SYBASE®
An SAP® Company

Unstructured Data Analytics in Sybase IQ

Sybase IQ 15.3

DOCUMENT ID: DC01268-01-1530-01
LAST REVISED: May 2011

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

Contents

Introduction to Unstructured Data Analytics in Sybase IQ 1
Audience 1
The Unstructured Data Analytics Option 1
Full Text Searching 2
Compatibility 2
Conformance to Standards 2
TEXT Indexes and Text Configuration Objects 3
TEXT Indexes 3
Comparison of WD and TEXT Indexes 3
Creating a TEXT Index Using Sybase Central 4
Creating a TEXT Index Using Interactive SQL 5
Guidelines for TEXT Index Size Estimation 5
TEXT Index Restrictions 6
Displaying a List of TEXT Indexes Using Sybase Central 6
Displaying a List of TEXT Indexes Using Interactive SQL 6
Editing a TEXT Index Using Sybase Central 6
Editing a TEXT Index Using Interactive SQL 7
Modifying the TEXT Index Location Using Sybase Central 7
Modifying the TEXT Index Location Using Interactive SQL 7
Dropping a TEXT Index Using Sybase Central 7
Dropping a TEXT Index Using Interactive SQL 8
TEXT Index Refresh 8
TEXT_DELETE_METHOD Database Option 8
NGRAM TEXT Index 9
Creating an NGRAM TEXT Index 9
Text Configuration Objects 10

Default Text Configuration Objects 10
Creating a Text Configuration Using Sybase Central 11
Creating a Text Configuration Using Interactive SQL 12
Text Configuration Object Settings 12
Displaying a List of Text Configurations Using Sybase Central 15
Displaying a List of Text Configurations Using Interactive SQL 15
Altering a Text Configuration Using Sybase Central 16
Altering a Text Configuration Using Interactive SQL 16
Modifying the Stoplist Using Sybase Central 16
Modifying the Stoplist Using Interactive SQL 17
Dropping a Text Configuration Using Sybase Central 17
Dropping a Text Configuration Using Interactive SQL 17
Text Configuration Object Examples 17
MAX_PREFIX_PER_CONTAINS_PHRASE Database Option 20
External Libraries 21
Pre-Filter and Term-Breaker External Libraries 21
External Library Restrictions 22
External Libraries on Multiplex Servers 22
Enable and Disable External Libraries on Startup 22
Unload External Libraries 23
Unstructured Data Queries 25
Full Text Search 25
Types of Full Text Searches 25
NGRAM TEXT Index Searches 34
Fuzzy Search 34
Non-fuzzy Search 35

Queries on LONG BINARY Columns 36
Queries on LONG VARCHAR Columns 37
CONTAINS Predicate Support 37
Performance Monitoring of LONG BINARY and LONG VARCHAR Columns 38
Stored Procedure Support 39
Term Management in a TEXT Index 39
sa_char_terms System Procedure 39
sa_nchar_terms System Procedure 40
sa_text_index_stats System Procedure 41
sa_text_index_vocab System Procedure 42
External Library Identification 43
sa_external_library_unload System Procedure ...43
sa_list_external_library Procedure 44
Large Object Data Compression 44
sp_iqsetcompression Procedure 45
sp_iqshowcompression Procedure 46
Information About Large Object Columns 46
Size of a LONG BINARY Column 47
Size of a LONG VARCHAR Column 47
Large Object Data Load and Unload 49
Large Object Data Exports 49
BFILE Function 49
Large Object Data Loads 51
Extended LOAD TABLE Syntax 51
Large Object Data Load Example 52
Control of Load Errors 52
Load of Large Object Data with Trailing Blanks ...53
Load of Large Object Data with Quotes 53
Truncation of Partial Multibyte Character Data 53
Load Support of Large Object Variables 54
Large Object Data Types 55
Large Object Data Types LONG BINARY and BLOB ...55
LONG BINARY Data Type Conversion 55

Large Object Data Types LONG VARCHAR and CLOB 56
LONG VARCHAR Data Type Conversion 57
Large Object Variables 58
Large Object Variable Data Type Conversion 58
Index Support of Large Object Columns 59
TEXT Index Support of Large Object Columns 59
WD Index Support of LONG VARCHAR (CLOB) Columns 59
SQL Statement Support 61
ALTER TEXT CONFIGURATION Statement 61
ALTER TEXT INDEX Statement 63
CREATE TEXT CONFIGURATION Statement 64
CREATE TEXT INDEX Statement 65
DROP TEXT CONFIGURATION Statement 66
DROP TEXT INDEX Statement 67
Function Support 69
Summary of Function Support of Large Object Data 69
BIT_LENGTH Function 70
BYTE_LENGTH Function 71
BYTE_LENGTH64 Function 71
BYTE_SUBSTR64 and BYTE_SUBSTR Functions 71
CHAR_LENGTH Function 72
CHAR_LENGTH64 Function 73
CHARINDEX Function 73
LOCATE Function 74
OCTET_LENGTH Function 75
PATINDEX Function 75
SUBSTRING Function 76
SUBSTRING64 Function 77
Aggregate Function Support of Large Object Columns 78
User-Defined Function Support of Large Object Columns 78
Error and Warning Messages 79

Error 1000195 79
Error 1000198 79
Error 1000332 80
Error 1001013 81
Error 1001051 81
Error 1001052 82
Error 1001053 82
Error 1001054 83
Warning 1001055 84
Warning 1001056 84
Error 1001057 85
Error 1001058 86
Error 1009189 86
Error 1012030 87
Index 89

Introduction to Unstructured Data Analytics in Sybase IQ

Learn about unstructured data analytics in Sybase® IQ and the compatibility and conformance to standards of Sybase IQ large object data.

Audience

This guide is for Sybase® IQ users who require reference material for working with unstructured data in Sybase IQ.

Learn about available syntax, parameters, functions, stored procedures, indexes, and options related to unstructured data analytics features in Sybase IQ. Use this guide as a reference together with the rest of the Sybase IQ documentation set to understand storage and retrieval of unstructured data within the Sybase IQ database.

The Unstructured Data Analytics Option

The Unstructured Data Analytics Option extends the capabilities of Sybase IQ to allow storage, retrieval, and full text searching of binary large objects (BLOBs) and character large objects (CLOBs) within the Sybase IQ database.

Note: Users must be specifically licensed to use the Unstructured Data Analytics functionality described in this product documentation.

As data volumes increase, the need to store large object (LOB) data in a relational database also increases. LOB data may be either:

- Unstructured – the database simply stores and retrieves the data, or
- Semistructured (for example, text) – the database supports the data structure and provides supporting functions (for example, string functions).

Typical LOB data sources include images, maps, documents (for example, PDF files, word processing files, and presentations), audio, video, and XML files. Sybase IQ can manage individual LOB objects containing gigabytes (GB), terabytes (TB), or even petabytes (PB) of data.

By allowing relational and unstructured data in the same location, Sybase IQ lets organizations access both types of data using the same application and the same interface. The full text search capability of Sybase IQ supports text archival applications (text analytics) in handling unstructured and semistructured data.

Full Text Searching

Full text searching uses **TEXT** indexes to search for terms and phrases in a database without having to scan table rows.

A **TEXT** index stores positional information for terms in the indexed column. Text configuration objects control the terms that are placed in a **TEXT** index when it is built or refreshed, and how a full text query is interpreted.

Using a **TEXT** index to find rows that contain a term or phrase is generally faster than scanning every row in the table.

Compatibility

SQL Anywhere® Server (SA) and Adaptive Server® Enterprise (ASE) store large text and binary objects.

SQL Anywhere can store large objects (up to a 2GB maximum length) in columns of data type `LONG VARCHAR` or `LONG BINARY`. The support of these data types by SQL Anywhere is SQL/2003 compliant. SQL Anywhere does not support the **BYTE_LENGTH64**, **BYTE_SUBSTR64**, **BFILE**, **BIT_LENGTH**, **OCTET_LENGTH**, **CHAR_LENGTH64**, and **SUBSTRING64** functions.

Adaptive Server Enterprise can store large text objects (up to a 2GB maximum length) and large binary objects (up to a 2GB maximum length) in columns of data type `TEXT` or `IMAGE`, respectively. The support of these data types by Adaptive Server Enterprise is an ANSI SQL Transact-SQL® extension.

A `LONG BINARY` column of a proxy table maps to a `VARBINARY(max)` column in a Microsoft SQL Server table.

Conformance to Standards

Sybase IQ `LONG BINARY` and `LONG VARCHAR` functionality conforms to the Core level of the ISO/ANSI SQL standard.

TEXT Indexes and Text Configuration Objects

Learn about working with **TEXT** indexes and text configuration objects.

A **TEXT** index stores positional information for terms in an indexed column. **TEXT** indexes are created using settings stored in a text configuration object. A text configuration object controls characteristics of **TEXT** index data, such as terms to ignore, and the minimum and maximum length of terms to include in the index.

TEXT Indexes

In a full text search, a **TEXT** index is searched, rather than table rows.

Before you can perform a full text search, you must create a **TEXT** index on the columns you want to search. A **TEXT** index stores positional information for terms in the indexed columns. Queries that use **TEXT** indexes are generally faster than those that must scan all the values in the table.

When you create a **TEXT** index, you can specify which text configuration object to use when creating and refreshing the **TEXT** index. A text configuration object contains settings that affect how an index is built. If you do not specify a text configuration object, the database server uses a default configuration object.

You can create **TEXT** indexes on these types of columns: `CHAR`, `VARCHAR`, and `LONG VARCHAR`, as well as `BINARY`, `VARBINARY`, and `LONG BINARY`. `BINARY`, `VARBINARY`, and `LONG BINARY` columns require that the **TEXT** index use a text configuration with an external prefilter library.

Comparison of WD and TEXT Indexes

A comparison of **WD** and **TEXT** indexes in terms of syntax and capability.

Table 1. WD versus TEXT Index

Feature	Supported by WD index?	Supported by TEXT index?
Conjunction of terms	Yes, expressed in the form: `tbl.col CONTAINS('great','white' ,'whale')`	Yes, expressed in the form: `CONTAINS(tbl.col,'great white whale')`

Feature	Supported by WD index?	Supported by TEXT index?
General boolean expressions	Yes, expressed in the form: `tbl.col CONTAINS ('great') AND ( tbl.col CONTAINS('white) OR tbl.col CONTAINS('whale') AND NOT tbl.col CONTAINS('ship'))`	Yes, expressed in the form: `CONTAINS(tbl.col, 'great AND ( white OR whale AND NOT ship )')`
Search for terms matching prefix	No	Yes, for example: `CONTAINS (tbl.col,'whale*')`
Acceleration of **LIKE** predicates	Yes, for example: `tbl.col LIKE 'whale%'`	No

Searches for terms in proximity	No	Yes, for example:
		`CONTAINS(tbl.col, 'white BEFORE whale')`
		`CONTAINS(tbl.col, 'whale NEAR white')`
		`CONTAINS(tbl.col, ' "white whale" ')`

Ordering of results based on search scoring	No	Yes

In **TEXT** index, searching for terms matching a prefix and searching for a **LIKE** expression have different semantics and may return very different results depending on the text configuration. The specification of minimum length, maximum length and a stoplist will govern the prefix processing but does not affect **LIKE** semantics.

Note: Meaning of boolean expressions will differ between **WD** index and **TEXT** index when term dropping occurs, because the effect of dropped terms in **TEXT** index processing has no equivalent in the **WD** index.

Creating a TEXT Index Using Sybase Central

Before you can perform a full text search, you must create a **TEXT** index on the columns you want to search.

A **TEXT** index stores positional information for terms in the indexed columns.

1. Connect to the database as a user with DBA or RESOURCE authority.

2. In the left pane, right-click the **Text Indexes** folder and select **Text Index** > **New**.
3. Select the table on which to create the **TEXT** index.
4. Type a name for the **TEXT** index. Click **Next**.
5. Select the column to include in the index. Click **Next**.
6. Select the text configuration object to use when processing the data for the **TEXT** index. Click **Next**.
7. For SQL Anywhere tables, in the Specify a Refresh Type dialog, click **Next**.

   For Sybase IQ tables, this option does not appear. The only supported refresh type is Immediate.
8. Select the dbspace where the **TEXT** index will be stored.
9. Click **Next**.
10. Type a comment describing the text configuration, and click **Finish**.

Creating a TEXT Index Using Interactive SQL

Before you can perform a full text search, you must create a **TEXT** index on the columns you want to search.

A **TEXT** index stores positional information for terms in the indexed columns.

1. Connect to the database as a user with DBA or RESOURCE authority.
2. Execute a **CREATE TEXT INDEX** statement.

This example creates a **TEXT** index, `myTxtIdx`, on the `CompanyName` column of the `Customers` table in the `iqdemo` database. The `default_char` text configuration object is used.

```
CREATE TEXT INDEX myTxtIdx ON Customers
    ( CompanyName ) CONFIGURATION default_char
```

Guidelines for TEXT Index Size Estimation

Formula estimates **TEXT** index main store size.

*Number of bytes = (15+L)*U + U*PAGESIZE * R + T*

where:

- L = average term length for the vocabulary
- U = number of unique terms in the vocabulary
- R = number of millions of documents
- T = total number of all terms in all documents

The temporary space required in bytes for the **TEXT** index is *(M+20)* T*, where:

- M = the maximum term length for the text configuration in bytes

Note: The temporary space required is subject to compressibility of the sort data.

TEXT Index Restrictions

Sybase IQ text configuration objects and **TEXT** indexes have limitations by design.

- Sybase IQ engine does not provide support for **TEXT** indexes spanning multiple columns.
- **TEXT** index manual refresh or automatic refresh options are not supported.
- **sp_iqrebuildindex** cannot be used to build **TEXT** indexes.
- You cannot create **TEXT** indexes within **BEGIN PARALLEL IQ…END PARALLEL IQ**.
- **NGRAM** term breaker is built on **TEXT** indexes, so use text configuration object settings to define whether to use an **NGRAM** or **GENERIC TEXT** index.
- **NGRAM TEXT** index search is mainly useful when words are misspelled. Sybase IQ does not support searches like synonyms and antonyms.

Displaying a List of TEXT Indexes Using Sybase Central

View a list of all the **TEXT** indexes in the database.

1. Connect to the database as a user with DBA or RESOURCE authority.
2. In the left pane, select the **Text Indexes** folder.

   A list of all **TEXT** indexes appears in the right pane.

Displaying a List of TEXT Indexes Using Interactive SQL

View a list of all the **TEXT** indexes in the database.

1. Connect to the database as a user with DBA or RESOURCE authority.
2. Execute a **SELECT** statement.

To list all Sybase IQ **TEXT** indexes:

```
SELECT * FROM sp_iqindex() WHERE index_type = 'TEXT';
```

To list all **TEXT** indexes, including those on catalog tables:

```
SELECT index_name, table_name, name FROM SYSIDX, SYSTEXTIDX,
SYSTABLE, SYSUSERS
WHERE SYSIDX.object_id=SYSTEXTIDX.index_id
AND SYSIDX.table_id=SYSTABLE.table_id
AND SYSTABLE.creator=SYSUSERS.uid;
```

Editing a TEXT Index Using Sybase Central

Change the settings for the **TEXT** index, including the dbspace and **TEXT** index name.

1. Connect to the database as a user with DBA or RESOURCE authority.
2. In the left pane, select the **Text Indexes** folder.

3. In the list of Text Indexes, right-click the object to modify and select **Properties**.
4. On the General tab, modify the settings as needed.
5. Click **OK**.

Editing a TEXT Index Using Interactive SQL

Change the settings for the **TEXT** index, including the dbspace and **TEXT** index name.

1. Connect to the database as a user with DBA or RESOURCE authority.
2. Execute an **ALTER TEXT INDEX** statement.

To rename the **TEXT** index `myTxtIdx` to `MyTextIndex`:

```
ALTER TEXT INDEX MyTxtIdx
    ON Customers
RENAME AS MyTextIndex;
```

Modifying the TEXT Index Location Using Sybase Central

Change the dbspace where the **TEXT** index is stored.

1. Connect to the database as a user with DBA or SPACE ADMIN authority or as table owner with CREATE privilege on dbspace.
2. In the left pane, select the **Text Indexes** folder.
3. In the list of Text Indexes, right-click the object to modify and select **Properties**.
4. On the General tab, select the dbspace from the drop-down list.
5. When the dbspace is updated, click **OK**.

Modifying the TEXT Index Location Using Interactive SQL

Change the dbspace where the **TEXT** index is stored.

1. Connect to the database as a user with DBA or SPACE ADMIN authority.
2. Execute an **ALTER TEXT INDEX** statement with the **MOVE TO** clause.

To move the **TEXT** index `MyTextIndex` to a dbspace named `tispace`:

```
ALTER TEXT INDEX MyTextIndex ON
    GROUPO.customers MOVE TO tispace;
```

Dropping a TEXT Index Using Sybase Central

Drop a **TEXT** index from the database.

1. Connect to the database as a user with DBA or RESOURCE authority.
2. In the left pane, select the **Text Indexes** folder.
3. In the list of Text Indexes, right-click the object to modify and select **Delete**.

4. In the confirmation dialog, click **Yes**.

Dropping a TEXT Index Using Interactive SQL

Drop a **TEXT** index from the database.

1. Connect to the database as a user with DBA or RESOURCE authority.
2. Execute a **DROP TEXT INDEX** statement.

To drop the `MyTextIndex` **TEXT** index:

```
DROP TEXT INDEX MyTextIndex ON Customers;
```

TEXT Index Refresh

The only supported refresh type for **TEXT** indexes on Sybase IQ tables is Immediate Refresh, which occurs when data in the underlying table changes.

Immediate-refresh **TEXT** indexes on Sybase IQ tables support isolation level 3. They are populated at creation time and every time the data in the column is changed using an **INSERT**, **UPDATE**, or **DELETE** statement. An exclusive lock is held on the table during the initial refresh.

TEXT_DELETE_METHOD Database Option

Specifies the algorithm used during a delete in a **TEXT** index.

Allowed Values
0 – 2

0 – the delete method is selected by the cost model.

1 – forces small method for deletion. Small method is useful when the number of rows being deleted is a very small percentage of the total number of rows in the table. Small delete can randomly access the index, causing cache thrashing with large data sets.

2 – forces large method for deletion. This algorithm scans the entire index searching for rows to delete. Large method is useful when the number of rows being deleted is a high percentage of the total number of rows in the table.

Default
0

Scope
DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description
`TEXT_DELETE_METHOD` specifies the algorithm used during a delete operation in a **TEXT** index. When this option is not set or is set to 0, the delete method is selected by the cost model.

The cost model considers the CPU-related costs as well as I/O-related costs in selecting the appropriate delete algorithm. The cost model takes into account:

- Rows deleted
- Index size
- Width of index data type
- Cardinality of index data
- Available temporary cache
- Machine-related I/O and CPU characteristics
- Available CPUs and threads

See *Performance and Tuning Guide > Optimizing Queries and Deletions > Optimizing delete operations*.

Example

To force the large method for deletion from a **TEXT** index:

```
SET TEMPORARY OPTION TEXT_DELETE_METHOD = 2
```

NGRAM TEXT Index

NGRAM TEXT index stores the text in the column by breaking the text into n-grams of text value N, where N is the value given by a user.

You can perform a search over an **NGRAM TEXT** index by matching the n-grams of the text value in the **CONTAINS** clause of the query against the stored n-grams in the index.

NGRAM TEXT index accommodates fuzzy searching capability over the text for both European and non-European languages. For more information on fuzzy searching, see *Unstructured Data Queries > NGRAM TEXT Index Searches > Fuzzy Search*.

Note: **NGRAM TEXT** index search is mainly useful when words are misspelled. Sybase IQ does not support searches like synonyms and antonyms.

NGRAM term breaker is built on **TEXT** indexes, so use text configuration object settings to define whether to use an **NGRAM** or **GENERIC TEXT** index.

For more information on text configuration object settings, see *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Usage > Querying and Modifying Data > Querying Data > Text configuration objects > Text configuration object settings*.

Creating an NGRAM TEXT Index

Reference to information about creating an NGRAM TEXT index.

For information on how to create a **NGRAM TEXT** index, see *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Usage > Querying and Modifying Data > Querying Data > Types of full text searches > Tutorial: Performing a fuzzy full text search*.

Text Configuration Objects

Text configuration objects control the terms that are placed in a **TEXT** index when it is built or refreshed, and how a full text query is interpreted.

When the database server creates or refreshes a **TEXT** index, it uses the settings for the text configuration object specified when the **TEXT** index was created. If a text configuration object is not specified, the database server chooses one of the default text configuration objects, based on the type of data in the columns being indexed. In a Sybase IQ database, the `default_char` text configuration object is always used.

Text configuration objects specify which prefilter library and which term breaker are used to generate terms from the documents to be indexed. They specify the minimum and maximum length of terms to be stored within the **TEXT** index, along with the list of terms that should not be included. Text configuration objects consist of these parameters:

- Document pre-filter – removes unnecessary information, such as formatting and images, from the document. The filtered document is then picked up by other modules for further processing. The document pre-filter is provided by a third-party vendor.
- Document term-breaker – breaks the incoming byte stream into terms separated by term separators or according to specified rules. The document term-breaker is provided by the server or a third-party vendor.
- Stoplist processor – specifies the list of terms to be ignored while building the **TEXT** index.

Default Text Configuration Objects

Sybase IQ provides default text configuration objects.

The default text configuration object `default_char` is used with non-NCHAR data. This configuration is created the first time you create a text configuration object or **TEXT** index.

The text configuration object `default_nchar` is supported for use with NCHAR for **TEXT** indexes on **IN SYSTEM** tables; you cannot use `default_nchar` text configuration for **TEXT** indexes on Sybase IQ tables.

The table "Default text configuration settings" shows the default settings for `default_char` and `default_nchar`, which are best suited for most character-based languages. Sybase strongly recommends that you do not change the settings in the default text configuration objects.

Table 2. Default text configuration settings

Setting	Installed value
TERM BREAKER	GENERIC

Setting	Installed value
MINIMUM TERM LENGTH	1
MAXIMUM TERM LENGTH	20
STOPLIST	(empty)

If you delete a default text configuration object, it is automatically re-created with default values the next time you create a **TEXT** index or text configuration object.

Creating a Text Configuration Using Sybase Central

Create a text configuration to specify how **TEXT** indexes dependent on the text configuration process handle terms within the data.

1. Connect to the database as a user with DBA or RESOURCE authority.
2. In the left pane, right-click the **Text Configurations Objects** folder and select **New > Text Configuration Object**.
3. Type a name for the text configuration.
4. Select the owner of the text configuration.
5. Select the type of database collation for the text configuration. Click **Next**.

   Note: Text configurations with `NCHAR` collation are not supported by Sybase IQ **TEXT** indexes.
6. Select the **Generic term-breaker** algorithm.
7. Enter the minimum and maximum term length.
8. If using an external term breaker library, select **Use an external term breaker** and specify the external term breaker function and library.

   Specify the function and library in the form `function-name@library-file-name`.
9. Click **Next**.
10. If using an external prefilter library, select **Use an external prefilter** and specify the external prefilter function and library.

    Specify the function and library in the form `function-name@library-file-name`.
11. Add any terms to ignore when building a **TEXT** index with this text configuration to the Stoplist. Separate terms with a space.

    Terms in this list are also ignored in a query.
12. Click **Next**.
13. Type a comment describing the text configuration, and click **Finish**.

Creating a Text Configuration Using Interactive SQL

Create a text configuration to specify how **TEXT** indexes dependent on the text configuration process handle terms within the data.

1. Connect to the database as a user with DBA or RESOURCE authority.
2. Execute a **CREATE TEXT CONFIGURATION** statement.

To create a text configuration object called `myTxtConfig` using the `default_char` text configuration object as a template:

```
CREATE TEXT CONFIGURATION myTxtConfig FROM default_char;
```

Text Configuration Object Settings

Learn about text configuration object settings, how they affect what is indexed, and how a full text search query is interpreted.

For examples of text configuration objects and their impact on **TEXT** indexes and full text searching, see *Text Configuration Object Setting Interpretations*.

See also

- *Text Configuration Object Setting Interpretations* on page 17

Term Breaker Algorithm (TERM BREAKER)

The **TERM BREAKER** setting specifies the algorithm to use for breaking strings into terms.

Sybase IQ supports **GENERIC** (the default) or **NGRAM** for storing terms.

Note: NGRAM term breakers store n-grams. An n-gram is a group of characters of length *n* where *n* is the value of **MAXIMUM TERM LENGTH**.

Regardless of the term breaker you specify, the database server records in the **TEXT** index the original positional information for terms when they are inserted into the **TEXT** index. In the case of n-grams, the positional information of the n-grams is stored, not the positional information for the original terms.

Table 3. TERM BREAKER impact

To TEXT index	To query terms
GENERIC TEXT index – when building a **GENERIC TEXT** index (the default), groups of alphanumeric characters appearing between non-alphanumeric characters are processed as terms by the database server. After the terms have been defined, terms that exceed the term length settings, and terms found in the stoplist, are counted but not inserted in the **TEXT** index. Performance on **GENERIC TEXT** indexes can be faster than **NGRAM TEXT** indexes. However, you cannot perform fuzzy searches on **GENERIC TEXT** indexes.	GENERIC TEXT index – when querying a **GENERIC TEXT** index, terms in the query string are processed in the same manner as if they were being indexed. Matching is performed by comparing query terms to terms in the **TEXT** index.
NGRAM TEXT index – when building an **NGRAM TEXT** index, the database server treats as a term any group of alphanumeric characters between non-alphanumeric characters. Once the terms are defined, the database server breaks the terms into n-grams. In doing so, terms shorter than n, and n-grams that are in the stoplist, are discarded. For example, for an **NGRAM TEXT** index with **MAXIMUM TERM LENGTH** 3, the string 'my red table' is represented in the **TEXT** index as these n-grams: red tab abl ble.	NGRAM TEXT index – when querying an **NGRAM TEXT** index, terms in the query string are processed in the same manner as if they were being indexed. Matching is performed by comparing n-grams from the query terms to n-grams from the indexed terms.

Minimum Term Length Setting (MINIMUM TERM LENGTH)

The **MINIMUM TERM LENGTH** setting specifies the minimum length, in characters, for terms inserted in the index or searched for in a full text query.

MINIMUM TERM LENGTH is not relevant for **NGRAM TEXT** indexes.

MINIMUM TERM LENGTH has special implications on prefix searching. The value of **MINIMUM TERM LENGTH** must be greater than 0. If you set it higher than **MAXIMUM TERM LENGTH**, then **MAXIMUM TERM LENGTH** is automatically adjusted to be equal to **MINIMUM TERM LENGTH**.

The default for **MINIMUM TERM LENGTH** is taken from the setting in the default text configuration object, which is typically 1.

Table 4. MINIMUM TERM LENGTH impact

To TEXT index	To query terms
GENERIC TEXT index – for **GENERIC TEXT** indexes, the **TEXT** index will not contain words shorter than **MINIMUM TERM LENGTH**.	GENERIC TEXT index – when querying a **GENERIC TEXT** index, query terms shorter than **MINIMUM TERM LENGTH** are ignored because they cannot exist in the **TEXT** index.
NGRAM TEXT index – for **NGRAM TEXT** indexes, this setting is ignored.	NGRAM TEXT index – the **MINIMUM TERM LENGTH** setting has no impact on full text queries on **NGRAM TEXT** indexes.

Maximum Term Length Setting (MAXIMUM TERM LENGTH)

The **MAXIMUM TERM LENGTH** setting specifies the maximum length, in characters, for terms inserted in the index or searched for in a full text query.

The **MAXIMUM TERM LENGTH** setting is used differently, depending on the term breaker algorithm. The value of **MAXIMUM TERM LENGTH** must be less than or equal to 60. If you set **MAXIMUM TERM LENGTH** lower than the **MINIMUM TERM LENGTH**, then **MINIMUM TERM LENGTH** is automatically adjusted to be equal to **MAXIMUM TERM LENGTH**.

The default for this setting is taken from the setting in the default text configuration object, which is typically 20.

Table 5. MAXIMUM TERM LENGTH impact

To TEXT index	To query terms
GENERIC TEXT index – for **GENERIC TEXT** indexes, **MAXIMUM TERM LENGTH** specifies the maximum length, in characters, for terms inserted in the **TEXT** index.	GENERIC TEXT index – for **GENERIC TEXT** indexes, query terms longer than **MAXIMUM TERM LENGTH** are ignored because they cannot exist in the **TEXT** index.
NGRAM TEXT index – for **NGRAM TEXT** indexes, **MAXIMUM TERM LENGTH** determines the length of the n-grams that terms are broken into. An appropriate choice of length for **MAXIMUM TERM LENGTH** depends on the language. Typical values are 4 or 5 characters for English, and 2 or 3 characters for Chinese.	NGRAM TEXT index – for **NGRAM TEXT** indexes, query terms are broken into n-grams of length n, where n is the same as **MAXIMUM TERM LENGTH**. The database server uses the n-grams to search the **TEXT** index. Terms shorter than **MAXIMUM TERM LENGTH** are ignored because they do not match the n-grams in the **TEXT** index.

Stoplist Setting (STOPLIST)

The stoplist setting specifies terms that are not indexed.

The default for the stoplist setting is taken from the setting in the default text configuration object, which typically has an empty stoplist.

Table 6. STOPLIST impact

To TEXT index	To query terms
GENERIC TEXT index – for **GENERIC TEXT** indexes, terms that are in the stoplist are not inserted into the **TEXT** index.	GENERIC TEXT index – for **GENERIC TEXT** indexes, query terms that are in the stoplist are ignored because they cannot exist in the **TEXT** index.
NGRAM TEXT index – for **NGRAM TEXT** indexes, the **TEXT** index does not contain the n-grams formed from the terms in the stoplist.	NGRAM TEXT index – terms in the stoplist are broken into n-grams and the n-grams are used for the stoplist. Likewise, query terms are broken into n-grams and any that match n-grams in the stoplist are dropped because they cannot exist in the **TEXT** index.

Consider carefully whether to put terms in to your stoplist. In particular, do not include words that have non-alphanumeric characters in them such as apostrophes or dashes. These characters act as term breakers. For example, the word you'll (which must be specified as 'you'll') is broken into you and ll and stored in the stoplist as these two terms. Subsequent full text searches for 'you' or 'they'll' are negatively impacted.

Stoplists in **NGRAM TEXT** indexes can cause unexpected results because the stoplist that is stored is actually in n-gram form, not the actual stoplist terms you specified. For example, in an **NGRAM TEXT** index where **MAXIMUM TERM LENGTH** is 3, if you specify **STOPLIST** 'there', these n-grams are stored as the stoplist: the her ere. This impacts the ability to query for any terms that contain the n-grams the, her, and ere.

Displaying a List of Text Configurations Using Sybase Central

View a list of all the text configurations in the database.

1. Connect to the database as a user with DBA or RESOURCE authority.
2. In the left pane, select **Text Configurations Objects**.

   A list of all text configurations appears in the right pane.

Displaying a List of Text Configurations Using Interactive SQL

View a list of all the text configurations in the database.

1. Connect to the database as a user with DBA or RESOURCE authority.
2. Execute a **SELECT** statement.

To list all text configuration objects:

```
SELECT * FROM SYSTEXTCONFIG;
```

Altering a Text Configuration Using Sybase Central

Change the settings of the text configuration object, including the dbspace and permitted term lengths range.

You can alter only text configuration objects that are not being used by a **TEXT** index.

1. Connect to the database as a user with DBA or RESOURCE authority.
2. In the left pane, select **Text Configurations Objects**.
3. In the list of Text Configurations, right-click the object to modify and select **Properties**.
4. Switch to the **Settings** tab, and modify the settings as needed.
5. Click **OK**.

Altering a Text Configuration Using Interactive SQL

Change the settings of the text configuration object, including the dbspace and permitted term lengths range.

You can alter only text configuration objects that are not being used by a **TEXT** index.

1. Connect to the database as a user with DBA or RESOURCE authority, or as the owner of the text configuration object.
2. Execute an **ALTER TEXT CONFIGURATION** statement.

To alter the minimum term length for the `myTxtConfig` text configuration object:

```
ALTER TEXT CONFIGURATION myTxtConfig
    MINIMUM TERM LENGTH 2;
```

Modifying the Stoplist Using Sybase Central

Modify the stoplist, which contains a list of terms to ignore when building a **TEXT** index with this text configuration.

You can alter only text configuration objects that are not being used by a **TEXT** index.

1. Connect to the database as a user with DBA or RESOURCE authority.
2. In the left pane, select **Text Configurations Objects**.
3. In the list of Text Configurations, right-click the object to modify and select **Properties**.
4. Switch to the **Stoplist** tab, and modify the stoplist words as needed. Use a space to separate the terms.
5. To sort the list of stoplist terms alphabetically and show them in a list, click **Sort Terms**.
6. When the stoplist is updated, click **OK**.

Modifying the Stoplist Using Interactive SQL

Modify the stoplist, which contains a list of terms to ignore when building a **TEXT** index with this text configuration.

You can alter only text configuration objects that are not being used by a **TEXT** index.

1. Connect to the database as a user with DBA or RESOURCE authority.
2. Execute an **ALTER TEXT CONFIGURATION** statement with the **STOPLIST** clause.

To add a stoplist to the `myTxtConfig` configuration object:

```
ALTER TEXT CONFIGURATION myTxtConfig
    STOPLIST 'because about therefore only';
```

Dropping a Text Configuration Using Sybase Central

Remove an unnecessary text configuration from the database.

Only text configurations that are not being used by a **TEXT** index can be dropped.

1. Connect to the database as a user with DBA or RESOURCE authority.
2. In the left pane, select **Text Configurations Objects**.
3. In the list of Text Configurations, right-click the object to modify and select **Delete**.
4. In the confirmation dialog, click **Yes**.

Dropping a Text Configuration Using Interactive SQL

Remove an unnecessary text configuration from the database.

Only text configurations that are not being used by a **TEXT** index can be dropped.

1. Connect to the database as a user with DBA or RESOURCE authority.
2. Execute a **DROP TEXT CONFIGURATION** statement.

To drop the text configuration object `myTxtConfig`:

```
DROP TEXT CONFIGURATION myTxtConfig;
```

Text Configuration Object Examples

Review the samples to understand how text configuration settings impact the **TEXT** index, and how the index is interpreted.

Text Configuration Object Setting Interpretations

Examples that show the settings for different text configuration objects, how the settings impact what is indexed, and how a full text query string is interpreted.

All the examples use the string 'I'm not sure I understand'.

Table 7. Text configuration setting interpretations

Configuration settings	Terms that are indexed	Query interpretation
TERM BREAKER: GENERIC MINIMUM TERM LENGTH: 1 MAXIMUM TERM LENGTH: 20 STOPLIST: "	`I m not sure I understand`	`'("I m" AND not sure) AND I AND understand'`
TERM BREAKER: GENERIC MINIMUM TERM LENGTH: 2 MAXIMUM TERM LENGTH: 20 STOPLIST: 'not and'	`sure understand`	`'understand'`
TERM BREAKER: GENERIC MINIMUM TERM LENGTH: 1 MAXIMUM TERM LENGTH: 20 STOPLIST: 'not and'	`I m sure I understand`	`'"I m" AND sure AND I AND understand'`

Text Configuration Object CONTAINS Query String Interpretations

Examples of how the settings of the text configuration object strings are interpreted for CONTAINS queries.

The parenthetical numbers in the Interpreted string column in the table "CONTAINS string interpretations" reflect the position information stored for each term. The numbers are for illustration purposes in the documentation. The actual stored terms do not include the parenthetical numbers.

Note: The maximum number of positions for a text document is 4294967295.

The interpretations in this table are only for **CONTAINS** queries. When data is parsed, **AND**, **NOT**, and **NEAR** are considered regular tokens; symbols like *****, **I**, and others are dropped as they are not alphanumeric.

Table 8. CONTAINS string interpretations

Configuration settings	String	Interpreted string
TERM BREAKER: GENERIC MINIMUM TERM LENGTH: 3 MAXIMUM TERM LENGTH: 20	`'w*'`	`'"w*(1)"'`
	`'we*'`	`'"we*(1)"'`
	`'wea*'`	`'"wea*(1)"'`
	`'we* -the'`	`'"we*(1)" -"the(1)"'`
	`'for* | wonderl*'`	`'"for*(1)" | "wonderl*(1)"'`
	`'wonderlandwonderlandwonderland*'`	`''`
	`'"tr* weather"'`	`'"weather(1)"'`
	`'"tr* the weather"'`	`'"the(1) weather(2)"'`
	`'"wonderlandwonderlandwonderland* wonderland"'`	`'"wonderland(1)"'`
	`'"wonderlandwonderlandwonderland* weather"'`	`'"weather(1)"'`
	`'"the_wonderlandwonderlandwonderland* weather"'`	`'"the(1) weather(3)"'`
	`'the_wonderlandwonderlandwonderland* weather'`	`'"the(1)" & "weather(1)"'`
	`'"light_a* the end" & tunnel'`	`'"light(1) the(3) end(4)" & "tunnel(1)"'`
	`light_b* the end" & tunnel'`	`'"light(1) the(3) end(4)" & "tunnel(1)"'`
	`'"light_at_b* end"'`	`'"light(1) end(4)"'`
	`'and-te*'`	`'"and(1) te*(2)"'`

Configuration settings	String	Interpreted string
	`'a_long_and_t* & jour-ney'`	`'"long(2) and(3) t*(4)" & "jour-ney(1)"'`

MAX_PREFIX_PER_CONTAINS_PHRASE Database Option

Specifies the number of prefix terms allowed in a text search expression.

Allowed Values
0 – 300

0 – no limit for prefix terms in search phrase

300 – upper limit (this is the overall limit for total number of terms allowed in a phrase)

Default
1

Scope
DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description
MAX_PREFIX_PER_CONTAINS_PHRASE specifies the threshold used to disallow more than one prefix in an expression for a text search.

When this option is set to 0, any number is allowed. Sybase IQ detects and reports an error, if the query has any **CONTAINS** expressions with a phrase having more prefix terms than specified by this option.

Examples
With the default MAX_PREFIX_PER_CONTAINS_PHRASE setting:

```
SET MAX_PREFIX_PER_CONTAINS_PHRASE = 1
```

this **CONTAINS** clause is valid:

```
SELECT ch1 FROM tab1
WHERE CONTAINS(ch1, '"concord bed* in mass"')
```

With the default MAX_PREFIX_PER_CONTAINS_PHRASE setting of 1, this **CONTAINS** clause returns a syntax error:

```
SELECT ch1 FROM tab1
WHERE CONTAINS (ch1, '"con* bed* in mass"')
```

When MAX_PREFIX_PER_CONTAINS_PHRASE is set equal to 0 (no limit) or 2, this **CONTAINS** clause is valid.

External Libraries

Learn about using external libraries to supply prefiltering and term breaking for documents.

Pre-Filter and Term-Breaker External Libraries

Sybase IQ can use external pre-filter and term-breaker libraries written in C or C++ to prefilter and tokenize the documents during index creation or query processing. These libraries can be dynamically loaded into the process space of the database server.

Note: External pre-filter and term-breaker libraries must be provided by a Sybase-certified partner. For information on certified vendor solutions, see the *Partner Certification Reports web site* and then filter the certification reports to show Sybase IQ certifications

The external dynamically loadable pre-filter and term-breaker libraries are specified in the text configuration, and need to be loaded by the database server. Each library contains an exported symbol that implements the external function specified in the text configuration object. This function returns a set of function descriptors that are used by the caller to perform the necessary tasks.

The external pre-filter and term-breaker libraries are loaded by the database server with the first **CREATE TEXT INDEX** request, when a query for a given column is received that requires the library to be loaded, or when the **TEXT** index needs to be updated.

The libraries are not loaded when an **ALTER TEXT CONFIGURATION** call is made, nor are they automatically unloaded when a **DROP TEXT CONFIGURATION** call is made. The external pre-filter and term-breaker libraries are not loaded if the server is started with the startup option to disallow the loading of external libraries.

Because these external C/C++ libraries involve the loading of non-server library code into the process space of the server, there are potential risks to data integrity, data security, and server robustness from poorly or maliciously written functions. To manage these risks, each Sybase IQ server can explicitly enable or disable this functionality. See *Enable and Disable External Libraries on Startup*.

The `ISYSTEXTCONFIG` system table stores information about the external libraries associated with a text configuration object. See *Reference: Building Blocks, Tables, and Procedures > System Tables and Views > System Views > SYSTEXTCONFIG System View*.

See also

- *Enable and Disable External Libraries on Startup* on page 22

External Library Restrictions

Sybase IQ text configuration objects and **TEXT** indexes using external libraries have limitations by design.

- For **TEXT** indexes on binary columns, you must use external libraries provided by external vendors for document conversion. Sybase IQ does not implicitly convert documents stored in binary columns.
- N-gram based text searches are not supported if an external term-breaker is used to tokenize the document.
- You cannot use external libraries to create **TEXT** indexes on SQL Anywhere tables; doing so results in an error.

External Libraries on Multiplex Servers

All multiplex servers must have access to the pre-filter and term-breaker external libraries.

Users must ensure that each external pre-filter and term-breaker library is copied to the machine hosting a multiplex server and placed in a location where the server is able to load the library.

Each multiplex server works independently of other servers when calling the external pre-filter and term-breaker. Each process space can have the external libraries loaded and perform its own executions. It is assumed that the implementation of the pre-filter and term-breaker functions is the same on each server and will return the same result.

Unloading an external library from one server process space does not unload the library from other server process spaces.

Enable and Disable External Libraries on Startup

Sybase IQ provides the **-sf** startup switch to enable or disable loading of external third-party libraries.

You can specify this switch in either the server startup command line or the server configuration file.

To enable the loading of external third-party libraries:

```
-sf -external_library_full_text
```

To disable the loading of external third-party libraries:

```
-sf external_library_full_text
```

To view a list of the libraries currently loaded in the server, use the **sa_list_external_library** stored procedure.

Unload External Libraries

Use the system procedure **dbo.sa_external_library_unload** to unload an external library when the library is not in use.

dbo.sa_external_library_unload takes one optional parameter, a `LONG VARCHAR`. The parameter specifies the name of the library to unload. If you do not specify a parameter, all external libraries that are not in use are unloaded.

Unload an external function library:

```
call sa_external_library_unload('library.dll')
```

Unstructured Data Queries

Learn about querying large object data, including the full text search capability that handles unstructured and semistructured data.

Full Text Search

Full text search uses **TEXT** indexes to quickly find all instances of a term (word) in a database without having to scan table rows.

TEXT indexes store positional information for terms in the indexed columns. Using a **TEXT** index to find rows that contain a term is faster than scanning every row in the table.

Full text search uses the **CONTAINS** search condition, which differs from searching using predicates such as **LIKE**, **REGEXP**, and **SIMILAR TO**, because the matching is term-based and not pattern-based.

String comparisons in full text search use all the normal collation settings for the database. For example, you configure the database to be case-insensitive, then full text searches are also case-insensitive.

See *CONTAINS Conditions* and *Reference: Building Blocks, Tables, and Procedures > SQL Language Elements > Search Conditions > CONTAINS Conditions*.

See also

- *CONTAINS Conditions* on page 26

Types of Full Text Searches

Using full text search, you can search for terms, prefixes, or phrases (sequences of terms). You can also combine multiple terms, phrases, or prefixes into Boolean expressions, or use proximity searches to require that expressions appear near to each other.

Perform a full text search using a **CONTAINS** clause in either a **WHERE** clause, or a **FROM** clause of a **SELECT** statement.

Note: The SQL Anywhere documentation provides full text search examples. Not all of these examples apply to Sybase IQ. For example, Sybase IQ does not support text searches that are part of the IF search condition.

See *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Usage > Querying and Modifying Data > Querying data > Types of full text searches*.

FROM Clause

Specifies the database tables or views involved in a **SELECT** statement.

Syntax

```
... FROM table-expression [, …]
```

Parameters

table-expression:{ *table-spec* | *table-expression join-type table-spec* [**ON** *condition*] | (*table-expression* [, …]) }

table-spec:{ [*userid*.] *table-name* [[**AS**] *correlation-name*] | *select-statement* [**AS** *correlation-name* (*column-name* [, …])] }

contains-expression:{*table-name* | *view-name* } **CONTAINS** (*column-name* [,...], *contains-query*) [[**AS**] *score-correlation-name*]

Usage

contains-expression– use the **CONTAINS** clause after a table name to filter the table, and return only those rows matching the full text query specified with *contains-query*.

Every matching row of the table is returned, along with a score column that can be referred to using *score-correlation-name*, if it is specified. If *score-correlation-name* is not specified, then the score column can be referred to by the default correlation name, *contains*.

With the exception of the optional correlation name argument, the **CONTAINS** clause takes the same arguments as the **CONTAINS** search condition. There must be a **TEXT** index on the columns listed in the **CONTAINS** clause.

See *CONTAINS Conditions* and *Reference: Building Blocks, Tables, and Procedures > SQL Language Elements > Search Conditions > CONTAINS Conditions*.

For the full syntax and description of the **FROM** clause, see *Reference: Statements and Options > SQL Statements > FROM Clause.*

See also

- *CONTAINS Conditions* on page 26

CONTAINS Conditions

Perform a full text query using the **CONTAINS** clause in the **FROM** clause of a **SELECT** statement, or by using the **CONTAINS** search condition (predicate) in a **WHERE** clause.

Both methods return the same rows; however, the **CONTAINS** clause also returns scores for the matching rows.

Syntax

```
CONTAINS ( column-name [,...], contains-query-string )
```

```
contains-query-string:
    simple-expression
    | or-expression

simple-expression:
    primary-expression
    | and-expression

or-expression:
    simple-expression { OR | | } contains-query-string

primary-expression:
    basic-expression
    | FUZZY " fuzzy-expression "
    | and-not-expression

and-expression:
    primary-expression [ AND | & ] simple-expression

and-not-expression:
    primary-expression [ AND | & ] { NOT | - } basic-expression

basic-expression:
    term
    | phrase
    | ( contains-query-string )
    | proximity-expression

fuzzy-expression:
    term
    | fuzzy-expression term

term:
    simple-term
    | prefix-term

prefix-term:
    simple-term*

phrase:
    " phrase-string "

proximity-expression:
    term ( BEFORE | NEAR) [minimum distance, | maximum distance ]
 term
    | term { BEFORE | NEAR | ~ } term

phrase-string:
    term
    | phrase-string term
```

Parameters

simple-term – a string separated by white space and special characters that represents a single indexed term (word) for which to search.

distance – a positive integer.

and-expression – use *and-expression* to specify that both *primary-expression* and *simple-expression* must be found in the **TEXT** index. By default, if no operator is specified between terms or expressions, an and-expression is assumed. For example, `'a b''` is interpreted as `'a AND b'`. An ampersand (&) can be used instead of AND, and can abut the expressions or terms on either side (for example, `'a & b'`).

and-not-expression – use *and-not-expression* to specify that *primary-expression* must be present in the **TEXT** index, but that *basic-expression* must not be found in the **TEXT** index. This is also known as a negation. When you use a hyphen for negation, a space must precede the hyphen, and the hyphen must abut the subsequent term. For example, `'a -b'` is equivalent to `'a AND NOT b'`; whereas for `'a - b'`, the hyphen is ignored and the string is equivalent to `'a AND b'`. `'a-b'` is equivalent to the phrase `'"a b"'`.

or-expression – use *or-expression* to specify that at least one of *simple-expression* or *contains-query-string* must be present in the **TEXT** index. For example, `'a|b'` is interpreted as `'a OR b'`.

fuzzy-expression – use fuzzy-expression to find terms that are similar to what you specify. Fuzzy matching is only supported on **NGRAM TEXT** indexes. See *Fuzzy Search*.

proximity-expression – use *proximity-expression* to search for terms that are near each other. For example, `'b NEAR[2,5] c'` searches for instances of b and c that are at most five and at least 2 terms away from each other. The order of terms is not significant; `'b NEAR c'` is equivalent to `'c NEAR b'`. If **NEAR** is specified without *distance*, a default of 10 terms is applied. You can specify a tilde (~) instead of **NEAR**. This is equivalent to specifying **NEAR** without a distance so a default of 10 terms is applied. **NEAR** expressions cannot be chained together (for example, `'a NEAR[1] b NEAR[1] c'`).

BEFORE is like **NEAR**, except that the order of terms is significant. `'b BEFORE c'` is not equivalent to `'c BEFORE b'`; in the former, the term `'b'` must precede `'c'` while in the latter the term `'b'` must follow `'c'`. **BEFORE** accepts both minimum and maximum distances like **NEAR**. The default minimum distance is 1. The minimum distance, if given, must be less than or equal to the maximum distance; otherwise, an error is returned.

prefix-term – use *prefix-term* to search for terms that start with the specified prefix. For example, `'datab*'` searches for any term beginning with *datab*. This is also known as a prefix search. In a prefix search, matching is performed for the portion of the term to the left of the asterisk.

Usage

The **CONTAINS** search condition takes a column list and *contains-query-string* as arguments.

The **CONTAINS** search condition can be used anywhere a search condition (also referred to as predicate) can be specified, and returns TRUE or FALSE. *contains-query-string* must be a constant string, or a variable, with a value that is known at query time.

If multiple columns are specified, then they must all refer to a single base table; a **TEXT** index cannot span multiple base tables. You can reference the base directly in the **FROM** clause, or use it in a view or derived table, provided that the view or derived table does not use **DISTINCT**, **GROUP BY**, **ORDER BY**, **UNION**, **INTERSECT**, **EXCEPT**, or a row limitation.

Queries using ANSI join syntax are supported (**FULL OUTER JOIN**, **RIGHT OUTER JOIN**, **LEFT OUTER JOIN**), but may have suboptimal performance. Use outer joins for **CONTAINS** in the **FROM** clause only if the `score` column from each of the **CONTAINS** clauses is required. Otherwise, move **CONTAINS** to an **ON** condition or **WHERE** clause.

These types of queries are unsupported:

- Remote queries using a SQL Anywhere table with a full **TEXT** index that is joined to a remote table.
- Queries using Sybase IQ and SQL Anywhere tables, where the full **TEXT** index to be used is on the SQL Anywhere table.
- Queries using TSQL style outer join syntax (`*=*`, `=*` and `*=`).

If you use a SQL variable less than 32KB in length as a search term and the type of variable is `LONG VARCHAR`, use **CAST** to convert the variable to `VARCHAR` data type. For example:

```
SELECT * FROM tab1 WHERE CONTAINS(c1, cast(v1 AS VARCHAR(64))
```

The following warnings apply to the use of non-alphanumeric characters in query strings:

- An asterisk in the middle of a term returns an error.
- Avoid using non-alphanumeric characters (including special characters) in fuzzy-expression, because they are treated as white space and serve as term breakers.
- If possible, avoid using non-alphanumeric characters that are not special characters in your query string. Any non-alphanumeric character that is not a special character causes the term containing it to be treated as a phrase, breaking the term at the location of the character. For example, `'things we've done'` is interpreted as `'things "we ve" done'`.

Within phrases, the asterisk is the only special character that continues to be interpreted as a special character. All other special characters within phrases are treated as white space and serve as term breakers.

Interpretation of *contains-query-string* takes place in two main steps:

- Step 1: Interpretation of operators and precedence: During this step, keywords are interpreted as operators, and rules of precedence are applied.
- Step 2: Application of text configuration object settings: During this step, the text configuration object settings are applied to terms. Any query terms that exceed the term length settings, or that are in the stop list, are dropped.

See also

- *Fuzzy Search* on page 34

Operator Precedence in a CONTAINS Search Condition

During query evaluation, expressions are evaluated using an order of precedence.

The order of precedence for evaluating query expressions is:

1. **FUZZY**, **NEAR**
2. **AND NOT**
3. **AND**
4. **OR**

Allowed Syntax for Asterisk (*)

The asterisk is used for prefix searching in a query.

An asterisk can occur at the end of the query string, or be followed by a space, ampersand, vertical bar, closing bracket, or closing quotation mark. Any other usage of asterisk returns an error.

The table "Asterisk interpretations" shows allowable asterisk usage:

Table 9. Asterisk interpretations

Query string	Equivalent to	Interpreted as
`'th*&best'`	`'th* AND best'` and `'th* best'`	Find any term beginning with th, and the term best.
`'th*|best'`	`'th* OR best'`	Find either any term beginning with th, or the term best.
`'very&(best|th*)'`	`'very AND (best OR th*)'`	Find the term very, and the term best or any term beginning with th.
`'"fast auto*"'`		Find the term fast, immediately followed by a term beginning with auto.
`'"auto* price"'`		Find a term beginning with auto, immediately followed by the term price.

Note: Interpretation of query strings containing asterisks varies depending on the text configuration object settings.

Allowed Syntax for Hyphen (-)

The hyphen can be used in a query for term or expression negation, and is equivalent to NOT.

Whether a hyphen is interpreted as a negation depends on its location in the query string. For example, when a hyphen immediately precedes a term or expression, it is interpreted as a negation. If the hyphen is embedded within a term, it is interpreted as a hyphen.

A hyphen used for negation must be preceded by a white space, and followed immediately by an expression.

When used in a phrase of a fuzzy expression, the hyphen is treated as white space and used as a term breaker.

The table "Hyphen interpretations" shows the allowed syntax for hyphen:

Table 10. Hyphen interpretations

Query string	Equivalent to	Interpreted as
`'the -best'`	`'the AND NOT best', 'the AND -best', 'the & -best', 'the NOT best'`	Find the term the, and not the term best.
`'the -(very best)'`	`'the AND NOT (very AND best)'`	Find the term the, and not the terms very and best.
`'the -"very best"'`	`'the AND NOT "very best"'`	Find the term the, and not the phrase very best.
`'alpha-numerics'`	`'"alpha numerics"'`	Find the term alpha, immediately followed by the term numerics.
`'wild -west'`	`'wild west'`, and `'wild AND west'`	Find the term wild, and the term west.

Allowed Syntax for Special Characters

The table "Special character interpretations" shows the allowed syntax for all special characters, except asterisk and hyphen.

The asterisk and hyphen characters are not considered special characters, if they are found in a phrase, and are dropped.

Note: The restrictions on specifying string literals also apply to the query string. For example, apostrophes must be within an escape sequence.

Table 11. Special character interpretations

Character or syntax	Usage examples and remarks
ampersand (&)	The ampersand is equivalent to **AND**, and can be specified as follows: • `'a & b'` • `'a &b'` • `'a& b'` • `'a&b'`
vertical bar (|)	The vertical bar is equivalent to **OR**, and can be specified as follows: • `'a| b'` • `'a |b'` • `'a | b` • `'a|b'`
double quotes (")	Double quotes are used to contain a sequence of terms where order and relative distance are important. For example, in the query string `'learn "full text search"'`, "full text search" is a phrase. In this example, learn can come before or after the phrase, or exist in another column (if the **TEXT** index is built on more than one column), but the exact phrase must be found in a single column.
parentheses ()	Parentheses are used to specify the order of evaluation of expressions, if different from the default order. For example, `'a AND (b|c)'` is interpreted as a, and b or c.
tilde (~)	The tilde is equivalent to **NEAR**, and has no special syntax rules. The query string `'full~text'` is equivalent to `'full NEAR text'`, and is interpreted as: the term full within ten terms of the term text.
square brackets [ ]	Square brackets are used in conjunction with the keyword **NEAR** to contain *distance*. Other uses of square brackets return an error.

Effect of Dropped Terms

A **TEXT** index may exclude terms that meet certain conditions.

TEXT indexes are built according to the settings defined for the text configuration object used to create the **TEXT** index. A**TEXT** index excludes terms that meet any of the following conditions:

- The term is included in the stop list.
- The term is shorter than the minimum term length (**GENERIC** only).
- The term is longer than the maximum term length.

The same rules apply to query strings. The dropped term can match zero or more terms at the end or beginning of the phrase. For example, suppose the term 'the' is in the stop list:

- If the term appears on either side of an **AND**, **OR**, or **NEAR**, then both the operator and the term are removed. For example, searching for `'the AND apple'`, `'the OR apple'`, or `'the NEAR apple'` are equivalent to searching for `'apple'`.
- If the term appears on the right side of an **AND NOT**, both the **AND NOT** and the term are dropped. For example, searching for `'apple AND NOT the'` is equivalent to searching for `'apple'`.
- If the term appears on the left side of an **AND NOT**, the entire expression is dropped. For example, searching for `'the AND NOT apple'` returns no rows. Another example: `'orange and the AND NOT apple'` is the same as `'orange AND (the AND NOT apple)'` which, after the **AND NOT** expression is dropped, is equivalent to searching for `'orange'`. Contrast this with the search expression `'(orange and the) and not apple'`, which is equivalent to searching for `'orange and not apple'`.
- If the term appears in a phrase, the phrase is allowed to match with any term at the position of the dropped term. For example, searching for `'feed the dog'` matches `'feed the dog'`, `'feed my dog'`, `'feed any dog'`, and so on.

Note: If all of the terms for which you are searching are dropped, Sybase IQ returns the error `CONTAINS has NULL search term`. SQL Anywhere reports no error and returns zero rows.

Query Match Score

You can sort query results using the score that indicates the closeness of a match.

When you include a **CONTAINS** clause in the **FROM** clause of a query, each match has a score associated with it. The score indicates how close the match is, and you can use score information to sort the data. Two main criteria determine score:

- The number of times a term appears in the indexed row. The more times a term appears in an indexed row, the higher its score.
- The number of times a term appears in the **TEXT** index. The more times a term appears in a **TEXT** index, the lower its score.

Depending on the type of full text search, other criteria affect scoring. For example, in proximity searches, the proximity of search terms impacts scoring. By default, the result set of a **CONTAINS** clause has the correlation name `contains` that has a single column in it called `score`. You can refer to `"contains".score` in the **SELECT** list, **ORDER BY** clause, or other parts of the query. However, because `contains` is a SQL reserved word, you must remember to put it in double quotes. Alternatively, you can specify another correlation name, for example, `CONTAINS ( expression ) AS ct`. The examples for full text search refer to the `score` column as `ct.score`.

This statement searches `MarketingInformation.Description` for terms starting with ‘stretch” or terms starting with “comfort”:

```
SELECT ID, ct.score, Description
FROM MarketingInformation
```

```
CONTAINS ( MarketingInformation.Description,
           'stretch* | comfort*' )
AS ct ORDER BY ct.score DESC;
```

NGRAM TEXT Index Searches

Fuzzy and non-fuzzy search capability over a **TEXT** index is possible for **TEXT** indexes of type **NGRAM**.

Fuzzy Search

Fuzzy search capability over a **TEXT** index is possible only if the **TEXT** index is of type **NGRAM**. The **GENERIC TEXT** index cannot handle the fuzzy search.

Fuzzy searching can be used to search for misspellings or variations of a word. To do so, use the **FUZZY** operator followed by a string in double quotes to find an approximate match for the string.

Using the **FUZZY** operator is equivalent to breaking the string manually into substrings of length *n* and separating them with **OR** operators. For example, if you have a text index configured with the **NGRAM** term breaker and a **MAXIMUM TERM LENGTH** of 3, specifying `'FUZZY "500 main street" '` is equivalent to specifying `'500 OR mai OR ain OR str OR tre OR ree OR eet'`.

The **FUZZY** operator is useful in a full text search that returns a score. Many approximate matches may be returned, but usually only the matches with the highest scores are meaningful.

Note: Fuzzy search does not support prefix or suffix searching. For example, the search clause cannot be “v*” or “*vis”.

For more information on fuzzy search, see *SQL Anywhere 11.0.1 > SQL Anywhere Server-SQL Usage > Querying and Modifying Data > Querying Data > Types of full text searches > Fuzzy searches*.

Example 1: fuzzy search over an NGRAM TEXT index

Create a table and an **NGRAM TEXT** index:

```
CREATE TEXT CONFIGURATION NGRAMTxtcfg
   FROM default_char;
ALTER TEXT CONFIGURATION NGRAMTxtcfg TERM BREAKER    NGRAM;
ALTER TEXT CONFIGURATION NGRAMTxtcfg maximum term    length 3;
CREATE TABLE t_iq(a int, b varchar(100));
CREATE TEXT INDEX TXT_IQ on t_iq(b) CONFIGURATION    NGRAMTxtcfg
```

Insert this data into the table:

```
INSERT INTO t_iq values (1,'hello this is hira ');
INSERT INTO t_iq values(2, ' book he ookw worm okwo
kwor');
INSERT INTO t_iq values(3,'Michael is a good person');
```

```
INSERT INTO t_iq values(4,'hello this is evaa');
INSERT INTO t_iq values(5,'he is a bookworm');
INSERT INTO t_iq values (6,'boo ook okw kwo wor orm');
```

After inserting the data, execute this query to perform fuzzy searching over an **NGRAM TEXT** index:

```
SELECT * FROM t_iq WHERE CONTAINS (b,'FUZZY "bookerm"');
```

The results of the query are:

```
a          b
2          book he ookw worm okwo kwor
5          he is a bookworm
6          boo ook okw kwo wor orm
```

Example 2: additional letter in the fuzzy search clause

This query illustrates an additional letter in the fuzzy search clause:

```
SELECT * FROM t_iq WHERE CONTAINS (b,'FUZZY "hellow"');
```

The results of the query are:

```
a          b
1          hello this is hira
4          hello this is evaa
```

Example 3: letter removed from the fuzzy search clause

In this query, a letter is removed from the fuzzy search clause:

```
SELECT * FROM t_iq WHERE CONTAINS(b, 'FUZZY "hllo"');
```

The results of the query are:

```
a          b
1          hello this is hira
4          hello this is evaa
```

Non-fuzzy Search

Non-fuzzy search on **NGRAM** breaks the term into corresponding n-grams and searches for the n-grams in the **NGRAM TEXT** index.

The query `CONTAINS ( M.Description, 'ams' ) ct;` illustrates a non-fuzzy **NGRAM** search over a 2GRAM index, which is semantically equal to searching query `CONTAINS( M.Description, '"am ms"' ) ct;`

If you search for a 'v*' TERM on a 2GRAM index, then v followed by any alphabet is considered as a matching 2GRAM for the searching term and is output as a result.

The query `CONTAINS (M.Description, 'white whale') ct;` illustrates a non-fuzzy **NGRAM** search over a 3GRAM index and is semantically equal to searching query `CONTAINS (M.Description, '"whi hit ite wha hal ale"');`

The difference between **NGRAM** fuzzy and non-fuzzy search is that fuzzy search is a disjunction over individual GRAMS. Non-fuzzy search is a conjunction over the individual GRAMS. When **GENERIC** and **NGRAM TEXT** indexes are created on the same column, then the **GENERIC TEXT** index is used for a query with non-fuzzy search and the **NGRAM TEXT** index is used for fuzzy search.

Example 1: non-fuzzy search after creating a GENERIC TEXT index on the same column

This query illustrates non-fuzzy search after creating a **GENERIC TEXT** index on the same column:

```
SELECT * FROM t_iq WHERE CONTAINS (b,'bookworm');
```

The results of the query are:

```
a          b
5          he is a bookworm
```

Example 2: fuzzy search with both NGRAM and GENERIC TEXT indexes on the same column

This query illustrates fuzzy search with both **NGRAM** and **GENERIC TEXT** indexes on the same column:

```
SELECT * FROM t_iq
WHERE CONTAINS (b,'FUZZY "bookwerm"');
```

The results of the query are:

```
a          b
2          book he ookw worm okwo kwor
5          he is a bookworm
6          boo ook okw kwo wor orm
```

Example 3: fuzzy search phrase in a non-fuzzy search clause

This query illustrates the behavior of a fuzzy search phrase in a non-fuzzy search clause:

```
SELECT * FROM t_iq WHERE CONTAINS (b,'bookwerm');
```

No result is returned for this query.

Queries on LONG BINARY Columns

In **WHERE** clauses of the **SELECT** statement, you can use `LONG BINARY` columns only in IS NULL and IS NOT NULL expressions, in addition to the **BYTE_LENGTH64**, **BYTE_SUBSTR64**, **BYTE_SUBSTR**, **BIT_LENGTH**, **OCTET_LENGTH**, **CHARINDEX**, and **LOCATE** functions.

You cannot use `LONG BINARY` columns in the **SELECT** statement clauses **ORDER BY**, **GROUP BY**, and **HAVING** or with the **DISTINCT** keyword.

Sybase IQ does not support **LIKE** predicates on **LONG BINARY** (BLOB) columns or variables. Searching for a pattern in a LONG BINARY column using a **LIKE** predicate returns the error `Invalid data type comparison in predicate`.

See also

- *Function Support* on page 69

Queries on LONG VARCHAR Columns

In **WHERE** clauses of the **SELECT** statement, you can use LONG VARCHAR columns only in IS NULL and IS NOT NULL expressions, in addition to the **BIT_LENGTH**, **CHAR_LENGTH**, **CHAR_LENGTH64**, **CHARINDEX**, **LOCATE**, **OCTET_LENGTH**, **PATINDEX**, **SUBSTRING64**, and **SUBSTRING** functions.

You can use the **LIKE** predicate to search for a pattern on a LONG VARCHAR column. All patterns of 126 or fewer characters are supported. Patterns longer than 254 characters are not supported. Some patterns between 127 and 254 characters in length are supported, depending on the contents of the pattern.

The **LIKE** predicate supports **LONG VARCHAR** (CLOB) variables of any size of data. Currently, a SQL variable can hold up to 2GB - 1 in length.

You cannot use LONG VARCHAR columns in the **SELECT** statement clauses **ORDER BY**, **GROUP BY**, and **HAVING** or with the **DISTINCT** keyword (**SELECT DISTINCT** and **COUNT DISTINCT**).

See also

- *Function Support* on page 69

CONTAINS Predicate Support

You can create a **WORD** (**WD**) index on a LONG VARCHAR (CLOB) column and use the **CONTAINS** predicate to search the column for string constants of maximum length 255 characters.

The **CONTAINS** predicate is not supported on **LONG BINARY** (BLOB) columns using **WD** indexes. If you attempt to search for a string in a LONG BINARY column with a **WD** index using a **CONTAINS** predicate, an error is returned. **TEXT** indexes that use an external library support **CONTAINS** on binary data.

See *Reference: Building Blocks, Tables, and Procedures > SQL Language Elements > Search Conditions > CONTAINS Conditions*.

Performance Monitoring of LONG BINARY and LONG VARCHAR Columns

The Sybase IQ performance monitor displays performance data for `LONG BINARY` and `LONG VARCHAR` columns.

Stored Procedure Support

Learn about stored procedure support for the LONG BINARY (BLOB) and LONG VARCHAR (CLOB) data type columns and full text searching.

Term Management in a TEXT Index

You can use stored procedures to break strings into terms, to discover how many terms are in the **TEXT** index and their position, and to display statistical information about the **TEXT** indexes.

sa_char_terms System Procedure

Breaks a CHAR string into terms and returns each term as a row along with its position.

Syntax

```
sa_char_terms( 'char-string' [, 'text-config-name'
[, 'owner' ] ] ]     )
```

Parameters

char-string – the CHAR string you are parsing.

text-config-name – the text configuration object to apply when processing the string. The default value is 'default_char'.

owner – the owner of the specified text configuration object. The default value is DBA.

Description

You can use **sa_char_terms** to find out how a string is interpreted when the settings for a text configuration object are applied. This can be helpful when you want to know what terms would be dropped during indexing or from a query string.

Permissions

None.

Example

Return the terms in the CHAR string 'the quick brown fox jumped over the fence':

```
CALL sa_char_terms
( 'the quick brown fox jumped over the fence' );
```

Table 12. CHAR string interpretation

Term	Position
the	1
quick	2
brown	3
fox	4
jumped	5
over	6
the	7
fence	8

sa_nchar_terms System Procedure

Breaks an NCHAR string into terms and returns each term as a row along with its position.

Syntax

```
sa_nchar_terms( 'char-string' [ , 'text-config-
name' [, 'owner' ] ] ]     )
```

Parameters

char-string – the NCHAR string you are parsing.

text-config-name – the text configuration object to apply when processing the string. The default value is 'default_nchar'.

owner – the owner of the specified text configuration object. The default value is DBA.

Description

You can use **sa_nchar_terms** to find out how a string is interpreted when the settings for a text configuration object are applied. This can be helpful when you want to know what terms would be dropped during indexing or from a query string.

The syntax for **sa_nchar_terms** is similar to the syntax for the **sa_char_terms** system procedure.

Note: The NCHAR data type is supported only for **IN SYSTEM** tables.

Permissions

None.

sa_text_index_stats System Procedure

Returns statistical information about the **TEXT** indexes in the database.

Syntax

```
sa_text_index_stats( )
```

Description

Use **sa_text_index_stats** to view statistical information for each **TEXT** index in the database.

Table 13. Statistical information for TEXT indexes returned by sa_text_index_stats

Column name	Type	Description
owner_id	UNSIGNED INT	ID of the owner of the table
table_id	UNSIGNED INT	ID of the table
index_id	UNSIGNED INT	ID of the **TEXT** index
text_config_id	UNSIGNED BIGINT	ID of the text configuration referenced by the **TEXT** index
owner_name	CHAR(128)	Name of the owner
table_name	CHAR(128)	Name of the table
index_name	CHAR(128)	Name of the **TEXT** index
text_config_name	CHAR(128)	Name of the text configuration object
doc_count	UNSIGNED BIGINT	Total number of indexed column values in the **TEXT** index
doc_length	UNSIGNED BIGINT	Total length of data in the **TEXT** index
pending_length	UNSIGNED BIGINT	Total length of the pending changes
deleted_length	UNSIGNED BIGINT	Total length of the pending deletions

Column name	Type	Description
last_refresh	TIMESTAMP	Date and time of the last refresh

The `pending_length`, `deleted_length`, and `last_refresh` values are NULL for **IMMEDIATE REFRESH TEXT** indexes.

Permissions
DBA authority required.

Example
Return statistical information for each **TEXT** index in the database:

```
CALL sa_text_index_stats( );
```

sa_text_index_vocab System Procedure

Lists all terms that appear in a **TEXT** index, and the total number of indexed values in which each term appears.

Syntax

```
sa_text_index_vocab(
    'text-index-name',
    'table-name',
    'table-owner'
    )
```

Parameters
text-index-name – use this CHAR(128) parameter to specify the name of the **TEXT** index.

table-name – use this CHAR(128) parameter to specify the name of the table on which the **TEXT** index is built.

table-owner – use this CHAR(128) parameter to specify the owner of the table.

Description
sa_text_index_vocab returns all terms that appear in a **TEXT** index, and the total number of indexed values in which each term appears (which is less than the total number of occurrences, if the term appears multiple times in some indexed values).

Parameter values cannot be host variables or expressions. The arguments *text-index-name*, *table-name*, and *table-owner* must be constraints or variables.

Permissions
DBA authority, or SELECT permission on the indexed table is required.

Example

Execute **sa_text_index_vocab** to return all the terms that appear in the **TEXT** index `MyTextIndex` on table `Customers` owned by `GROUPO`:

```
sa_text_index_vocab
('MyTextIndex','Customers','GROUPO');
```

Table 14. Terms in the index

term	freq
a	1
Able	1
Acres	1
Active	5
Advertising	1
Again	1
...	...

External Library Identification

The **sa_list_external_library** stored procedure lists the libraries that are currently loaded in the server. Once identified, use **sa_external_library_unload** to unload the library from the server.

sa_external_library_unload System Procedure

Unloads an external library.

Syntax

```
sa_external_library_unload ( [ 'external-library'  ] )
```

Parameters

external-library – optionally use this `LONG VARCHAR` parameter to specify the name of a library to be unloaded. If no library is specified, all external libraries that are not in use are unloaded.

Description

If an external library is specified, but is in use or is not loaded, an error is returned. If no parameter is specified, an error is returned if no loaded external libraries are found.

Permissions
DBA authority required.

Examples
Unload an external library called `myextlib.dll`:

```
CALL sa_external_library_unload( 'myextlib.dll' );
```

Unload all libraries that are not currently in use:

```
CALL sa_external_library_unload();
```

sa_list_external_library Procedure

Lists the external libraries currently loaded in the server.

Syntax

```
sa_list_external_library( )
```

Description
Returns a list of external libraries loaded in the engine along with their reference count.

The reference count is the number of instances of the library in the engine. An external library can be unloaded by executing the procedure **sa_external_library_unload**, only if its reference count is 0.

Permissions
DBA authority required.

Example
List the external libraries and their reference count:

```
CALL sa_list_external_library()
```

Large Object Data Compression

The **sp_iqsetcompression** stored procedure controls the compression of large object columns.

sp_iqsetcompression controls the compression of columns of data type `LONG BINARY` and `LONG VARCHAR` when writing database buffers to disk. You can also use **sp_iqsetcompression** to disable compression. This functionality saves CPU cycles, because certain data formats stored in a `LONG BINARY` or `LONG VARCHAR` column (for example, JPG files) are already compressed and gain nothing from additional compression.

The **sp_iqshowcompression** stored procedure displays the compression setting of large object columns.

sp_iqsetcompression Procedure

Sets compression of data in columns of `LONG BINARY` (`BLOB`) and `LONG VARCHAR` (`CLOB`) data types.

Syntax

```
sp_iqsetcompression ( owner, table, column, on_off_flag )
```

Permissions

Requires DBA authority.

Description

sp_iqsetcompression provides control of compression of `LONG BINARY` (`BLOB`) and `LONG VARCHAR` (`CLOB`) data type columns. The compression setting applies only to Sybase IQ base tables.

A side effect of **sp_iqsetcompression** is that a **COMMIT** occurs after you change the compression setting.

Table 15. sp_iqsetcompression parameters

Name	Description
owner	Owner of the table for which you are setting compression
table	Table for which you are setting compression
column	Column for which you are setting compression
on_off_flag	Compression setting: ON enables compression, OFF disables compression

Example

Assume this table definition:

```
CREATE TABLE USR.pixTable (picID INT NOT NULL,
picJPG LONG BINARY NOT NULL);
```

To turn off compression on the LOB column `picJPG`, call **sp_iqsetcompression** (you must have DBA permission):

```
CALL sp_iqsetcompression('USR', 'pixTable', 'picJPG',
'OFF') ;
```

This command returns no rows.

sp_iqshowcompression Procedure

Displays compression settings for columns of `LONG BINARY` (`BLOB`) and `LONG VARCHAR` (`CLOB`) data types.

Syntax

sp_iqshowcompression (*owner*, *table*, *column*)

Permissions

Requires DBA authority.

Description

Returns the column name and compression setting. Compression setting values are 'ON' (compression enabled) and 'OFF' (compression disabled).

Table 16. sp_iqshowcompression parameters

Name	Description
owner	Owner of the table for which you are setting compression
table	Table for which you are setting compression
column	Column for which you are setting compression

Example

Assume this table definition:

```
CREATE TABLE USR.pixTable (picID INT NOT NULL,
picJPG LONG BINARY NOT NULL);
```

To check the compression status of the columns in the `pixTable` table, call **sp_iqshowcompression** (you must have DBA permission):

```
CALL sp_iqshowcompression('USR', 'pixTable',
'picJPG') ;
```

This command returns one row:

```
'picJPG','ON'
```

Information About Large Object Columns

The stored procedure **sp_iqindexsize** displays the size of an individual `LONG BINARY` and `LONG VARCHAR` column.

Size of a LONG BINARY Column

sp_iqindexsize output that shows a `LONG BINARY` column with approximately 42GB of data.

The page size is 128KB. The `largelob` Info type is in the last row.

Username	Indexname	Type	Info	KBytes	Pages	Compressed Pages
DBA	test10.DBA.ASIQ_IDX_T128_C3_FP	FP	Total	42953952	623009	622923
DBA	test10.DBA.ASIQ_IDX_T128_C3_FP	FP	vdo	0	0	0
DBA	test10.DBA.ASIQ_IDX_T128_C3_FP	FP	bt	0	0	0
DBA	test10.DBA.ASIQ_IDX_T128_C3_FP	FP	garray	0	0	0
DBA	test10.DBA.ASIQ_IDX_T128_C3_FP	FP	bm	136	2	1
DBA	test10.DBA.ASIQ_IDX_T128_C3_FP	FP	barray	2312	41	40
DBA	test10.DBA.ASIQ_IDX_T128_C3_FP	FP	dpstore	170872	2551	2549
DBA	test10.DBA.ASIQ_IDX_T128_C3_FP	FP	largelob	42780632	620415	620333

In this example, the compression ratio is 42953952/(623009*128) = 53.9%.

Size of a LONG VARCHAR Column

sp_iqindexsize output that shows a `LONG VARCHAR` column with approximately 42GB of data.

The page size is 128KB. The `largelob` Info type is in the last row.

Username	Indexname	Type	Info	KBytes	Pages	Compressed Pages
DBA	test10.DBA.ASIQ_IDX_T128_C3_FP	FP	Total	42953952	623009	622923
DBA	test10.DBA.ASIQ_IDX_T128_C3_FP	FP	vdo	0	0	0
DBA	test10.DBA.ASIQ_IDX_T128_C3_FP	FP	bt	0	0	0
DBA	test10.DBA.ASIQ_IDX_T128_C3_FP	FP	garray	0	0	0
DBA	test10.DBA.ASIQ_IDX_T128_C3_FP	FP	bm	136	2	1
DBA	test10.DBA.ASIQ_IDX_T128_C3_FP	FP	barray	2312	41	40
DBA	test10.DBA.ASIQ_IDX_T128_C3_FP	FP	dpstore	170872	2551	2549
DBA	test10.DBA.ASIQ_IDX_T128_C3_FP	FP	largelob	42780632	620415	620333

In this example, the compression ratio is 42953952/(623009*128) = 53.9%.

Large Object Data Load and Unload

Learn how to export and load large object data in Sybase IQ.

Large Object Data Exports

The Sybase IQ data extraction facility includes the **BFILE** function, which allows you to extract individual LONG BINARY and LONG VARCHAR cells to individual operating system files on the server.

You can use **BFILE** with or without the data extraction facility.

BFILE Function

Extracts individual LONG BINARY and LONG VARCHAR cells to individual operating system files on the server.

Syntax

```
BFILE( file-name-expression, large-object-column )
```

Parameters

file-name-expression – the name of the output file into which the LONG BINARY or LONG VARCHAR data is written. This file name can be up to (32K -1) bytes in length, but must be a valid path name that is supported by the file system.

large-object-column – the name of the LONG BINARY or LONG VARCHAR column.

Usage

BFILE returns:

- 1, if the file is successfully written
- 0, if the file is not successfully opened or written
- NULL, if the LONG BINARY or LONG VARCHAR cell value is NULL

If the LONG BINARY or LONG VARCHAR cell value is NULL, no file is opened and no data is written.

The file path is relative to where the server was started and the open and write operations execute with the permissions of the server process. Tape devices are not supported for the **BFILE** output file.

LONG BINARY and LONG VARCHAR cells retrieved other than with the **BFILE** function (that is, retrieved through the client/server database connection later) are limited in size to a maximum length of 2GB. Use **SUBSTRING64** or **BYTE_SUBSTR64** to retrieve LONG BINARY cells greater than 2GB using a **SELECT** (**SELECT**, **OPEN CURSOR**). Use

SUBSTRING64 to retrieve `LONG VARCHAR` cells greater than 2GB using a **SELECT** (**SELECT**, **OPEN CURSOR**). Some connection drivers, for example ODBC, JDBC™, and Open Client™, do not allow more than 2GB to be returned in one **SELECT**.

You can use **BFILE** with or without the data extraction facility.

BFILE Function Example

Use **BFILE** to extract and reload LOB data.

Create table `LobA`:

```
create table LobA
  (rowid  int primary key,
    col1   clob null,
    col2   blob null)
```

Assume `LobA` has two rows of data.

Extract the non-LOB data and the paths to the files into which the LOB data is extracted:

```
BEGIN
  SET TEMPORARY OPTION
    Temp_Extract_Name1 = LobA_data.txt';
  SELECT rowid,
    'row' + string(rowid) + '.' + 'col1',
    'row' + string(rowid) + '.' + 'col2'
  FROM LobA;
END
```

The file `LobA_data.txt` is created and contains this non-LOB data and these filenames:

```
  1,row1.col1,row1.col2,
  2,row2.col1,row2.col2,
```

Perform the LOB data extraction:

```
SELECT
  BFILE('row' + string(rowid) + '.' + 'col1',col1),
  BFILE('row' + string(rowid) + '.' + 'col2',col2)
FROM LobA;
```

After the extraction, there is a file for each cell of LOB data extracted. For example, if table `LobA` contains two rows of data with rowid values of 1 and 2, you have these files:

- `row1.col1`
- `row1.col2`
- `row2.col1`
- `row2.col2`

Reload the extracted data:

```
LOAD TABLE LobA
(rowid,
  col1 ASCII FILE (',') NULL('NULL'),
  col2 BINARY FILE (',') NULL('NULL'))
```

```
FROM LobA_data.txt'
DELIMITED BY ','
ROW DELIMITED BY '\n'
ESCAPES OFF;
```

Large Object Data Loads

Load LONG BINARY and LONG VARCHAR data using the extended syntax of the **LOAD TABLE** statement.

You can load large object data of unlimited size, unless restricted by the operating system, from a primary file in ASCII or BCP format. The maximum length of fixed-width data loaded from a primary file into large object columns is 32K - 1.

You can also specify a secondary load file in the primary load file. Each individual secondary data file contains exactly one LONG BINARY or LONG VARCHAR cell value.

Extended LOAD TABLE Syntax

LOAD TABLE has extended syntax for loading large object data.

```
LOAD [ INTO ] TABLE [ owner ].table-name
... ( column-name load-column-specification [, ...] )
... FROM 'filename-string' [, ...]
... [ QUOTES { ON | OFF } ]
... ESCAPES OFF
... [ FORMAT { ascii | binary | bcp } ]
... [ DELIMITED BY 'string' ]
...
```

```
load-column-specification:
...
| { BINARY | ASCII } FILE( integer )
| { BINARY | ASCII } FILE ( 'string' )
```

The keywords **BINARY FILE** (for LONG BINARY) or **ASCII FILE** (for LONG VARCHAR) specify to the load that the primary input file for the column contains the path of the secondary file (which contains the LONG BINARY or LONG VARCHAR cell value), rather than the LONG BINARY or LONG VARCHAR data itself. The secondary file pathname can be either fully qualified or relative. If the secondary file pathname is not fully qualified, then the path is relative to the directory in which the server was started. Tape devices are not supported for the secondary file.

Sybase IQ supports loading LONG BINARY and LONG VARCHAR values of unlimited length (subject to operating system restrictions) in the primary load file. When binary data of hexadecimal format is loaded into a LONG BINARY column from a primary file, Sybase IQ requires that the total number of hexadecimal digits is an even number. The error "Odd length of binary data value detected on column" is reported, if the cell

value contains an odd number of hexadecimal digits. Input files for LONG BINARY loads should always contain an even number of hexadecimal digits.

Sybase IQ does not support loading large object columns from primary files using **LOAD TABLE…FORMAT BINARY**. You can load large object data in binary format from secondary files.

For details on loading data using binary format, see *System Administration Guide: Volume 1 > Data Import and Export > Binary Load Formats*.

For **LOAD TABLE FORMAT BCP**, the load specification may contain only column names, **NULL**, and **ENCRYPTED**. This means that you cannot use secondary files when loading LONG BINARY and LONG VARCHAR columns using the **LOAD TABLE FORMAT BCP** option.

See *Reference: Statements and Options > SQL Statements > LOAD TABLE Statement*.

Large Object Data Load Example

Create and load a table with LONG BINARY data.

```
CREATE TABLE ltab (c1 INT, filename CHAR(64),
    ext CHAR(6), lobcol LONG BINARY NULL);
```

```
LOAD TABLE ltab (
    c1,
    filename,
    ext NULL('NULL'),
    lobcol BINARY FILE (',') NULL('NULL')
)
FROM 'abc.inp'
QUOTES OFF ESCAPES OFF;
```

The primary file abc.inp contains this data:

```
1,boston,jpg,/s1/loads/lobs/boston.jpg,
2,map_of_concord,bmp,/s1/loads/maprs/concord.bmp,
3,zero length test,NULL,,
4,null test,NULL,NULL,
```

After the LONG BINARY data is loaded into table tab, the first and second rows for column lobcol contain the contents of files boston.jpg and concord.bmp, respectively. The third and fourth rows contain a zero-length value and NULL, respectively.

Control of Load Errors

The database option SECONDARY_FILE_ERROR allows you to specify the action of the load if an error occurs while opening or reading from a secondary **BINARY FILE** or **ASCII FILE**.

If SECONDARY_FILE_ERROR is ON, the load rolls back if an error occurs while opening or reading from a secondary **BINARY FILE** or **ASCII FILE**.

If SECONDARY_FILE_ERROR is OFF (the default), the load continues, regardless of any errors that occur while opening or reading from a secondary **BINARY FILE** or **ASCII FILE**. The LONG BINARY or LONG VARCHAR cell is left with one of these values:

- NULL, if the column allows nulls
- Zero-length value, if the column does not allow nulls

Any user can set SECONDARY_FILE_ERROR for the PUBLIC group or temporary; the option setting takes effect immediately.

When logging integrity constraint violations to the load error **ROW LOG** file, the information logged for a LONG BINARY or LONG VARCHAR column is:

- Actual text as read from the primary data file, if the logging occurs within the first pass of the load operation
- Zero-length value, if the logging occurs within the second pass of the load operation

Load of Large Object Data with Trailing Blanks

The **LOAD TABLE...STRIP** option has no effect on LONG VARCHAR data.

Trailing blanks are not stripped from LONG VARCHAR data, even if the **STRIP** option is on.

Load of Large Object Data with Quotes

The **LOAD TABLE...QUOTES** option does not apply to loading LONG BINARY (BLOB) or LONG VARCHAR (CLOB) data from the secondary file, regardless of its setting,

A leading or trailing quote is loaded as part of CLOB data. Two consecutive quotes between enclosing quotes are loaded as two consecutive quotes with the **QUOTES ON** option.

Truncation of Partial Multibyte Character Data

Partial multibyte LONG VARCHAR data is truncated during the load according to the value of the TRIM_PARTIAL_MBC database option.

- If TRIM_PARTIAL_MBC is ON, a partial multibyte character is truncated for both primary data and the **LOAD** with **ASCII FILE** option.
- If TRIM_PARTIAL_MBC is OFF, the **LOAD** with **ASCII FILE** option handles the partial multibyte character according to the value of the SECONDARY_FILE_ERROR database option.

The table "Partial multibyte character on loading LONG VARCHAR with ASCII FILE option" lists how a trailing multibyte character is loaded, depending on the values of TRIM_PARTIAL_MBC and SECONDARY_FILE_ERROR.

Table 17. Partial multibyte character on loading LONG VARCHAR with ASCII FILE option

TRIM_PARTIAL_MBC	SECONDARY_FILE_ERROR	Trailing partial multibyte character found
ON	ON/OFF	Trailing partial multibyte character truncated
OFF	ON	Cell — null, if null allowed LOAD error — roll back, if null not allowed
OFF	OFF	Cell — null, if null allowed Cell — zero-length, if null not allowed

Load Support of Large Object Variables

For information on support of large object variables by the **LOAD TABLE**, **INSERT…VALUES**, **INSERT…SELECT**, **INSERT…LOCATION**, **SELECT…INTO**, and **UPDATE** SQL statements, see Large Object Data Types.

See also

- *Large Object Data Types* on page 55
- *Large Object Variables* on page 58

Large Object Data Types

Learn about the characteristics of the large object LONG BINARY and LONG VARCHAR data type columns and index support of large object data.

Large Object Data Types LONG BINARY and BLOB

Binary large object (BLOB) data in Sybase IQ is stored in columns of data type LONG BINARY or BLOB.

An individual LONG BINARY data value can have a length ranging from zero (0) to 512TB (terabytes) for an IQ page size of 128KB, or 2PB (petabytes) for an IQ page size of 512KB. (The maximum length is equal to 4GB multiplied by the database page size.) To accommodate a table with LONG BINARY data, an IQ database must be created with an IQ page size of at least 128KB (131072 bytes).

A table or database can contain any number of LONG BINARY columns up to the supported maximum columns per table and maximum columns per database, respectively.

LONG BINARY columns can be either NULL or NOT NULL and can store zero-length values. The domain BLOB is a LONG BINARY data type that allows NULL.

You cannot construct a non-FP index or join index on a LONG BINARY column.

Prefetching is disabled, if the result set contains BLOB columns.

Modify LONG BINARY columns using the **UPDATE**, **INSERT**, **LOAD TABLE**, **DELETE**, **TRUNCATE**, **SELECT...INTO** and **INSERT...LOCATION** SQL statements. Positioned updates and deletes are not supported on LONG BINARY columns.

You can insert an Adaptive Server Enterprise IMAGE column into a LONG BINARY column using the **INSERT...LOCATION** command. All IMAGE data inserted is silently right-truncated at 2147483648 bytes (2GB).

LONG BINARY Data Type Conversion

There are limited implicit data type conversions to and from the LONG BINARY data type and non-LONG BINARY data types.

There are no implicit data type conversions from the LONG BINARY data type to another non-LONG BINARY data type, except to the BINARY and VARBINARY data types for **INSERT** and **UPDATE**. There are implicit conversions to LONG BINARY data type from TINYINT, SMALLINT, INTEGER, UNSIGNED INTEGER, BIGINT, UNSIGNED BIGINT, CHAR, and VARCHAR data types. There are no implicit conversions from BIT,

REAL, DOUBLE, or NUMERIC data types to LONG BINARY data type. Implicit conversion can be controlled using the CONVERSION_MODE database option.

The currently supported byte substring functions for the LONG BINARY data type are accepted as input for implicit conversion for the **INSERT** and **UPDATE** statements. See *Function Support*.

The LONG BINARY data type can be explicitly converted to BINARY or VARBINARY. No other explicit data type conversions (for example, using the **CAST** or **CONVERT** function) exist either to or from the LONG BINARY data type.

Truncation of LONG BINARY data during conversion of LONG BINARY to BINARY or VARBINARY is handled the same way the truncation of BINARY and VARBINARY data is handled. If the STRING_RTRUNCATION option is ON, any right-truncation (of any values, not just non-space characters) on **INSERT** or **UPDATE** of a binary column results in a truncation error and the transaction is rolled back.

See also

- *Function Support* on page 69

Large Object Data Types LONG VARCHAR and CLOB

Character large object (CLOB) data in Sybase IQ is stored in columns of data type LONG VARCHAR or CLOB.

An individual LONG VARCHAR data value can have a length ranging from zero (0) to 512TB (terabytes) for an IQ page size of 128KB, or 2PB (petabytes) for an IQ page size of 512KB. (The maximum length is equal to 4GB multiplied by the database page size.) To accommodate a table with LONG VARCHAR data, an IQ database must be created with an IQ page size of at least 64KB (65536 bytes).

A table or database can contain any number of LONG VARCHAR columns up to the supported maximum columns per table and maximum columns per database, respectively.

Sybase IQ supports both single-byte and multibyte LONG VARCHAR data.

LONG VARCHAR columns can be either NULL or NOT NULL, and can store zero-length values. The domain CLOB is a LONG VARCHAR data type that allows NULL. To create a non-null LONG VARCHAR column, explicitly specify NOT NULL in the column definition.

You can create a LONG VARCHAR column using the domain CLOB, when you create a table or add a column to an existing table. For example:

```
CREATE TABLE lvtab (c1 INTEGER, c2 CLOB,
                    c3 CLOB NOT NULL);

ALTER TABLE lvtab ADD c4 CLOB;
```

You can create a **WORD** (**WD**) index on a LONG VARCHAR column. You cannot construct other non-**FP** index types and join indexes on a LONG VARCHAR column.

You can modify a LONG VARCHAR column using the **UPDATE**, **INSERT...VALUES**, **INSERT...SELECT**, **LOAD TABLE**, **DELETE**, **TRUNCATE**, **SELECT...INTO** and **INSERT...LOCATION** SQL statements. Positioned updates and deletes are not supported on LONG VARCHAR columns.

You can insert an Adaptive Server Enterprise TEXT column into a LONG VARCHAR column using the **INSERT...LOCATION** command. All TEXT data inserted is silently right-truncated at 2147483648 bytes (2GB).

LONG VARCHAR Data Type Conversion

There are limited implicit data type conversions to and from the LONG VARCHAR data type and non-LONG VARCHAR data types.

There are no implicit data type conversions from the LONG VARCHAR data type to another non-LONG VARCHAR data type, except LONG BINARY, and CHAR and VARCHAR for **INSERT** and **UPDATE** only. There are implicit conversions to LONG VARCHAR data type from CHAR and VARCHAR data types. There are no implicit conversions from BIT, REAL, DOUBLE, NUMERIC, TINYINT, SMALLINT, INT, UNSIGNED INT, BIGINT, UNSIGNED BIGINT, BINARY, VARBINARY, or LONG BINARY data types to LONG VARCHAR data type. Implicit conversion can be controlled using the CONVERSION_MODE database option.

The currently supported string functions for the LONG VARCHAR data type are accepted as input for implicit conversion for the **INSERT** and **UPDATE** statements. See *Function Support*.

The LONG VARCHAR data type can be explicitly converted to CHAR and VARCHAR. No other explicit data type conversions (for example, using the **CAST** or **CONVERT** function) exist either to or from the LONG VARCHAR data type.

Truncation of LONG VARCHAR data during conversion of LONG VARCHAR to CHAR is handled the same way the truncation of CHAR data is handled. If the STRING_RTRUNCATION option is ON and string right-truncation of non-spaces occurs, a truncation error is reported and the transaction is rolled back. Trailing partial multibyte characters are replaced with spaces on conversion.

Truncation of LONG VARCHAR data during conversion of LONG VARCHAR to VARCHAR is handled the same way the truncation of VARCHAR data is handled. If the STRING_RTRUNCATION option is ON and string right-truncation of non-spaces occurs, a truncation error is reported and the transaction is rolled back. Trailing partial multibyte characters are truncated on conversion.

See also

- *Function Support* on page 69

Large Object Variables

Sybase IQ supports large object variables.

Inbound LONG BINARY and LONG VARCHAR variables (host variables or SQL variables used by IQ) have no maximum length.

Outbound LONG BINARY and LONG VARCHAR variables (variables set by IQ) have a maximum length of 2GB - 1.

The **LOAD TABLE**, **INSERT…VALUES**, **INSERT…SELECT**, **INSERT…LOCATION**, **SELECT… INTO**, and **UPDATE** SQL statements accept LONG BINARY and LONG VARCHAR variables of any size of data. Currently, a SQL variable can hold up to 2GB - 1 in length.

The **BIT_LENGTH**, **BYTE_LENGTH**, **BYTE_LENGTH64**, **BYTE_SUBSTR**, **BYTE_SUBSTR64**, **CHARINDEX**, **LOCATE**, **OCTET_LENGTH**, and **SUBSTRING64** functions support LONG BINARY and LONG VARCHAR variables of any size data that the SQL variable can hold. In addition, the **CHAR_LENGTH**, **CHAR_LENGTH64**, **PATINDEX**, **SUBSTR**, and **SUBSTRING** functions support LONG VARCHAR variables of any size data that the SQL variable can hold.

Large Object Variable Data Type Conversion

The database option ENABLE_LOB_VARIABLES controls the data type conversion of large object variables.

ENABLE_LOB_VARIABLES Option

Controls the data type conversion of large object variables.

Allowed Values
ON, OFF

Default
OFF

Scope
DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description
ENABLE_LOB_VARIABLES controls the data type conversion of large object variables.

When ENABLE_LOB_VARIABLES is OFF, large object variables less than 32K are implicitly converted; an error is reported if a large object variable is greater than or equal to 32K. A LONG VARCHAR variable is implicitly converted to a VARCHAR data type and

truncated at 32K. A LONG BINARY variable is implicitly converted to a VARBINARY data type and truncated at 32K.

When ENABLE_LOB_VARIABLES is ON, large object variables of any size retain their original data type and size.

Example
Retain the data type and size of large object variables greater than 32K:

```
SET TEMPORARY OPTION ENABLE_LOB_VARIABLES = ON
```

Index Support of Large Object Columns

Sybase IQ supports the **TEXT** index on LONG BINARY and LONG VARCHAR columns, and the **WORD** (**WD**) index on LONG VARCHAR columns.

TEXT Index Support of Large Object Columns

The Sybase IQ **TEXT** index supports LONG BINARY and LONG VARCHAR columns.

See *SQL Statement Support* and *TEXT Indexes and Text Configuration Objects*.

See also

- *SQL Statement Support* on page 61
- *TEXT Indexes and Text Configuration Objects* on page 3

WD Index Support of LONG VARCHAR (CLOB) Columns

Sybase IQ offers limited support for the **WORD** (**WD**) index on LONG VARCHAR (CLOB) columns.

- You can create a **WD** index on columns of CHAR, VARCHAR, and LONG VARCHAR data types in Sybase Central.
- The widest column supported by the **WD** index is the maximum width for a LOB column. (The maximum length is equal to 4GB multiplied by the database page size.)
  The maximum word length supported by Sybase IQ is 255 bytes.
- All **sp_iqcheckdb** options for **WD** indexes over CHAR and VARCHAR columns are also supported for LONG VARCHAR (CLOB) columns, including allocation, check, and verify modes.
- You can use **sp_iqrebuildindex** to rebuild a **WD** index over a LONG VARCHAR (CLOB) column.

Chinese text or documents in a binary format require ETL preprocessing to locate and transform words into a form that can be parsed by the **WD** index.

SQL Statement Support

Learn about the SQL statements and syntax that support working with **TEXT** indexes and text configurations.

ALTER TEXT CONFIGURATION Statement

Alters a text configuration object.

Syntax

```
ALTER TEXT CONFIGURATION [ owner.]config-name
   STOPLIST stoplist
   | DROP STOPLIST
   | { MINIMUM | MAXIMUM } TERM LENGTH integer
   | TERM BREAKER
      { GENERIC
        [ EXTERNAL NAME library-and-entry-point-name-string ]
       | NGRAM }
   | PREFILTER EXTERNAL NAME library-and-entry-point-name-string
```

```
stoplist: string-expression
```

```
library-and-entry-point-name-string: [operating-system:]function-
name@library
```

Examples

- **Example 1 –** Create a text configuration object, `maxTerm16`, and then change the maximum term length to 16:

  ```
  CREATE TEXT CONFIGURATION maxTerm16 FROM default_char;
  ```

  ```
  ALTER TEXT CONFIGURATION maxTerm16 MAXIMUM TERM LENGTH 16;
  ```

- **Example 2 –** Add stoplist terms to the `maxTerm16` configuration object:

  ```
  ALTER TEXT CONFIGURATION maxTerm16
  ```

  ```
  STOPLIST 'because about therefore only';
  ```

- **Example 3 –** Update the text configuration object, `my_text_config`, to use the entry point `my_term_breaker` in the external library `mytermbreaker.dll` for breaking the text:

  ```
  CREATE TEXT CONFIGURATION my_text_config FROM default_char;
  ```

  ```
  ALTER TEXT CONFIGURATION my_text_config
  ```

  ```
  TERM BREAKER GENERIC EXTERNAL NAME
  'platform:my_term_breaker@mytermbreaker';
  ```

- **Example 4 –** Update the text configuration object, `my_text_config`, to use the entry point `my_prefilter` in the external library `myprefilter.dll` for prefiltering the documents:

```
ALTER TEXT CONFIGURATION my_text_config
PREFILTER EXTERNAL NAME 'platform:my_prefilter@myprefilter';
```

Usage

Use **ALTER TEXT CONFIGURATION** to change a text configuration object.

TEXT indexes are dependent on a text configuration object. Sybase IQ **TEXT** indexes use immediate refresh, and cannot be truncated; you must drop the indexes before you can alter the text configuration object.

To view the settings for text configuration objects, query the **SYSTEXTCONFIG** system view.

STOPLIST clause – use this clause to create or replace the list of terms to ignore when building a **TEXT** index. Terms specified in this list are also ignored in a query. Separate stoplist terms with spaces.

Stoplist terms cannot contain whitespace. Stoplist terms should not contain non-alphanumeric characters. Non-alphanumeric characters are interpreted as spaces and break the term into multiple terms. For example, “and/or” is interpreted as the two terms “and” and “or”. The maximum number of stoplist terms is 7999.

DROP STOPLIST clause – use this clause to drop the stoplist for a text configuration object.

MINIMUM TERM LENGTH clause – specifies the minimum length, in characters, of a term to include in the **TEXT** index. The value specified in the **MINIMUM TERM LENGTH** clause is ignored when using **NGRAM TEXT** indexes.

Terms that are shorter than this setting are ignored when building or refreshing the **TEXT** index. The value of this option must be greater than 0. If you set this option to be higher than **MAXIMUM TERM LENGTH**, the value of **MAXIMUM TERM LENGTH** is automatically adjusted to be the same as the new **MINIMUM TERM LENGTH** value.

MAXIMUM TERM LENGTH clause – with **GENERIC TEXT** indexes, specifies the maximum length, in characters, of a term to include in the **TEXT** index. Terms that are longer than this setting are ignored when building or refreshing the **TEXT** index.

The value of **MAXIMUM TERM LENGTH** must be less than or equal to 60. If you set this option to be lower than **MINIMUM TERM LENGTH**, the value of **MINIMUM TERM LENGTH** is automatically adjusted to be the same as the new **MAXIMUM TERM LENGTH** value.

TERM BREAKER clause – specifies the name of the algorithm to use for separating column values into terms. The choices for **IN SYSTEM** tables are **GENERIC** (the default) or **NGRAM**. The **GENERIC** algorithm treats any string of one or more alphanumerics, separated by non-alphanumerics, as a term.

The **NGRAM** algorithm breaks strings into n-grams. An n-gram is an n-character substring of a larger string. The **NGRAM** term breaker is required for fuzzy (approximate) matching, or for

documents that do not use whitespace or non-alphanumeric characters to separate terms. **NGRAM** is supported for **IN SYSTEM** tables.

NGRAM term breaker is built on **TEXT** indexes, so use text configuration object settings to define whether to use an **NGRAM** or **GENERIC TEXT** index.

TERM BREAKER can include the specification for the external term breaker library using **EXTERNAL NAME** and the library entry point.

DROP PREFILTER clause – drops the external prefilter and sets NULL to the prefilter columns in **ISYSTEXTCONFIG** table.

PREFILTER EXTERNAL NAME clause – specifies the entry_point and the library name of the external pre-filter library provided by external vendors.

Side Effects:

- Automatic commit.

Permissions

The user must have DBA authority to alter the text configuration object to specify the external libraries and functions for external pre-filter or term-breaker.

All other modifications to the text configuration can be done by either the owner of the configuration object or by a user having DBA authority.

ALTER TEXT INDEX Statement

Alters the definition of a **TEXT** index.

Syntax

```
ALTER TEXT INDEX [ owner.]text-index-name
   ON [ owner.]table-name
   alter-clause

alter-clause :
    rename-object | move-object

rename-object :
    RENAME { AS | TO } new-name

move-object:
    MOVE TO  dbspace-name
```

Examples

- – Create a **TEXT** index, `MyTextIndex`, defining it as **IMMEDIATE REFRESH**, rename the **TEXT** index to `Text_index_daily`, and move the **TEXT** index to a dbspace named `tispace`:

```
CREATE TEXT INDEX MyTextIndex ON Customers ( CompanyName )
IMMEDIATE REFRESH;
```

```
ALTER TEXT INDEX MyTextIndex ON Customers RENAME AS
Text_index_daily;
```

```
ALTER TEXT INDEX Text_Index_Daily ON Customers MOVE TO tispace;
```

Usage

Use **ALTER TEXT INDEX** to rename or move the **TEXT** index.

RENAME clause – rename the **TEXT** index.

MOVE clause – move the **TEXT** index to the specified dbspace.

Side Effects:

- Automatic commit.

Permissions

Must be the owner of the underlying table, or have DBA authority, or have REFERENCES permission to rename the index.

To move the **TEXT** index, you must have DBA or SPACE ADMIN authority, or you must be the table owner and have CREATE permission on the dbspace.

CREATE TEXT CONFIGURATION Statement

Creates a text configuration object.

Syntax

```
CREATE TEXT CONFIGURATION [ owner.]new-config-name
  FROM [ owner.]existing-config-name
```

Examples

- – Create a text configuration object, `max_term_sixteen`, using the `default_char` text configuration object, then use **ALTER TEXT CONFIGURATION** to change the maximum term length for `max_term_sixteen` to 16:

```
CREATE TEXT CONFIGURATION max_term_sixteen FROM default_char;
```

```
ALTER TEXT CONFIGURATION max_term_sixteen MAXIMUM TERM LENGTH 16;
```

Usage

Use **CREATE TEXT CONFIGURATION** to create a text configuration object.

Create a text configuration object using another text configuration object as a template, then alter the options as needed using the **ALTER TEXT CONFIGURATION** statement.

To view the list of all text configuration objects and their settings in the database, query the **SYSTEXTCONFIG** system view.

FROM clause – specifies the name of a text configuration object to use as the template for creating the new text configuration object. The names of the default text configuration objects are `default_char` and `default_nchar`. Only `default_char` is supported for Sybase IQ tables; `default_nchar` is supported only on SQL Anywhere tables.

Side Effects:

- Automatic commit.

Permissions

Must have DBA or RESOURCE authority.

All text configuration objects have PUBLIC access. Any user with permission to create a **TEXT** index can use any text configuration object.

CREATE TEXT INDEX Statement

Creates a **TEXT** index.

Syntax

```
CREATE TEXT INDEX text-index-name
 ON [ owner.]table-name( column-name, ...)
   [ IN dbspace-name ]
   [ CONFIGURATION [ owner.]text-configuration-name]
   [  IMMEDIATE REFRESH ]
```

Examples

- – Create a **TEXT** index, `myTxtIdx`, on the `CompanyName` column of the `Customers` table in the `iqdemo` database, using the `max_term_sixteen` text configuration object:

```
CREATE TEXT INDEX myTxtIdx ON Customers (CompanyName );
```

```
CONFIGURATION max_term_sixteen;
```

Usage

Use **CREATE TEXT INDEX** to create a **TEXT** index and to specify the text configuration object to use.

You cannot create a **TEXT** index on views or temporary tables. You cannot create a **TEXT** index on an IN SYSTEM materialized view.

TEXT indexes will not be replicated to join indexes tables. A **TEXT** index can be created on a column of a table that participates in a join index.

The **BEGIN PARALLEL IQ…END PARALLEL IQ** statement does not support **CREATE TEXT INDEX**.

ON clause – specifies the table and column on which to build the **TEXT** index.

IN clause – specifies the dbspace in which the **TEXT** index is located. If this clause is not specified, then the **TEXT** index is created in the same dbspace as the underlying table.

CONFIGURATION clause – specifies the text configuration object to use when creating the **TEXT** index. If this clause is not specified, the `default_char` text configuration object is used.

REFRESH clause – **IMMEDIATE REFRESH** is used as the default and is the only permitted value for tables in Sybase IQ. Specify **IMMEDIATE REFRESH** to refresh the **TEXT** index each time changes in the underlying table impact data in the **TEXT** index.

An **IMMEDIATE REFRESH TEXT** index is populated at creation and is refreshed whenever the data in the underlying column is changed. Once a **TEXT** index is created, you cannot change it to, or from, **IMMEDIATE REFRESH**.

Side Effects:

- Automatic commit.

Permissions

You must be the owner of the underlying table, or have DBA authority, or have REFERENCES permission.

You must have CREATE permission on the dbspace.

DROP TEXT CONFIGURATION Statement

Drops a text configuration object.

Syntax

DROP TEXT CONFIGURATION [*owner.*]*text-config-name*

Examples

- – Create and drop the `mytextconfig` text configuration object:

```
CREATE TEXT CONFIGURATION mytextconfig FROM default_char;
```

```
DROP TEXT CONFIGURATION mytextconfig;
```

Usage

Use **DROP TEXT CONFIGURATION** to drop a text configuration object.

Attempting to drop a text configuration object with dependent **TEXT** indexes results in an error. You must drop the dependent **TEXT** indexes before dropping the text configuration object.

Text configuration objects are stored in the **ISYSTEXTCONFIG** system table.

Side Effects:

- Automatic commit.

Permissions

Must be the owner of the text configuration object or have DBA authority.

DROP TEXT INDEX Statement

Removes a **TEXT** index from the database.

Syntax

```
DROP TEXT INDEX text-index-name
   ON [ owner ] table-name
```

Examples

- – Create and drop the `TextIdx` **TEXT** index:

```
CREATE TEXT INDEX TextIdx ON Customers ( Street );
DROP TEXT INDEX TextIdx ON Customers;
```

Usage

Use **DROP TEXT INDEX** to remove a **TEXT** index from the database.

ON clause – specifies the table on which the **TEXT** index is built.

You must drop dependent **TEXT** indexes before you can drop a text configuration object.

Side Effects:

- Automatic commit.

Permissions

Must be the owner of the underlying table, or have DBA authority, or have REFERENCES permission.

Function Support

Learn about the Sybase IQ functions that support the `LONG BINARY` and `LONG VARCHAR` data types.

Summary of Function Support of Large Object Data

A summary of function support of large object data types and variables.

The table "Function Support of LOB Data Types and Variables" summarizes the function support of `LONG BINARY` (`BLOB`) and `LONG VARCHAR` (`CLOB`) data types and `LONG BINARY` and `LONG VARCHAR` variables.

In addition to the functions listed in this table, you can use the **BFILE** function to extract LOB data. See *Large Object Data Exports*.

Scalar and aggregate user-defined functions support large object data types as input parameters. See *User-Defined Function Support of Large Object Columns*.

Table 18. Function Support of LOB Data Types and Variables

Function	BLOB data supported?	BLOB variables supported?	CLOB data supported?	CLOB variables supported?
BIT_LENGTH()	Yes	Yes	Yes	Yes
BYTE_LENGTH()	Yes*	Yes*	Yes*	Yes*
BYTE_LENGTH64()	Yes	Yes	Yes	Yes
BYTE_SUBSTR()	Yes	Yes	Yes	Yes
BYTE_SUBSTR64()	Yes	Yes	Yes	Yes
CHAR_LENGTH()	No	No	Yes	Yes
CHAR_LENGTH64()	No	No	Yes	Yes
CHARINDEX()	Yes	Yes	Yes	Yes
LOCATE()	Yes	Yes	Yes	Yes
OCTET_LENGTH()	Yes	Yes	Yes	Yes
PATINDEX()	No	No	Yes	Yes

Function	BLOB data supported?	BLOB variables supported?	CLOB data supported?	CLOB variables supported?
SUBSTR() / SUBSTRING()	No	No	Yes	Yes
SUBSTRING64()	Yes	Yes	Yes	Yes

*The **BYTE_LENGTH** function supports both `LONG BINARY` columns and variables and `LONG VARCHAR` columns and variables, only if the query returns less than 2GB. If the byte length of the returned `LONG BINARY` or `LONG VARCHAR` data is greater than 2GB, **BYTE_LENGTH** returns an error that says you must use the **BYTE_LENGTH64** function.

For full descriptions of these functions and examples, see *Reference: Building Blocks, Tables, and Procedures > SQL Functions*.

See also

- *User-Defined Function Support of Large Object Columns* on page 78
- *Large Object Data Exports* on page 49

BIT_LENGTH Function

The **BIT_LENGTH** function returns an unsigned 64-bit value containing the bit length of the large object column or variable parameter. If the argument is NULL, **BIT_LENGTH** returns NULL.

Syntax

```
BIT_LENGTH( large-object-column )
```

Parameters

large-object-column – the name of a `LONG VARCHAR` or `LONG BINARY` column or variable.

Usage

BIT_LENGTH supports all Sybase IQ data types and `LONG BINARY` and `LONG VARCHAR` variables of any size of data. Currently, a SQL variable can hold up to 2GB - 1 in length.

BYTE_LENGTH Function

The **BYTE_LENGTH** function returns the number of bytes in a string.

Usage

The **BYTE_LENGTH** function supports both `LONG BINARY` columns and variables and `LONG VARCHAR` columns and variables, only if the query returns less than 2GB. If the byte length of the returned `LONG BINARY` or `LONG VARCHAR` data is greater than or equal to 2GB, **BYTE_LENGTH** returns an error that says you must use the **BYTE_LENGTH64** function.

For **BYTE_LENGTH** function syntax and usage, see *Reference: Building Blocks, Tables, and Procedures> SQL Functions > Alphabetical List of Functions > BYTE_LENGTH Function [String]*.

BYTE_LENGTH64 Function

The **BYTE_LENGTH64** function returns an unsigned 64-bit value containing the byte length of the large object column or variable parameter.

Syntax

```
BYTE_LENGTH64( large-object-column )
```

Parameters

large-object-column – the name of a `LONG VARCHAR` or `LONG BINARY` column or variable.

Usage

The **BYTE_LENGTH64** function supports both `LONG BINARY` and `LONG VARCHAR` columns and `LONG BINARY` and `LONG VARCHAR` variables of any size of data. Currently, a SQL variable can hold up to 2GB - 1 in length.

BYTE_SUBSTR64 and BYTE_SUBSTR Functions

The **BYTE_SUBSTR64** and **BYTE_SUBSTR** functions return the byte substring of the large object column or variable parameter.

Syntax

```
BYTE_SUBSTR64( large-object-column, start, length )
```

```
BYTE_SUBSTR( large-object-column, start, length )
```

Parameters

large-object-column – the name of a `LONG VARCHAR` or `LONG BINARY` column or variable.

start – an integer expression indicating the start of the substring. A positive integer starts from the beginning of the string, with the first byte at position 1. A negative integer specifies a substring starting from the end of the string, with the final byte at position -1.

length – an integer expression indicating the length of the substring. A positive length specifies the number of bytes to return, starting at the *start* position. A negative length specifies the number of bytes to return, ending at the *start* position.

Usage

- Nested operations of the functions **BYTE_LENGTH64**, **BYTE_SUBSTR64**, and **BYTE_SUBSTR** do not support large object columns or variables.
- The **BYTE_SUBSTR64** and **BYTE_SUBSTR** functions support both `LONG BINARY` and `LONG VARCHAR` columns and `LONG BINARY` and `LONG VARCHAR` variables of any size of data. Currently, a SQL variable can hold up to 2GB - 1 in length.

CHAR_LENGTH Function

The **CHAR_LENGTH** function returns a signed 32-bit value containing the character length of the `LONG VARCHAR` column or variable parameter, including the trailing blanks.

Syntax

CHAR_LENGTH(*long-varchar-object*)

Parameters

long-varchar-object – the name of a `LONG VARCHAR` column or `LONG VARCHAR` variable.

Usage

- **CHAR_LENGTH** supports `LONG VARCHAR` columns and `LONG VARCHAR` variables of any size of data. Currently, a SQL variable can hold up to 2GB - 1 in length.
- If the argument is NULL, **CHAR_LENGTH** returns NULL.
- If the character length exceeds 2GB - 1 (2147483647), an error is returned.

CHAR_LENGTH64 Function

The **CHAR_LENGTH64** function returns an unsigned 64-bit value containing the character length of the LONG VARCHAR column or variable parameter, including the trailing blanks.

Syntax

```
CHAR_LENGTH64( long-varchar-object )
```

Parameters

long-varchar-object – the name of a LONG VARCHAR column in a table or a LONG VARCHAR variable.

Usage

- **CHAR_LENGTH64** supports LONG VARCHAR columns and LONG VARCHAR variables of any size of data. Currently, a SQL variable can hold up to 2GB - 1 in length.
- If the argument is NULL, **CHAR_LENGTH64** returns NULL.

CHARINDEX Function

The **CHARINDEX** function returns a 64-bit signed integer containing the position of the first occurrence of the specified string in the large object column or variable parameter. For CHAR and VARCHAR columns, **CHARINDEX** returns a 32-bit signed integer position.

Syntax

```
CHARINDEX( string-expression, large -object-column )
```

Parameters

string-expression – the string of up to 255 bytes, for which you are searching.

large-object-column – the name of the LONG VARCHAR or LONG BINARY column or variable.

Usage

- All the positions or offsets, returned or specified, in the **CHARINDEX** function are always character offsets and may be different from the byte offset for multibyte data.
- If the large object cell being searched contains more than one instance of *string-expression*, **CHARINDEX** returns only the position of the first instance.
- If the column does not contain the string, the **CHARINDEX** function returns zero (0).
- Searching for a string longer than 255 bytes returns NULL.

- Searching for a zero-length string returns 1.
- If any of the arguments is NULL, the result is NULL.
- **CHARINDEX** supports searching LONG VARCHAR and LONG BINARY columns and LONG VARCHAR and LONG BINARY variables of any size of data. Currently, a SQL variable can hold up to 2GB - 1 in length.

See *Reference: Building Blocks, Tables, and Procedures > SQL Functions > Alphabetical List of Functions > CHARINDEX Function [String]*.

LOCATE Function

The **LOCATE** function returns a 64-bit signed integer containing the position of the specified string in the large object column or variable parameter. For CHAR and VARCHAR columns, **LOCATE** returns a 32-bit signed integer position.

Syntax

```
LOCATE( large-object-column, string-expression
[, numeric-expression ] )
```

Parameters

large-object-column – the name of the LONG VARCHAR or LONG BINARY column or variable to search.

string-expression – the string of up to 255 bytes, for which you are searching.

numeric-expression – the character position or offset at which to begin the search in the string. The *numeric-expression* is a 64-bit signed integer for LONG VARCHAR and LONG BINARY columns and is a 32-bit signed integer for CHAR, VARCHAR, and BINARY columns. The first character is position 1. If the starting offset is negative, **LOCATE** returns the last matching string offset, rather than the first. A negative offset indicates how much of the end of the string to exclude from the search. The number of characters excluded is calculated as (-1 * offset) - 1.

Usage

- All the positions or offsets, returned or specified, in the **LOCATE** function are always character offsets and may be different from the byte offset for multibyte data.
- If the large object cell being searched contains more than one instance of the string:
  - If *numeric-expression* is specified, **LOCATE** starts the search at that offset in the string.
  - If *numeric-expression* is not specified, **LOCATE** returns only the position of the first instance.
- If the column does not contain the string, **LOCATE** returns zero (0).
- Searching for a string longer than 255 bytes returns NULL.
- Searching for a zero-length string returns 1.

- If any of the arguments is NULL, the result is NULL.
- **LOCATE** supports searching LONG VARCHAR and LONG BINARY columns and LONG VARCHAR and LONG BINARY variables of any size of data. Currently, a SQL variable can hold up to 2GB - 1 in length.

See *Reference: Building Blocks, Tables, and Procedures > SQL Functions > Alphabetical List of Functions > LOCATE Function [String]*.

OCTET_LENGTH Function

The **OCTET_LENGTH** function returns an unsigned 64-bit value containing the byte length of the large object column or variable parameter.

Syntax

```
OCTET_LENGTH( column-name )
```

Parameters

large-object-column – the name of a LONG VARCHAR or LONG BINARY column or variable.

Usage

- If the argument is NULL, **OCTET_LENGTH** returns NULL.
- **OCTET_LENGTH** supports all Sybase IQ data types and LONG VARCHAR and LONG BINARY variables of any size of data. Currently, a SQL variable can hold up to 2GB - 1 in length.

PATINDEX Function

The **PATINDEX** function returns a 64-bit unsigned integer containing the position of the first occurrence of the specified pattern in a LONG VARCHAR column or variable. For CHAR and VARCHAR columns, **PATINDEX** returns a 32-bit unsigned integer position.

Syntax

```
PATINDEX( '%pattern%', long-varchar-column )
```

Parameters

pattern – the pattern for which you are searching. This string is limited to 126 bytes for patterns with wildcards. If you omit the leading percent wildcard, **PATINDEX** returns one (1) if the pattern occurs at the beginning of the column value, and zero (0) if the pattern does not occur at the beginning of the column value. Similarly, if you omit the trailing percent wildcard, the pattern should occur at the end of the column value. The pattern uses the same wildcards as the **LIKE** comparison.

Patterns without wildcards—percent (%) and underscore (_)— can be up to 255 bytes in length.

long-varchar-column – the name of the LONG VARCHAR column or variable.

Usage

- All the positions or offsets, returned or specified, in the **PATINDEX** function are always character offsets and may be different from the byte offset for multibyte data.
- If the LONG VARCHAR cell being searched contains more than one instance of the string pattern, **PATINDEX** returns only the position of the first instance.
- If the column does not contain the pattern, **PATINDEX** returns zero (0).
- Searching for a pattern longer than 126 bytes returns NULL.
- Searching for a zero-length pattern returns 1.
- If any of the arguments is NULL, the result is zero (0).
- **PATINDEX** supports LONG VARCHAR variables of any size of data. Currently, a SQL variable can hold up to 2GB - 1 in length. **PATINDEX** does not support LONG BINARY variables or searching LONG BINARY columns.

See *Reference: Building Blocks, Tables, and Procedures > SQL Functions > Alphabetical List of Functions > PATINDEX Function [String]* and *Reference: Building Blocks, Tables, and Procedures > SQL Language Elements > Search Conditions > LIKE Conditions*.

SUBSTRING Function

The **SUBSTRING** function returns a variable-length character string of the LONG VARCHAR column or variable parameter. If any of the arguments are NULL, **SUBSTRING** returns NULL.

Syntax

```
{ SUBSTRING | SUBSTR } ( long-varchar-column, start [, length ] )
```

Parameters

long-varchar-column – the name of a LONG VARCHAR column or variable.

start – an integer expression indicating the start of the substring. A positive integer starts from the beginning of the string, with the first character at position 1. A negative integer specifies a substring starting from the end of the string, with the final character at position -1.

length – an integer expression indicating the character length of the substring. A positive length specifies the number of characters to return, starting at the *start* position. A negative length specifies the number of characters to return, ending at the *start* position.

Usage

SUBSTRING supports `LONG VARCHAR` variables of any size of data. Currently, a SQL variable can hold up to 2GB - 1 in length. **SUBSTRING** does not support `LONG BINARY` variables or searching `LONG BINARY` columns.

SUBSTRING64 Function

The **SUBSTRING64** function returns a variable-length character string of the large object column or variable parameter.

Syntax

```
SUBSTRING64 ( large-object-column, start [, length ] )
```

Parameters

large-object-column – the name of a `LONG VARCHAR` or `LONG BINARY` column or variable.

start – an 8-byte integer indicating the start of the substring. **SUBSTRING64** interprets a negative or zero *start* offset as if the string were padded on the left with "non-characters." The first character starts at position 1.

length – an 8-byte integer indicating the length of the substring. If *length* is negative, an error is returned.

Example

Values returned by **SUBSTRING64**, given a column named `col1` that contains the string ("ABCDEFG"):

`SUBSTRING64( col1, 2, 4 )` returns the string "BCDE"

`SUBSTRING64( col1, 1, 3 )` returns the string "ABC"

`SUBSTRING64( col1, 0, 3 )` returns the string "AB"

`SUBSTRING64( col1, -1, 3 )` returns the string "A"

Usage

- If any of the arguments are NULL, **SUBSTRING64** returns NULL.
- Nested operations of the functions **SUBSTRING64**, **SUBSTRING**, **SUBSTR**, **BYTE_SUBSTR**, and **BYTE_SUBSTR64** do not support large object columns or variables.
- **SUBSTRING64** supports searching `LONG VARCHAR` and `LONG BINARY` columns and `LONG VARCHAR` and `LONG BINARY` variables of any size of data. Currently, a SQL variable can hold up to 2GB - 1 in length.

Aggregate Function Support of Large Object Columns

Only the aggregate function **COUNT (*)** is supported for LONG BINARY and LONG VARCHAR columns.

The **COUNT DISTINCT** parameter is not supported. An error is returned if a LONG BINARY or LONG VARCHAR column is used with the **MIN**, **MAX**, **AVG**, or **SUM** aggregate functions.

User-Defined Function Support of Large Object Columns

Scalar and aggregate user-defined functions support large object data types LONG VARCHAR (CLOB) and LONG BINARY (BLOB) up to 4GB (gigabytes) as input parameters. LOB data types are not supported as output parameters.

User-defined function support requires a separately licensed Sybase IQ option. See the *User-Defined Functions Guide*.

Error and Warning Messages

Reference the error and warning messages that may be returned when you are working with unstructured data, including `LONG BINARY` and `LONG VARCHAR` columns.

Error 1000195

"LOAD specification '%2' only valid for column(s) having datatype '%3'. %1"

Item	Value
SQLCode	-1000195L
Constant	EMSG_BINARYFILE
SQLState	QDB95
ODBC 2 State	ERROR
ODBC 3 State	ERROR
Sybase Error Code	20855
Severity Code	14
Parameter 1	location of the exception
Parameter 2	type of load specification
Parameter 3	data type of column

Probable Cause

The named load specification in a **LOAD TABLE** statement is only valid for columns with the given data type.

Error 1000198

"Cannot create join index with table(s) having column(s) of datatype %2. %1"

Item	Value
SQLCode	-1000198L
Constant	EMSG_CANNOT_CREATE_JOIN_INDEX
SQLState	QDB98

Item	Value
ODBC 2 State	ERROR
ODBC 3 State	ERROR
Sybase Error Code	20858
Severity Code	14
Parameter 1	location of the exception
Parameter 2	data type of column

Probable Cause

This error is reported when you attempt to create a join index on a table that has one or more `LONG VARCHAR` or `LONG BINARY` data type columns.

The **JOIN INDEX** functionality is supported for most data types. There are a few data types, however, for which this functionality is not supported (for example, `LONG BINARY` and `LONG VARCHAR`).

Error 1000332

"Odd length of binary data value detected on column %2 %1"

Item	Value
SQLCode	-1000332L
Constant	EMSG_ODDNUMBER_NIBBLES
SQLState	QDD20
ODBC 2 State	ERROR
ODBC 3 State	ERROR
Sybase Error Code	21206
Severity Code	14
Parameter 1	location of the exception
Parameter 2	column name

Probable Cause

When binary data of hexadecimal format is loaded into a `LONG BINARY` column from a primary load file, Sybase IQ requires that the total number of hexadecimal digits is an even number.

This error is reported, if the cell value contains an odd number of hexadecimal digits. Input files for `LONG BINARY` loads should always contain an even number of hexadecimal digits.

Error 1001013

"Invalid data type comparison %1"

Item	Value
SQLCode	-1001013L
Constant	EMSG_TYPECOMPAREERROR
SQLState	QFA13
ODBC 2 State	ERROR
ODBC 3 State	ERROR
Sybase Error Code	20522
Severity Code	14
Parameter 1	location of the exception

Probable Cause

This error is reported if you attempt to search for a pattern in a `LONG BINARY` column using a **LIKE** predicate.

LIKE predicates are not supported on **LONG BINARY** (`BLOB`) columns.

Error 1001051

"Query returns %3 data > 2GB. Use %2 %1"

Item	Value
SQLCode	-1001051L
Constant	EMSG_LOB_OVER_2G_W_ARG
SQLState	QFA47
ODBC 2 State	ERROR
ODBC 3 State	ERROR
Sybase Error Code	21097
Severity Code	14

Item	Value
Parameter 1	SA parse source code line
Parameter 2	function recommended
Parameter 3	long binary or long varchar data type

Probable Cause
This error is reported when a query attempts to return a `LONG BINARY` or `LONG VARCHAR` value greater than 2 gigabytes.

Error 1001052

"Parameter %2 must be long binary/varchar type. %3 %1"

Item	Value
SQLCode	-1001052L
Constant	EMSG_ONLY_SUPPORT_LOB_W_ARG
SQLState	QFA48
ODBC 2 State	ERROR
ODBC 3 State	ERROR
Sybase Error Code	21098
Severity Code	14
Parameter 1	SA parse source code line
Parameter 2	LOB argument name
Parameter 3	recommended function name

Probable Cause
This error is reported when an invalid data type is used for a large object (LOB) function parameter.

Error 1001053

"Wrong number of parameters to function %2 %1"

Item	Value
SQLCode	-1001053L

Item	Value
Constant	EMSG_WRONG_NUM_PARAMS_W_ARG
SQLState	QFA49
ODBC 2 State	ERROR
ODBC 3 State	ERROR
Sybase Error Code	21099
Severity Code	14
Parameter 1	SA parse source code line
Parameter 2	function name

Probable Cause
This error is reported when a large object (LOB) function is passed an incorrect number of arguments.

Error 1001054

"You cannot specify long binary/varchar column in the ORDER/GROUP by clause or in an aggregate function. %1"

Item	Value
SQLCode	-1001054L
Constant	EMSG_LOB_NOT_ALLOWED_GROUP
SQLState	QFA50
ODBC 2 State	ERROR
ODBC 3 State	ERROR
Sybase Error Code	21100
Severity Code	14
Parameter 1	location of the exception

Probable Cause
This error is reported when you attempt to use a `LONG BINARY` column in an **ORDER BY**, **GROUP BY**, or aggregation clause.

Warning 1001055

"An error occurred loading %1 column, %2, for %3, rowid %4."

Item	Value
SQLCode	1001055L
Constant	EMSG_LOB_LOAD_ERROR_WARN
SQLState	QFA51
ODBC 2 State	OK
ODBC 3 State	OK
Sybase Error Code	21101
Severity Code	10
Parameter 1	long binary or long varchar data type
Parameter 2	FP index name
Parameter 3	secondary file name
Parameter 4	rowid

Probable Cause

This warning message is returned when an error is encountered either opening or reading a `LONG BINARY` or `LONG VARCHAR` secondary file during a load operation.

This warning message is returned in the server log and the IQ message file when the `SECONDARY_FILE_ERROR` option is OFF and an error occurs.

Warning 1001056

"An error occurred extracting %1 column, %2, for %3."

Item	Value
SQLCode	1001056L
Constant	EMSG_LOB_EXTRACT_ERROR_WARN
SQLState	QFA52
ODBC 2 State	OK
ODBC 3 State	OK

Item	Value
Sybase Error Code	21102
Severity Code	10
Parameter 1	long binary or long varchar data type
Parameter 2	FP index name
Parameter 3	secondary file name

Probable Cause
This warning message is returned when you attempt to extract a `LONG BINARY` or `LONG VARCHAR` column and an error is encountered during the extract operation.

This warning message is returned in the server log and the IQ message file when the `SECONDARY_FILE_ERROR` option is OFF and an error occurs.

Error 1001057

"You must use BFILE() to extract %2 column. %1"

Item	Value
SQLCode	-1001057L
Constant	EMSG_LOB_EXTRACT_USE_BFILE
SQLState	QFA53
ODBC 2 State	ERROR
ODBC 3 State	ERROR
Sybase Error Code	21103
Severity Code	14
Parameter 1	location of the exception
Parameter 2	long binary or long varchar data type

Probable Cause
This error is reported when you execute a query containing a `LONG BINARY` or `LONG VARCHAR` column with the database option `TEMP_EXTRACT_NAME1` set ON and you did not specify the **BFILE** function.

Error 1001058

“The secondary file name, %2, is too long. %1”

Item	Value
SQLCode	-1001058L
Constant	EMSG_LOB_SECONDARY_FILE_TOOLONG
SQLState	QFA54
ODBC 2 State	OK
ODBC 3 State	OK
Sybase Error Code	21104
Severity Code	14
Parameter 1	location of the exception
Parameter 2	secondary file name

Probable Cause

This error is reported when the length of the **LOAD TABLE** secondary file pathname exceeds the pathname length limit of the operating system.

The action taken when this error is reported depends on the value of the `SECONDARY_FILE_ERROR` database option.

Error 1009189

“Text document exceeds maximum number of terms. Support up to 4294967295 terms per document. %1”

Item	Value
SQLCode	-1009189L
Constant	EMSG_MAXTERM_ERROR
SQLState	QSB84
ODBC 2 State	ERROR
ODBC 3 State	ERROR
Sybase Error Code	21210

Item	Value
Severity Code	14
Parameter 1	location of the exception

Probable Cause
Error from external prefilter or term breaker library.

Error 1012030

“for long binary/varchar Column ‘%2’, database page size of (%3) must be greater than %4. %1”

Item	Value
SQLCode	-1012030
Constant	EMSG_CAT_PAGESIZETOOSMALL
SQLState	QUA30
ODBC 2 State	ERROR
ODBC 3 State	ERROR
Sybase Error Code	20953
Severity Code	14
Parameter 1	location of the exception
Parameter 2	column number
Parameter 3	requested page size
Parameter 4	minimum allowed page size

Probable Cause
The database page size is too small to create a `LONG BINARY` or `LONG VARCHAR` column.

The database page size must be 128K or greater to create a `LONG BINARY` or `LONG VARCHAR` column.

Index

A

Adaptive Server Enterprise
  inserting IMAGE data 55
  inserting TEXT data 57
adding
  text configuration object 11, 12
  TEXT index 4, 5
ALTER TEXT CONFIGURATION 16, 17
  syntax 61
ALTER TEXT INDEX 6, 7
  syntax 63
altering
  text configuration object 16, 61
  TEXT index 6, 7, 63
analytics
  Unstructured Data Analytics Option 1

B

BEGIN PARALLEL IQ statement 66
BFILE function 49
  example 50
  extraction example 50
  extraction facility 49
  syntax 49
binary large object
  aggregate function support 78
  BIT_LENGTH function 70
  BLOB 55
  BYTE_LENGTH function 71
  BYTE_LENGTH64 function 71
  BYTE_SUBSTR function 71
  BYTE_SUBSTR64 function 71
  columns 55
  data type 55
  data type conversions 55
  description 55
  in queries 25, 36
  index support 59
  indexes 55, 59
  inserting IMAGE data 55
  LONG BINARY 55
  modifying 55
  monitoring performance 38
  OCTET_LENGTH function 75
  size 55
  sp_iqindexsize 47
  stored procedure support 39
  SUBSTRING64 function 77
  TEXT index 59
  user-defined function support 78
  variables 58
binary large object variable
  data type conversions 58
BIT_LENGTH function
  description 70
  syntax 70
BLOB
  aggregate function support 78
  binary large object 55
  BIT_LENGTH function 70
  BYTE_LENGTH function 71
  BYTE_LENGTH64 function 71
  BYTE_SUBSTR function 71
  BYTE_SUBSTR64 function 71
  CHARINDEX function 73
  columns 55
  data type 55
  data type conversions 55
  description 55
  exporting data 49
  function support 69
  in queries 25, 36
  index support 59
  indexes 55, 59
  inserting data 55
  inserting IMAGE data 55
  loading data 51
  LOCATE function 74
  LONG BINARY 55
  modifying 55
  monitoring performance 38
  OCTET_LENGTH function 75
  prefetching 55
  size 55
  sp_iqindexsize 47
  stored procedure support 39
  SUBSTRING64 function 77
  TEXT index 59
  updating data 55

user-defined function support 78
variable function support 69
variables 58
BLOB variable
data type conversions 58
breaking
terms 39, 40
BYTE_LENGTH64 function
description 71
syntax 71
BYTE_SUBSTR function
description 71
syntax 71
BYTE_SUBSTR64 function
description 71
syntax 71

C

char
breaking into terms¶ 39
CHAR_LENGTH function
description 72
syntax 72
CHAR_LENGTH64 function
description 73
syntax 73
character large object
aggregate function support 78
BIT_LENGTH function 70
BYTE_LENGTH function 71
BYTE_LENGTH64 function 71
BYTE_SUBSTR function 71
BYTE_SUBSTR64 function 71
CHAR_LENGTH function 72
CHAR_LENGTH64 function 73
CHARINDEX function 73
CLOB 56
columns 56
data type 56
data type conversions 57
description 55
in queries 25, 37
index support 55, 59
indexes 57, 59
inserting TEXT data 57
LOCATE function 74
LONG VARCHAR 56
modifying 57
OCTET_LENGTH function 75
PATINDEX function 75
size 56
sp_iqindexsize 47
stored procedure support 39
SUBSTRING function 76
SUBSTRING64 function 77
TEXT index 59
variables 58
WD index 57, 59
WORD index 57, 59
character large object variable
data type conversions 58
CHARINDEX function
description 73
syntax 73
CLOB
aggregate function support 78
BIT_LENGTH function 70
BYTE_LENGTH function 71
BYTE_LENGTH64 function 71
BYTE_SUBSTR function 71
BYTE_SUBSTR64 function 71
CHAR_LENGTH function 72
CHAR_LENGTH64 function 73
character large object 56
CHARINDEX function 73
columns 56
data type 56
data type conversions 57
description 55
exporting data 49
function support 69
in queries 25, 37
index support 55, 59
indexes 57, 59
inserting data 57
inserting TEXT data 57
loading data 51
LOCATE function 74
LONG VARCHAR 56
modifying 57
OCTET_LENGTH function 75
PATINDEX function 75
size 56
sp_iqindexsize 47
stored procedure support 39
SUBSTRING function 76
SUBSTRING64 function 77
TEXT index 59

updating data 57
user-defined function support 78
variable function support 69
variables 58
WD index 57, 59
WORD index 57, 59
CLOB variable
data type conversions 58
compatibility
with Adaptive Server Enterprise 2
with ASE 2
with SA 2
with SQL Anywhere 2
compression of LOB data 44, 45
changing settings 45
displaying settings 46
CONTAINS
table expressions 26, 33
CONTAINS conditions
TEXT index 26
CONTAINS examples
text configuration object 18
contains-expression
FROM clause 26
CREATE TEXT CONFIGURATION 11, 12
syntax 64
CREATE TEXT INDEX 4, 5
syntax 65
creating
text configuration object 11, 12, 64
TEXT index 4, 5, 65

D

data compression of LOB 44, 45
changing settings 45
displaying settings 46
data type
BLOB 55
CLOB 56
LONG BINARY 55
LONG VARCHAR 56
data type conversion
LONG BINARY to BINARY 55
LONG BINARY to VARBINARY 55
LONG BINARY variables 58
LONG VARCHAR to CHAR 57
LONG VARCHAR to VARCHAR 57
database option
ENABLE_LOB_VARIABLES 58
MAX_PREFIX_PER_CONTAINS_PHRASE 20
TEXT_DELETE_METHOD 8
dbspace
modifying 7
TEXT indexes 7
default_char 10
default_nchar 10
deleting
text configuration object 17
TEXT index 7, 8
disabling
compression 44, 45
external library 22
DROP TEXT CONFIGURATION 17
syntax 66
DROP TEXT INDEX 7, 8
syntax 67
dropping
text configuration object 17, 66
TEXT index 7, 8, 67

E

editing
text configuration object 16
TEXT indexes 6, 7
ENABLE_LOB_VARIABLES option 58
enabling
compression 44, 45
external library 22
END PARALLEL IQ
CREATE TEXT INDEX 66
error messages 79
BFILE extract error 85
CREATE JOIN INDEX error 79
data greater than 2GB error 81
error 1000195 79
error 1000198 79
error 1000332 80
error 1001013 81
error 1001051 81
error 1001052 82
error 1001053 82
error 1001054 83
error 1001057 85
error 1001058 86
error 1009189 86
error 1012030 87
invalid data type comparison error 81

invalid data type error 82
LOAD specification error 79
number of terms error 86
odd length error 80
ORDER BY or GROUP BY error 83
page size error 87
secondary file name error 86
wrong number of parameters error 82
examples
text configuration object 17, 18
exporting
BFILE example 50
BFILE function 49
BLOB 49
CLOB 49
large object data 49
LOB 49
LONG BINARY 49
exporting LONG VARCHAR 49
external libraries 21
disabling 22
enabling 22
identifying 43
listing 44
multiplex servers 22
restrictions 22
unloading 23, 43
extraction facility
BFILE function 49

F

FROM clause
CONTAINS 26, 33
contains-expression 26
syntax 26
full text searching 2, 3, 25
types 25
functions
BFILE 49
BFILE example 50
BIT_LENGTH 70
BYTE_LENGTH 71
BYTE_LENGTH64 71
BYTE_SUBSTR 71
BYTE_SUBSTR64 71
CHAR_LENGTH 72
CHAR_LENGTH64 73
CHARINDEX 73
for BLOB 69
for BLOB variables 69
for CLOB 69
for CLOB variables 69
for LOB 69
LOCATE 74
LONG BINARY aggregate support 78
LONG BINARY user-defined function support 78
LONG VARCHAR aggregate support 78
LONG VARCHAR user-defined function support 78
OCTET_LENGTH 75
PATINDEX 75
SUBSTRING 76
SUBSTRING64 77
fuzzy search 34

I

identifying
external libraries 43, 44
IMAGE data
inserting from ASE 55
inserting into LONG BINARY 55
immediate refresh 8, 41, 42
index
listing 6
indexes
binary large object 55, 59
BLOB 55, 59
character large object 57, 59
CLOB 57, 59
Containment 57, 59
full text searching 2, 25
large object data 59
LOB 59
LONG BINARY 55, 59
LONG VARCHAR 57, 59
refreshing 8
TEXT 3, 59
WD 57, 59
WORD 57, 59
inserting
BLOB 55
CLOB 57
large object data 55, 57
LOB 55, 57
LONG BINARY 55
LONG VARCHAR 57

instances
external libraries¶ 44

L

large object data
exporting 49
index support 59
inserting 55, 57
loading 51
updating 55, 57
libraries, external 10
licensing 1
list
external libraries 44
text configuration object 15
TEXT index 6
LOAD TABLE
example 52
extended syntax 51
primary load file 51
secondary load file 51
loading
BLOB 51
CLOB 51
controlling errors 52
large object data 51
LOAD TABLE example 52
LOB 51
LONG BINARY 51
LONG VARCHAR 51
SECONDARY_FILE_ERROR option 52
stripping trailing blanks 53
TRIM_PARTIAL_MBC option 53
truncating character data 53
LOB
exporting data 49
function support 69
index support 59
inserting data 55, 57
introduction 1
loading data 51
typical sources 1
updating data 55, 57
user-defined function support 78
LOB compression
changing settings 45
disabling 44, 45
displaying settings 46
enabling 44, 45
LOB variables
data type conversion 58
LOCATE function
description 74
syntax 74
LONG BINARY
aggregate function support 78
binary large object 55
BIT_LENGTH function 70
BLOB 55
BYTE_LENGTH function 71
BYTE_LENGTH64 function 71
BYTE_SUBSTR function 71
BYTE_SUBSTR64 function 71
CHARINDEX function 73
columns 55
data type conversions 55
DELETE 55
exporting data 49
in queries 25, 36
index support 59
indexes 55, 59
INSERT 55
inserting data 55
inserting IMAGE data 55
LOAD TABLE 55
loading data 51
LOCATE function 74
modifying 55
monitoring performance 38
OCTET_LENGTH function 75
SELECT...INTO 55
size 55
sp_iqindexsize 47
stored procedure support 39
SUBSTRING64 function 77
TEXT index 59
TRUNCATE 55
UPDATE 55
updating data 55
user-defined function support 78
variables 58
LONG BINARY variable
data type conversions 58
LONG VARCHAR
aggregate function support 78
BIT_LENGTH function 70
BYTE_LENGTH function 71
BYTE_LENGTH64 function 71

BYTE_SUBSTR function 71
BYTE_SUBSTR64 function 71
CHAR_LENGTH function 72
CHAR_LENGTH64 function 73
character large object 56
CHARINDEX function 73
CLOB 56
columns 56
data type conversions 57
DELETE 57
exporting data 49
in queries 25, 37
index support 59
indexes 57, 59
INSERT 57
inserting data 57
inserting TEXT data 57
LOAD TABLE 57
loading data 51
LOCATE function 74
modifying 57
OCTET_LENGTH function 75
PATINDEX function 75
SELECT...INTO 57
size 56
sp_iqindexsize 47
stored procedure support 39
SUBSTRING function 76
SUBSTRING64 function 77
TEXT index 59
TRUNCATE 57
UPDATE 57
updating data 57
user-defined function support 78
variables 58
WD index 57, 59
WORD index 57, 59
LONG VARCHAR variable
data type conversions 58

M

MAX_PREFIX_PER_CONTAINS_PHRASE option 20
messages
error 79
warning 79
modifying
dbspace 7
stoplist 16, 17
multibyte characters
TRIM_PARTIAL_MBC option 53
trimming partial 53
truncating on load 53
multiplex servers
external libraries 22

N

nchar
breaking into terms¶ 40
NGRAM
TEXT index search 34, 35
NGRAM TEXT index 9
creating 9
fuzzy search 9
non-fuzzy search 34, 35

O

OCTET_LENGTH function
description 75
syntax 75
option
ENABLE_LOB_VARIABLES 58
MAX_PREFIX_PER_CONTAINS_PHRASE 20
TEXT_DELETE_METHOD 8
Unstructured Data Analytics 1

P

PATINDEX function
description 75
syntax 75
performance monitor
binary large object 38
BLOB 38
LONG BINARY 38
prefetching 55
prefilter library 10, 12, 21
maximum term length 14
minimum term length 13
restrictions 22
prefix
term limit 20

Q

queries
- binary large object 25, 36
- BLOB 25, 36
- character large object 25, 37
- CLOB 25, 37
- LONG BINARY 25, 36
- LONG VARCHAR 25, 37

R

refresh
- immediate¶ 41, 42

refreshing
- immediate 8
- TEXT index 8

restrictions
- external library 22
- TEXT index 6

S

sa_char_terms stored procedure 39
sa_external_library_unload stored procedure 23, 43
sa_list_external_library stored procedure 22, 44
sa_nchar_terms stored procedure 40
sa_text_index_stats stored procedure 41
sa_text_index_vocab stored procedure 42
search conditions
- CONTAINS clause 33
- CONTAINS conditions 26

searching
- CONTAINS clause 33
- CONTAINS conditions 26
- full text 2, 25
- fuzzy 34
- NGRAM TEXT index 34, 35
- non-fuzzy 34, 35
- prefix term limit 20
- TEXT indexes 3

SECONDARY_FILE_ERROR option 52
SELECT statement
- FROM clause syntax 26

sp_iqindexsize
- binary large object 47
- BLOB 47
- character large object 47
- CLOB 47
- LONG BINARY 47
- LONG VARCHAR 47

sp_iqindexsize stored procedure 46
sp_iqsetcompression stored procedure 45
sp_iqshowcompression stored procedure 46
standards 2
statistics
- TEXT indexes 41, 42

stoplist 10, 14
- modifying 16, 17

stored procedures
- binary large object 39
- BLOB 39
- character large object 39
- CLOB 39
- LONG BINARY 39
- LONG VARCHAR 39
- sa_char_terms 39
- sa_external_library_unload 43
- sa_list_external_library 44
- sa_nchar_terms 40
- sa_text_index_stats 41
- sa_text_index_vocab 42
- sp_iqindexsize 46
- sp_iqsetcompression 45
- sp_iqshowcompression 46

STRING_RTRUNCATION option 56, 57
SUBSTRING function
- description 76
- syntax 76

SUBSTRING64 function
- description 77
- syntax 77

T

term breaker
- algorithm 12
- setting 12
- text configuration object 12

term breaker library 10, 12–14, 21
- restrictions 22

terms
- breaking 39, 40
- full text searching 25
- ignoring 14
- maximum length 14
- minimum length 13
- row position 39, 40

stoplist 16, 17
TEXT index 39
text configuration object 10
altering 16, 61
CONTAINS examples 18
creating 11, 12, 64
defaults 10
dropping 17, 66
examples 17, 18
listing 15
settings 12–14
term breaker 12
TEXT data
inserting from ASE 57
inserting into LONG VARCHAR 57
TEXT index 3, 59
altering 63
changing dbspace 7
comparison with WD index 3
CONTAINS conditions 26
creating 4, 5, 65
creating NGRAM 9
deleting rows 8
dropping 7, 8, 67
editing 6, 7
fuzzy search 34
listing 6
NGRAM 9
non-fuzzy search 34, 35
refreshing 8
restrictions 6
statistics 41, 42
terms 39
text configuration object 10
text search
FROM contains-expression 26
TEXT_DELETE_METHOD option 8
TRIM_PARTIAL_MBC option 53

U

unloading
external libraries¶ 43
external library 23
Unstructured Data Analytics Option 1
licensing 1
updating
BLOB 55
CLOB 57
large object data 55, 57
LOB 55, 57
LONG BINARY 55
LONG VARCHAR 57
upgrading
existing LONG BINARY columns 55
LONG BINARY 55

V

variables
binary large object 58
binary large object conversion 58
BLOB 58
BLOB conversion 58
character large object 58
character large object conversion 58
CLOB 58
CLOB conversion 58
function support for BLOB 69
function support for CLOB 69
LONG BINARY 58
LONG BINARY conversion 58
LONG VARCHAR 58
LONG VARCHAR conversion 58

W

warning messages 79
extract operation warning 84
LOAD warning 84
warning 1001055 84
warning 1001056 84
WD index 59
comparison with TEXT index 3