The data to be searched with, is not provided within the runtime XML itself, but via a table or view that gets iterated and search requests are processed for each row of this input view. The output is as well provided in two tables.

The score table stores each requesting record from the input view, along with all matches identified, within one request group. The group table is meant to calculate final match groups out of the request groups as the request groups likely store the same record in different match groups, but for reviewing purpose such groups should be molten together according to business needs.

The score table is based on the structure of the result table, in addition it consists of the predefined columns _GROUP_ID and _SOURCE_ID along with all keys from the search rule set definition and all keys from the input view. The column names for input view keys are concatenated of the input view alias + "_" + the key column name, in the example below "INPUT_ID".

The batch mode is used with the defintion of <batch> tag. Instead of a list of <column> as in transaction use case, the input is provided through the definition of an <inputview>.

Sample Code

```
-- run a search rule set in batch mode
CALL EXECUTE_SEARCH_RULE_SET('
<query>
  <ruleset name="apps.customer:Search.searchruleset" />
  <batch scoretableschema="MY_SCHEMA" scoretablename="MY_SCORE_TABLE"
threads="20">
        <inputview schema="MY_SCHEMA" name="V_INPUT_DATA" alias="INPUT"
blocksize="3000" limit="1000000">
            <keycolumn name="ID" alias="ID"/>
            <mapcolumn inputcolumn="FIRSTNAME" rulesetcolumn="FIRSTNAME"/>
            <mapcolumn inputcolumn="LASTNAME" rulesetcolumn="LASTNAME"/>
            <mapcolumn inputcolumn="CITY" rulesetcolumn="CITY"/>
            <mapcolumn inputcolumn="STREET" rulesetcolumn="STREET"/>
            <mapcolumn inputcolumn="BIRTHDATE" rulesetcolumn="BIRTHDATE"/
>
        </inputview>
    </batch>
```

```
</query>
');
```

Score Table

The score table, similar to the result table, is used to store the results of each request within a batch run. The structure of the score table is derived from the runtime XML and must match the expected structure. The score table expects the four predefined columns, `_SCORE`, `_RULE_ID`, `_GROUP_ID`, `_SOURCE_ID` plus all keycolumns defined in the search rule set plus all keycolumns defined in the input view. The input view keycolumn column names are the concatenated values of the input view alias plus a "_" plus the keycolumn alias.

The runtime XML example above expects a score table structure as following:

Table 169:

Column	Type
`_SCORE`	DOUBLE
`_RULE_ID`	VARCHAR
`_GROUP_ID`	VARCHAR
`_SOURCE_ID`	VARCHAR
`ID`	VARCHAR
`INPUT_ID`	VARCHAR

The names of these columns can be overwritten, the lenghts of the VARCHAR fields must reflect the expected content.

Group Table

The group table provides final calculated match groups, where, different to the score table, each record is only part of one group.

Table 170:

Column	Type
`_GROUP_ID`	VARCHAR
`_SOURCE_ID`	VARCHAR

Column	Type
ID	VARCHAR
INPUT_ID	VARCHAR

4.3.4.13.11 Dynamic Search Rule Sets

With this feature you can use the functionality of search rule sets without having the need to first activate the search rule set via the SAP HANA repository or SAP HANA HDI. Within the XML-Tag `<ruleset>` you can store a complete rule set definition.

The name attribute of `<ruleset>` must be left out, the optional attribute `scoreSelection` can be set.

Sample Code

```
CALL EXECUTE_SEARCH_RULE_SET('
<query>

  <ruleset scoreSelection="firstRule">
      <attributeView name="APP.CUSTOMER">
        <keyColumn name="ID"/>
      </attributeView>
      <termMappingsTableBased schema="APP" table="TERMMAPPINGS">
        <column name="FIRSTNAME">
          <list id="firstname"/>
        </column>
        <column name="LASTNAME">
          <list id="lastname"/>
        </column>
      </termMappingsTableBased>
      <stopwordsTableBased schema="APP" table="STOPWORDS">
        <column name="COMPANYNAME">
          <list id="company"/>
        </column>
      </stopwordsTableBased>
      <rule name="Rule 1">
        <column minFuzziness="0.8" name="FIRSTNAME">
          <ifMissing action="skipColumn"/>
        </column>
        <column minFuzziness="0.8" name="LASTNAME">
          <ifMissing action="skipColumn"/>
        </column>
        <column minFuzziness="0.8" name="COMPANYNAME">
          <textColumnOptions />
          <ifMissing action="skipColumn"/>
        </column>
      </rule>
  </ruleset>
  <column name="FIRSTNAME">billy</column>
  <column name="LASTNAME">smith</column>
</query>
');
```

Related Information

Configuration Settings and Properties [page 257]

4.3.4.14 Support Information

This section contains information on monitoring views, traces, and sizing.

4.3.4.14.1 Monitoring Views

This topic lists the monitoring views related to fuzzy search and fuzzy search indexes.

Checking the memory usage of all fuzzy search indexes with M_FUZZY_SEARCH_INDEXES

The M_FUZZY_SEARCH_INDEXES view shows the memory usage of all fuzzy search indexes. A fuzzy search index has no name and is described by its schema name, table name and column name.

The view shows all columns with a fuzzy search index in a table regardless of whether or not it is loaded.

Shadow columns (SHORTTEXT, FULLTEXT INDEX) are also shown in the view.

The view has the following columns:

Table 171:

View Column	SQL Data Type	Dimension	Default	Comment
HOST	VARCHAR	64		Host name
PORT	INTEGER			Internal port
SCHEMA_NAME	NVARCHAR	256		Schema name
TABLE_NAME	NVARCHAR	256		Table name
COLUMN_NAME	NVARCHAR	256		Column name
PART_ID	INTEGER			Partition ID; 0 for non-partitioned tables; 1 through n for the partitions
FUZZY_SEARCH_MODE	VARCHAR	16		'default' or 'postcode'/'house number' (same as in SYS.Columns view)

View Column	SQL Data Type	Dimension	Default	Comment
MEM-ORY_SIZE_IN_TOTAL	BIGINT			Sum of MEM-ORY_SIZE_IN_MAIN and MEM-ORY_SIZE_IN_DELTA
MEM-ORY_SIZE_IN_MAIN	BIGINT			Current memory con-sumption of the fuzzy search index of the column in main index
MEM-ORY_SIZE_IN_DELTA	BIGINT			Current memory con-sumption of the fuzzy search index of the column in delta index
LOADED	VARCHAR	5		'TRUE' or 'FALSE'. Flag to indicate that the fuzzy search index of the column is loaded

Example: Memory usage in delta index

```
ID INT PRIMARY KEY,
HNR    VARCHAR(255));
INSERT INTO FUZZY_HOUSENUMBERS VALUES ('1','10');
INSERT INTO FUZZY_HOUSENUMBERS VALUES ('2','10-12');
INSERT INTO FUZZY_HOUSENUMBERS VALUES ('3','10a');
INSERT INTO FUZZY_HOUSENUMBERS VALUES ('4','10b');
ALTER TABLE FUZZY_HOUSENUMBERS ALTER (hnr nvarchar(255) FUZZY SEARCH INDEX ON);
SELECT to_int(score() * 100) AS s, id, hnr FROM FUZZY_HOUSENUMBERS WHERE
contains((hnr), '10a', FUZZY(0.8)) ORDER BY s DESC,id;
SELECT * FROM SYS.M_FUZZY_SEARCH_INDEXES WHERE TABLE_NAME = 'FUZZY_HOUSENUMBERS';
```

The result:

Table 172:

Column	Value
HOST	YOUR HOST
PORT	YOUR PORT
SCHEMA_NAME	SYSTEM
TABLE_NAME	FUZZY_HOUSENUMBERS
COLUMN_NAME	HNR
PART_ID	0
FUZZY_SEARCH_MODE	DEFAULT
MEMORY_SIZE_IN_TOTAL	2752
MEMORY_SIZE_IN_MAIN	0
MEMORY_SIZE_IN_DELTA	2752
LOAD	TRUE

Memory usage in main index:

```
MERGE DELTA OF FUZZY_HOUSENUMBERS;
SELECT * FROM SYS.M_FUZZY_SEARCH_INDEXES WHERE SCHEMA_NAME = 'SYSTEM' AND
TABLE_NAME = 'FUZZY_HOUSENUMBERS';
```

The result:

Table 173:

Column	Value
HOST	YOUR HOST
PORT	YOUR PORT
SCHEMA_NAME	SYSTEM
TABLE_NAME	FUZZY_HOUSENUMBERS
COLUMN_NAME	HNR
PART_ID	0
FUZZY_SEARCH_MODE	DEFAULT
MEMORY_SIZE_IN_TOTAL	2704
MEMORY_SIZE_IN_MAIN	2704
MEMORY_SIZE_IN_DELTA	0
LOAD	TRUE

> **Note**
>
> The memory values can vary depending on your environment.

Checking the memory usage with M_HEAP_MEMORY

All data structures for fuzzy search share a common 'Pool/FuzzySearch' allocator. Statistics can be obtained from the system view `M_HEAP_MEMORY`.

```
SELECT * FROM m_heap_memory WHERE category LIKE '%FuzzySearch%';
```

Table 174:

Column	Value
HOST	"lu3412587"
PORT	30003
VOLUME_ID	2
STATISTICS_ID	36723
CATEGORY	"Pool/FuzzySearch"
DEPTH	2

Column	Value
INCLUSIVE_SIZE_IN_USE	4989216
INCLUSIVE_COUNT_IN_USE	14399
INCLUSIVE_ALLOCATED_SIZE	15428600
INCLUSIVE_DEALLOCATED_SIZE	10439384
INCLUSIVE_ALLOCATED_COUNT	49177
INCLUSIVE_DEALLOCATED_COUNT	34778
INCLUSIVE_MAX_SINGLE_ALLOCATION_SIZE	524288
INCLUSIVE_PEAK_ALLOCATION_SIZE	4989216
EXCLUSIVE_SIZE_IN_USE	4989216
EXCLUSIVE_COUNT_IN_USE	14399
EXCLUSIVE_ALLOCATED_SIZE	15428600
EXCLUSIVE_DEALLOCATED_SIZE	10439384
EXCLUSIVE_ALLOCATED_COUNT	49177
EXCLUSIVE_DEALLOCATED_COUNT	34778
EXCLUSIVE_MAX_SINGLE_ALLOCATION_SIZE	524288
EXCLUSIVE_PEAK_ALLOCATION_SIZE	4989216
EXCLUSIVE_ALLOC_ERRORS	0
MALLOC_PROXY_CACHE_MISSES	0
FLAGS	"(none)"

This information is also available in the Management Console.

```
hdbadm@lu3412587:/usr/sap/HDB/HDB00 $ hdbcons
SAP HANA DB Management Client Console (type '\?' to get help for client commands)
Try to open connection to server process 'hdbindexserver' on system 'HDB',
instance '00'
SAP HANA DB Management Server Console (type 'help' to get help for server
commands)
Executable: hdbindexserver (PID: 2644)
[OK]
--
> mm list Pool/FuzzySearch -s
+----------------+-----+-----------------+-----+-----------------+------+-------+
|Name            |Used |Local size       |TUsed|Total size       |ACount|TACount|
+----------------+-----+--------+--------+-----+--------+--------+------+-------+
|Pool/FuzzySearch|14399|4989216B|4872.2KB|14399|4989216B|4872.2KB| 49177|  49177|
+----------------+-----+--------+--------+-----+--------+--------+------+-------+
[OK]
--
>
```

4.3.4.14.2 Sizing Information

This topic contains information about the sizing of fuzzy search indexes.

When setting `FUZZY SEARCH INDEX ON`, index structures are created in memory to make the fuzzy search faster. It is important to be aware of the additional memory usage when sizing an SAP HANA server. It is not possible to give exact numbers here, since the size of a fuzzy search index depends on the contents of the column and on the compression mode used for this column. A fuzzy search index on a VARCHAR column for example is large if each row contains a distinct value. The index is much smaller if the number of distinct values in this column is small. The size of the fuzzy search index can be determined with the M_FUZZY_SEARCH_INDEXES monitoring view.

More information: Monitoring Views [page 284].

Fuzzy Search Indexes for String Types

When creating a fuzzy search index on a string-type column (VARCHAR, NVARCHAR), the size of the fuzzy search index can be up to twice the memory size of the column itself.

Fuzzy Search Indexes for Text Types

The fuzzy search index on a text-type column (SHORTTEXT, TEXT or a hidden column created with the `CREATE FULLTEXT INDEX` statement) needs about 10 percent of the memory size of the column. Labels parameters

4.3.4.14.3 Activating the Trace in the SAP HANA Studio

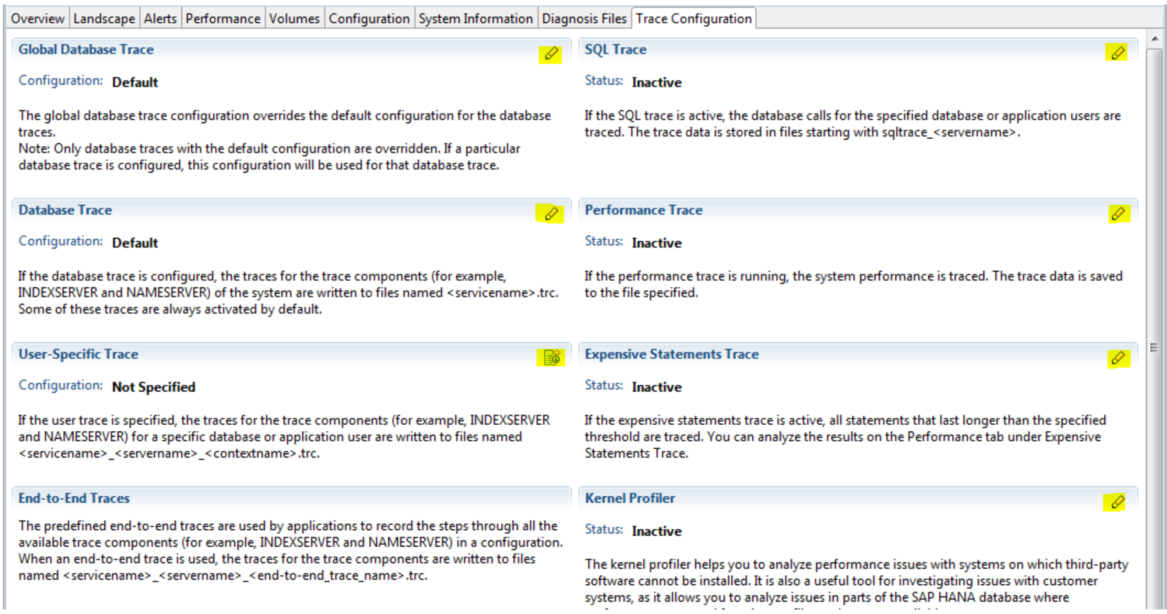
Context

Useful tracing and logging information for a fuzzy search is stored in the database traces. The traces are written for several services of the system (for example, INDEXSERVER and NAMESERVER). If the database trace is configured, trace information for the specified database will be written to files named as follows: <servicename>_<hostname>.<portnumber>.000.trc.

Tracing the output of a fuzzy search is available for the components `fuzzysearch`, `searchrulesexecution`, and `searchruleruntime`. The trace of the component `fuzzysearch` shows information about selected search options and the scoring and number of search results. You should use this trace for single fuzzy search requests. If the trace for `searchrulesexecution` and/or `searchruleruntime` is configured, information about the search rule configuration and search rule set results are tracked. The components `searchrulesexecution` and/or `searchruleruntime` should be used if a search rule set that combines multiple fuzzy search requests is used.

Procedure

1. In the SAP HANA studio, open the Administration perspective by double-clicking your system name in the navigation pane.
2. Select the tab *Trace Configuration* and choose the icon *Edit Configuration* at the right corner of the tracing section you want to configure.



3. Set the trace levels in the window *Database Trace*.

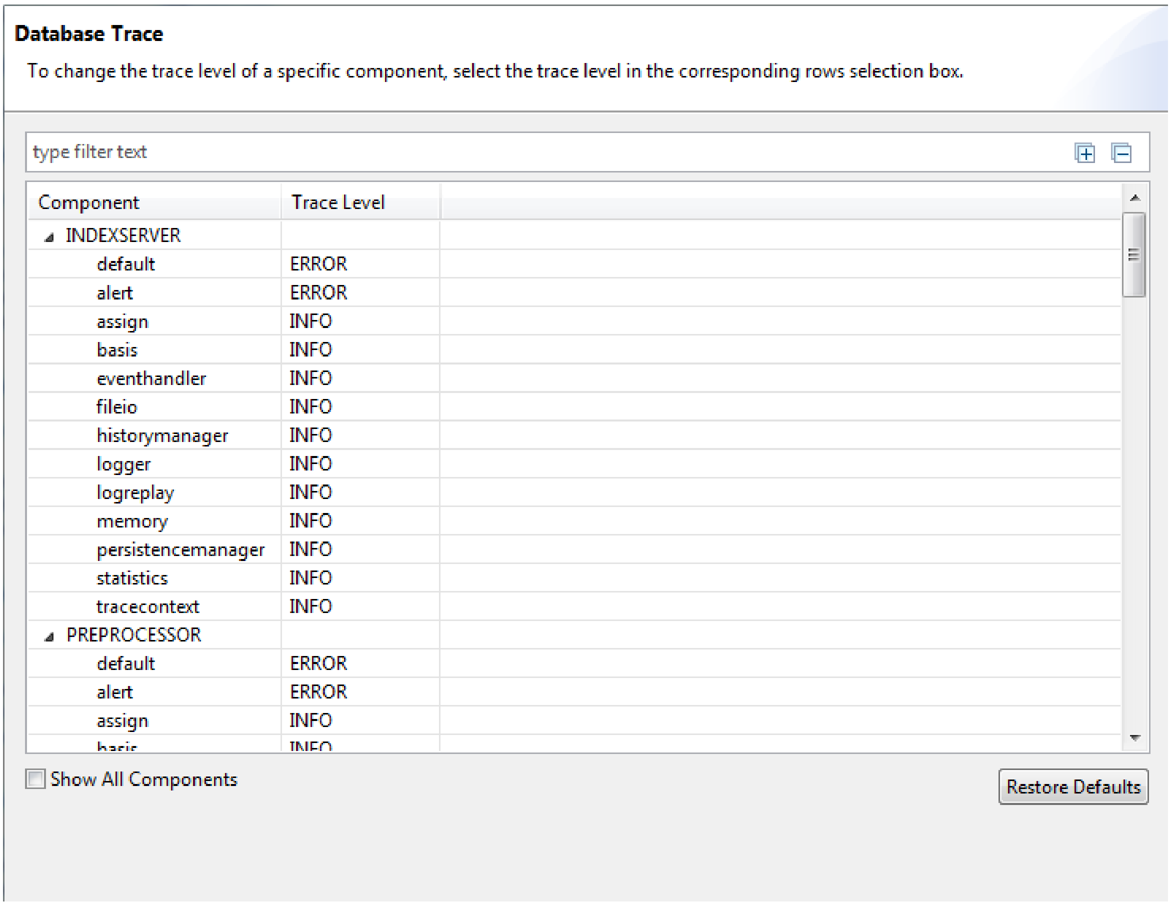
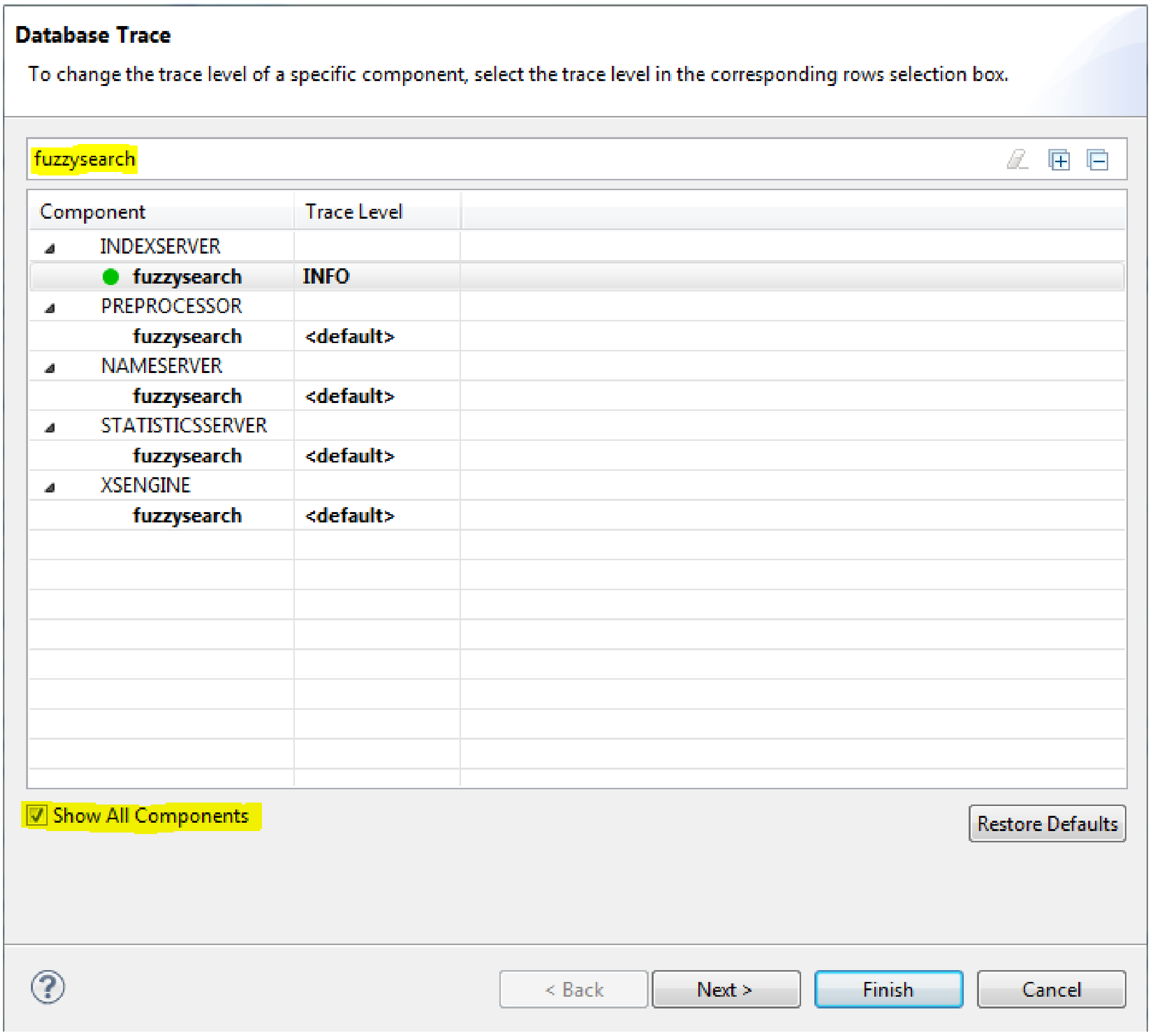


Table 175:

Trace Level	Description
NONE	Tracing is switched off. However, information about error situations is still recorded.
ERROR and FATAL	Information about errors is recorded.
WARNING	Information about potentially problematic situations is recorded.
INFO	Information about activity in the database is recorded.
DEBUG	Detailed information intended for debugging is recorded.

4. To configure the trace targets, choose *Show All Components* and filter for fuzzysearch or searchrules, as shown below.

   To get information about a fuzzy search, you only have to configure the INDEXSERVER. All other servers do not output relevant information regarding a fuzzy search.



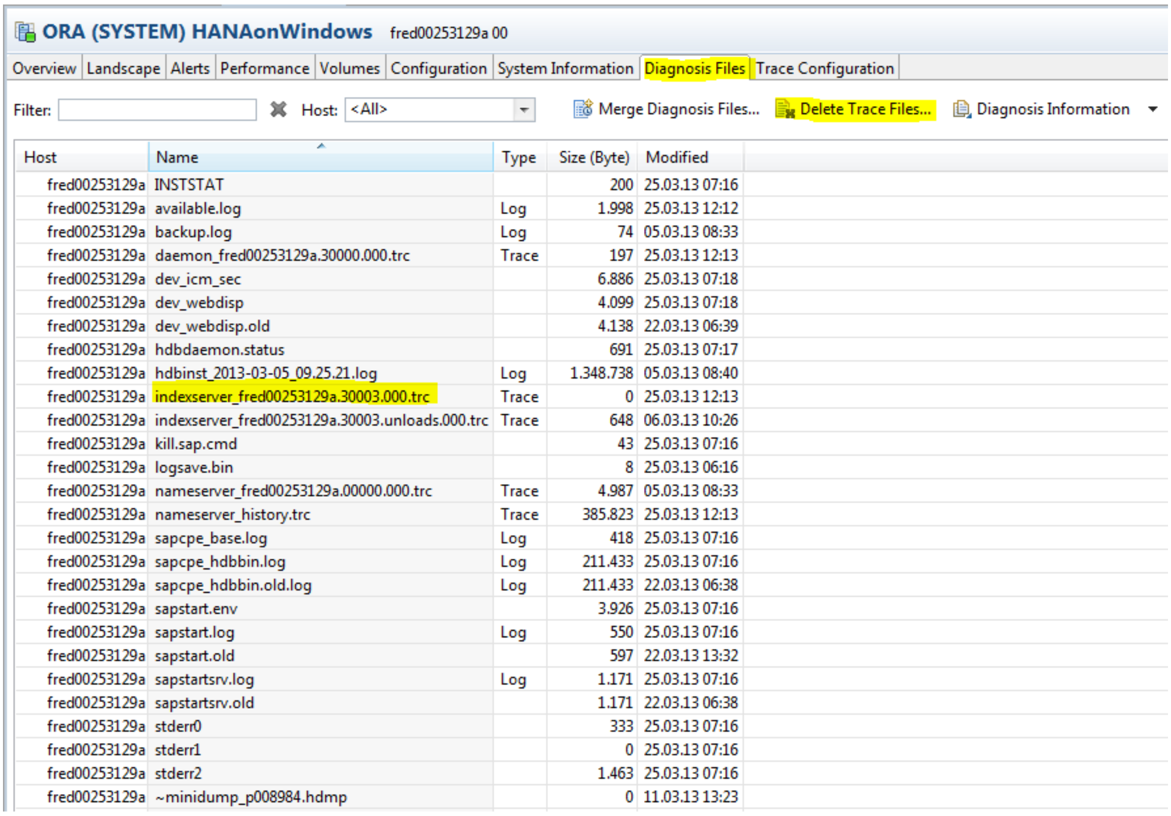
To get basic tracing data with information for every search, set the trace level to INFO.

To get detailed information for every search, set the trace level to DEBUG.

> **Note**
>
> DEBUG information cannot be read by end users and should be used for support issues only.

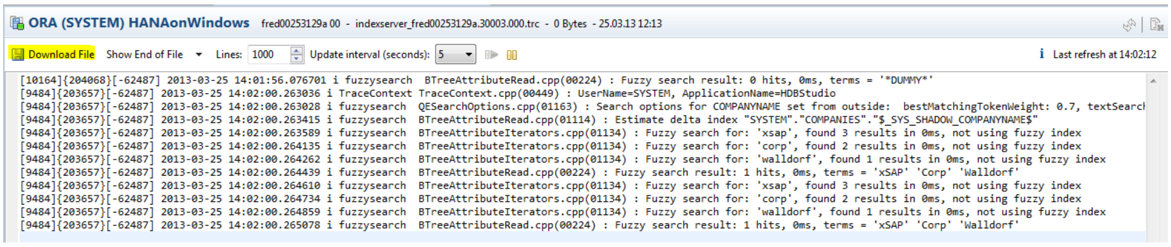
The default name for the trace file is `indexserver_$host_$port.000.trc`.

After executing a fuzzy search statement, you can find all trace files in the section *Diagnosis Files* of your administration perspective. To get only the information of the last request, you should delete older trace files before executing a fuzzy search. By clicking the column header of the traces table, you can reorder the trace files view, for example, by the date of the last modification.



To open a selected trace file, double-click it.

If you have problems with a fuzzy search, you can download trace files of your search and send them to SAP customer support. You can download an opened trace file by using the function Download File it the left corner of the trace file window.



4.3.4.15 Frequently Asked Questions

Why are there results with a score lower than the requested fuzzySimilarity?

In text fields, the parameter fuzzySimilarity sets the minimum similarity that a token has to match to be included in the search result. All other fuzzy search operations (for example, applying term mappings, considering stopwords, abbreviationSimilarity) can influence the score that you will see.

How many misspellings are allowed with a particular fuzzySimilarity?

This question is not easy to answer. The scoring algorithm is not linear to the number of misspellings; the position of the misspelling is also important. You can use the following example to familiarize yourself with it:

```
DROP TABLE test;
CREATE COLUMN TABLE test
(
id INTEGER PRIMARY KEY,
companyname SHORTTEXT(200) FUZZY SEARCH INDEX ON
);
INSERT INTO test VALUES ('1','abc');
INSERT INTO test VALUES ('2','abx');
INSERT INTO test VALUES ('3','xbc');
INSERT INTO test VALUES ('4','axc');
INSERT INTO test VALUES ('5','abcx');
INSERT INTO test VALUES ('6','xabc');
SELECT TO_DECIMAL(SCORE(),3,2) AS score, * FROM test
WHERE CONTAINS(companyname, 'abc',
FUZZY(0.5,'textSearch=compare,bestMatchingTokenWeight=1'))
ORDER BY score DESC, id;
```

Table 176:

SCORE	ID	COMPANYNAME
1	1	abc
0.89	5	abcx
0.82	2	abx
0.75	6	xabc
0.61	3	xbc
0.61	4	axc

```
SELECT TO_DECIMAL(SCORE(),3,2) AS score, * FROM test
WHERE CONTAINS(companyname, 'abcx',
FUZZY(0.5,'textSearch=compare,bestMatchingTokenWeight=1'))
ORDER BY score DESC, id;
```

Table 177:

SCORE	ID	COMPANYNAME
1	5	abcx
0.89	1	abc
0.88	6	xabc
0.75	3	abx
0.59	3	xbc
0.59	4	axc

How do I find out if the fuzzy search index is enabled for column x?

See The CONTAINS() Predicate [page 152].

How do I enable the fuzzy search index for a particular column?

See The CONTAINS() Predicate [page 152].

The additional data structures will increase the total memory footprint of the loaded table. In unfavorable cases the memory footprint of the column can double.

How can I see how much memory is used for a fuzzy search index?

See Monitoring Views [page 284].

Is the score between request and result always stable for TEXT columns?

It depends on how you look at the topic. The algorithm is indeed deterministic, but you need to take all parameters into account. Cases can be constructed where a small change in the fuzzySimilarity will change the rank between the same strings.

Why is this? The fuzzySimilarity is the minimum score that tokens need to reach to be considered for the result. If you use andThreshold or the keyword "OR" in your search, not all tokens have to reach the fuzzySimilarity to be part of the result. This can lead to a change in the total score if you change the fuzzySimilarity. Let us look at an example:

```
DROP TABLE companies;
CREATE COLUMN TABLE companies
(
id INTEGER PRIMARY KEY,
companyname SHORTTEXT(200)
);
INSERT INTO companies VALUES(1, 'aktien gesellschaft');
INSERT INTO companies VALUES(2, 'aktiv gesellschaft');
```

Important: The similarity between "aktien" and "aktiv" is 0.77.

If the fuzzySimilarity is lower than 0.77, the token scoring will be part of the result score. If the fuzzySimilarity is higher than 0.77, the token scoring will not be considered, so the total scoring will be lower.

```
SELECT TO_DECIMAL(SCORE(),3,2) AS score, id, companyname
FROM companies
WHERE CONTAINS(companyname, 'aktiv OR gesellschaft', FUZZY(0.75,
'textSearch=compare'))
ORDER BY score DESC, id;
```

Table 178:

SCORE	ID	COMPANYNAME
1	2	aktiv gesellschaft
0.89	1	aktien gesellschaft

```
SELECT TO_DECIMAL(SCORE(),3,2) AS score, id, companyname
FROM companies
WHERE CONTAINS(companyname, 'aktiv OR gesellschaft', FUZZY(0.80,
'textSearch=compare'))
ORDER BY score DESC, id;
```

Table 179:

SCORE	ID	COMPANYNAME
1	2	aktiv gesellschaft
0.71	1	aktien gesellschaft

5 Creating Search UIs With SAPUI5

SAPUI5 is a powerful, general-purpose user interface technology that is installed directly on the SAP HANA platform. SAPUI5 is tightly integrated into the SAP HANA programming model (Extended Application Services, OData). You can use SAPUI5 to develop flexible and domain-specific user interfaces that fully leverage the full-text search capabilities of SAP HANA.

To build rich, flexible search UIs for productive use, we recommended that you use the following technologies:

- Core Data Services (CDS) for defining data persistency and searchable views
- OData for web-based data access through CDS views
- SAPUI5 on OData

> **Note**
>
> Note the following if you choose SAPUI5 on OData: To get access to the additional facet information in the OData response, you need to insert code for binding on a table and register for the `dataReceived` event. See the following code example:

```
  //
// SAP UI5 example coding for binding and registering for the dataReceived event
//
resultTable.bindItems({
            path: "/" + settings.entitySet,
            template: new sap.m.ColumnListItem({
                cells: that.getDataBindingCells(properties, resultTable)
            }),
            parameters: {
                custom: {
                    facets: 'all'
                    // format: "json"
                }
            },
//
// Include the following code to get access to the additional facet information
in the OData response.
//
            events: {
                dataReceived: function(evt) {

console.log(evt.mParameters.data["@com.sap.vocabularies.Search.v1.Facets"]);
                }
            }
        });
```

6 SAP File Processing for SAP HANA

SAP File Processing is a component of SAP HANA that provides structured information from unstructured files. The rich set of HTTP APIs enables application programmers to integrate SAP File Processing features in client applications.

The guide for SAP File Processing for SAP HANA is a detailed description of SAP File Processing including the concepts, programming APIs and operation. System architects will be able to integrate SAP File Processing in business processes and system landscapes. Administrators will learn how to set up and operate File Processing and developers will find detailed information on programming APIs.

7 SAP HANA File Loader

The file loader is a set of HTTP services that you can use to develop your own applications to search in file contents.

The file loader package also contains a basic example application with monitoring and statistical information about the current file loader schedule.

> **Note**
>
> Technically, the file loader is an SAP HANA XS application that is shipped as a delivery unit. Accordingly the File Loader documentation is available in a separate document. It is linked under the related information section and describes the concept and the steps an application programmer has to follow to load binary files into SAP HANA with the file loader functionality. Afterwards, it is possible to run services like search on this content.

> **Note**
>
> If you start the development of new projects on SAP HANA 2.0, we recommend the use of the new HTTP services that are delivered with *SAP File Processing for SAP HANA*.
>
> SAP File Processing is a component of SAP HANA that provides structured information from unstructured files. The rich set of HTTP APIs enables application programmers to integrate SAP File Processing features in client applications.

Important Disclaimers and Legal Information

Coding Samples

Any software coding and/or code lines / strings ("Code") included in this documentation are only examples and are not intended to be used in a productive system environment. The Code is only intended to better explain and visualize the syntax and phrasing rules of certain coding. SAP does not warrant the correctness and completeness of the Code given herein, and SAP shall not be liable for errors or damages caused by the usage of the Code, unless damages were caused by SAP intentionally or by SAP's gross negligence.

Accessibility

The information contained in the SAP documentation represents SAP's current view of accessibility criteria as of the date of publication; it is in no way intended to be a binding guideline on how to ensure accessibility of software products. SAP in particular disclaims any liability in relation to this document. This disclaimer, however, does not apply in cases of willful misconduct or gross negligence of SAP. Furthermore, this document does not result in any direct or indirect contractual obligations of SAP.

Gender-Neutral Language

As far as possible, SAP documentation is gender neutral. Depending on the context, the reader is addressed directly with "you", or a gender-neutral noun (such as "sales person" or "working days") is used. If when referring to members of both sexes, however, the third-person singular cannot be avoided or a gender-neutral noun does not exist, SAP reserves the right to use the masculine form of the noun and pronoun. This is to ensure that the documentation remains comprehensible.

Internet Hyperlinks

The SAP documentation may contain hyperlinks to the Internet. These hyperlinks are intended to serve as a hint about where to find related information. SAP does not warrant the availability and correctness of this related information or the ability of this information to serve a particular purpose. SAP shall not be liable for any damages caused by the use of related information unless damages have been caused by SAP's gross negligence or willful misconduct. All links are categorized for transparency (see: http://help.sap.com/disclaimer).

go.sap.com/registration/contact.html

© 2017 SAP SE or an SAP affiliate company. All rights reserved.
No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company. The information contained herein may be changed without prior notice.
Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.
These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.
SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.
Please see http://www.sap.com/corporate-en/legal/copyright/index.epx for additional trademark information and notices.