Performing Full Text Searches on Chinese, Japanese, and Korean Data in SQL Anywhere 11

A whitepaper from Sybase iAnywhere

Authors: Hong Shi and Evguenia Eflov

Date: June 2009

This whitepaper was written in the context of SQL Anywhere 11.0.1.

SYBASE iAnywhere

Contents

SYBASE iAnywhere

Introduction

This document provides background information and procedures for performing a full text search on a database containing Chinese, Japanese, or Korean (CJK) data.

In a SQL Anywhere database with the default settings, you can use the NCHAR, NVARCHAR, or LONG NVARCHAR data types to store multi-byte character set (MBCS) data in a column. To store CJK data in a column using other data types such as CHAR, VARCHAR, and LONG VARCHAR, you must select one of the following database collations when you create the database:

Language	Windows collation	Unix collation
Japanese	932JPN	EUC_JAPAN
Korean	949KOR	EUC_KOREA
Chinese (Simplified)	936ZHO	EUC_CHINA
Chinese (Traditional - Hong Kong)	950ZHO_HK	EUC_TAIWAN
Chinese (Traditional - Taiwan)	950ZHO_TW	EUC_TAIWAN
Multiple languages	UTF8BIN	UTF8BIN
Multiple languages	UCA	UCA

The CHAR database collation is used to sort and compare data when a text index is created using a text configuration object that inherits settings from the default_char text configuration object. Text indexes that use the CHAR collation cannot be created on columns containing data of type NCHAR, NVARCHAR, or LONG NVARCHAR. Use the Initialization utility (dbinit) -z option to specify the CHAR database collation.

The NCHAR database collation is used to sort and compare data when a text index is created with a text configuration object that inherits settings from the default_nchar text configuration. Text indexes that use the NCHAR collation can be created on columns of any data type. When using a text index created with an NCHAR collation, data conversion might affect the performance of index updates if columns are of the type CHAR, VARCHAR, and LONG VARCHAR. The performance of text indexes on columns is unaffected on databases that use the same CHAR and NCHAR collation. Use the Initialization utility (dbinit) -zn option to specify the NCHAR database.

On a database using different CHAR and NCHAR collations, indexing the same set of data with a text index using the CHAR database collation and a text index using the NCHAR database collation might produce different vocabulary and full text search results.

Case and accent sensitivity affect comparisons between different CJK characters, even though the languages do not support these concepts directly. When you select the non-case-sensitive non-accent-sensitive collation tailoring, SQL Anywhere identifies different characters within the character set as comparable in accordance with the requirements of each language. For example, the Japanese equivalent Hiragana and Katakana characters are matched when using non-sensitive collation tailoring. For additional detail on the Unicode Collation Algorithm and how different character strings are compared and analyzed, see http://www.unicode.org/unicode/reports/tr10/.

Prerequisites

Before completing the procedures described in this document, it is recommended that you review the following information:

- "Understanding collations" in *Database Administration – International languages and character sets*:

  http://dcx.sybase.com/1101en/dbadmin_en11/supplied-collations-choosing-natlang.html

- "Unicode Collation Algorithm (UCA)" in *Database Administration – International languages and character sets*:

  http://dcx.sybase.com/1101en/dbadmin_en11/natlang-s-7003956.html

This information provides an overview of the methodology used to sort and group characters and recommendations for creating databases with specific collations.

The following software is required to complete the procedures in this document:

- SQL Anywhere 11.0.1

The appropriate Unicode fonts that contain the CJK character sets must be installed and configured.

To install the Unicode font sets on a Microsoft Windows operating system:

1. Click **Start** > **Control Panel**.
2. Double-click **Regional And Language Options**.
3. Click the **Languages** tab.
4. Click **Install Files For East Asian Languages**.
5. Click **OK**.

To install the Unicode font sets on a Unix operating system, refer to the instructions in the *readme_en.txt* file.

Setting up and performing a full text search on a database containing CJK data

Use the following procedure to set up an NGRAM text index and perform a full text search for CJK characters. For more information on a full text search with NGRAM text indexes, see "Tutorial: Performing a full text search on an NGRAM text index" in the SQL Anywhere Help.

The correct Asian characters required to execute queries in this tutorial are provided in the queries table in the *uca.db* and *utf8.db* sample databases. The sample databases are available for download from http://www.sybase.com/detail?id=1061814. The SQL statements in this tutorial fetch and use data from the sample database tables; the statements do not need to contain character constant strings and you do not need to type CJK characters.

The *utf8.db* sample database is used to demonstrate the behavior differences between different database collations. Two tables, cjkdata and queries, are included with both sample databases. The cjkdata table contains only multi-byte non-whitespace characters. The queries table includes sample CONTAINS queries.

For the UTF8BIN collation used in the CHAR collation of the utf8 database, the following non-whitespace single byte characters are not indexed and do not appear in the vocabulary:

```
! " # $ % & ( ) * + , - . / : ;< = > ? @ [ \ ] ^ _ ` { | } ~
```

For the UCA collation, the following single and multi-byte characters (in addition to whitespace characters) are not considered alphanumeric characters and do not appear in the dictionary:

- Unassigned characters
- Private Use Area (PUA) characters
- Characters defined as non-alphabetic by the Unicode standard

Start the sample databases and view sample data

1. Copy the sample database files to a new directory. For example, *c:/NGRAMsample/*.
2. At a command prompt, execute a command to start the uca and utf8 databases. For example:

   ```
   dbeng11 c:\NGRAMsample\uca.db c:\NGRAMsample\utf8.db
   ```

3. Connect to the uca and utf8 databases. For example:

   ```
   dbisql -c "ENG=uca;DBN=uca;UID=dba;PWD=sql"
   dbisql -c "ENG=uca;DBN=utf8;UID=dba;PWD=sql"
   ```

4. Execute the following statement in both Interactive SQL windows to view the sample table data in the uca and utf8 databases:

   ```
   SELECT *
   FROM "DBA"."cjkdata"
   ORDER BY "id";
   ```

   Both Interactive SQL windows display five rows of identical data.

5. Execute the following statement in both Interactive SQL windows to view the CONTAINS queries used in this tutorial:

   ```
   SELECT *
   FROM "DBA"."queries"
   ORDER BY "id";
   ```

   Both Interactive SQL windows display five rows of identical data.

Build and test the text configuration objects and text indexes with the NCHAR collation

1. Execute the following statements in both Interactive SQL windows to create two pairs of text configuration objects on the uca and utf8 databases. One text configuration object inherits settings from default_nchar and the other inherits settings from default_char.

   ```
   CREATE TEXT CONFIGURATION "myNcharNGRAMTextConfig1" FROM "default_nchar";
   CREATE TEXT CONFIGURATION "myNcharNGRAMTextConfig2" FROM "default_nchar";
   CREATE TEXT CONFIGURATION "myCharNGRAMTextConfig1" FROM "default_char";
   CREATE TEXT CONFIGURATION "myCharNGRAMTextConfig2" FROM "default_char";
   ```

2. Execute the following statements in both Interactive SQL windows to change the TERM BREAKER algorithm of all text configurations to NGRAM, and the MAXIMUM TERM LENGTH (N) to an appropriate

value. Typically, the NGRAM length for ideograms is 2 or 3 and 4 or 5 for word-based language models. So, for CJK data the recommended value for N is 2 or 3 and 4 or 5 for English data. Some types of queries might benefit from setting the value of N to 1. Queries that use terms longer than one character are slower when executed on a text index with an N value of 1. In this example one set of text configuration objects uses N=1, and the other uses N=2.

```
ALTER TEXT CONFIGURATION "myNcharNGRAMTextConfig1" TERM BREAKER NGRAM;
ALTER TEXT CONFIGURATION "myNcharNGRAMTextConfig1" MAXIMUM TERM LENGTH 1;
ALTER TEXT CONFIGURATION "myNcharNGRAMTextConfig2" TERM BREAKER NGRAM;
ALTER TEXT CONFIGURATION "myNcharNGRAMTextConfig2" MAXIMUM TERM LENGTH 2;
ALTER TEXT CONFIGURATION "myCharNGRAMTextConfig1" TERM BREAKER NGRAM;
ALTER TEXT CONFIGURATION "myCharNGRAMTextConfig1" MAXIMUM TERM LENGTH 1;
ALTER TEXT CONFIGURATION "myCharNGRAMTextConfig2" TERM BREAKER NGRAM;
ALTER TEXT CONFIGURATION "myCharNGRAMTextConfig2" MAXIMUM TERM LENGTH 2;
```

3. Execute the following statement in both Interactive SQL windows to create the text index ngram1:

```
CREATE TEXT INDEX "ngram1" ON "DBA"."cjkdata"( "value" )
   CONFIGURATION "DBA"."myNcharNGRAMTextConfig1";
```

   The text indexes in this tutorial use IMMEDIATE REFRESH. The use of IMMEDIATE REFRESH is not recommended in production databases unless the underlying data changes infrequently or the changes are minor.

4. Execute the following statement in both Interactive SQL windows to verify that the vocabulary for the text indexes in both databases is identical:

```
SELECT "term", "freq"
   FROM "sa_text_index_vocab" ( 'ngram1', 'cjkdata', 'DBA' )
   ORDER BY "freq" DESC, "term" ASC;
```

   These text indexes index 1-grams, so each term in the vocabulary is a single character. Multi-byte punctuation is interpreted as non-alphanumeric characters and is not indexed.

5. Execute the following statement in both Interactive SQL windows to create the full text search variable that is used to store the multi-byte query strings returned from the queries table:

```
CREATE VARIABLE "q" NVARCHAR( 100 )";
```

6. Execute the following statements in both Interactive SQL windows to search for a single alphanumeric character, wide B:

```
SELECT "contains_query" INTO "q"
   FROM "DBA"."queries"
   WHERE "id" = 1;
SELECT "q", "id", "value"
   FROM "DBA"."cjkdata"
   WHERE CONTAINS( "value", "q" );
```

   Both Interactive SQL windows display two rows containing the wide character B and the ids 1 and 5.

Execute the following statement in both Interactive SQL windows to search for a single non-alphanumeric character:

```
SELECT "contains_query" INTO "q"
   FROM "DBA"."queries"
   WHERE "id" = 2;
SELECT "q", "id", "value"
   FROM "DBA"."cjkdata"
   WHERE CONTAINS( "value", "q" );
```

No results are returned in either of the Interactive SQL windows because the wide question mark (?) is not recognized as an alphanumeric character by the UCA collation and is therefore not in the dictionary.

7. Execute the following statement in both Interactive SQL windows to verify that the data containing the wide question mark (?) character exists in the value column:

```
SELECT "q",*
   FROM "DBA"."cjkdata"
   WHERE "value" LIKE '%' || "q" || '%';
```

Both Interactive SQL windows display two rows containing the ids 1 and 4.

Build and test new text indexes with the CHAR collation

1. In both Interactive SQL windows, execute the following statement to drop the existing text index:

```
DROP TEXT INDEX "ngram1" ON "DBA"."cjkdata";
```

2. In both Interactive SQL windows, execute the following statement to create a new text index using the myCharNGRAMTextConfig1 text configuration object:

```
CREATE TEXT INDEX "ngram1" ON "DBA"."cjkdata"( "value" )
CONFIGURATION "DBA"."myCharNGRAMTextConfig1";
```

3. Execute the following statement in both Interactive SQL windows to verify that the text index vocabulary on the databases is different:

```
SELECT "term", "freq"
   FROM "sa_text_index_vocab" ( 'ngram1', 'cjkdata', 'DBA' )
   ORDER BY "freq" DESC, "term" ASC;
```

In the Interactive SQL window for the uca database, 19 rows of data are returned. This is the vocabulary generated for the sample data using the UCA collation, and it is identical to the vocabulary created in the previous procedure.

In the Interactive SQL window for the utf8bin database, 23 rows of data are returned because all wide punctuation characters are treated as alphanumeric characters according to the CHAR collation of the utf8 database which is UTF8BIN.

4. Execute the following statements in both Interactive SQL windows to search for a single alphanumeric character:

```
SELECT "contains_query" INTO "q"
   FROM "DBA"."queries"
   WHERE "id" = 1;
SELECT “q”, “id”, “value”
   FROM "DBA"."cjkdata"
   WHERE CONTAINS( "value", "q" );
```

   Both Interactive SQL windows display two rows containing the ids 1 and 5.

5. Execute the following statement in both Interactive SQL windows to search for a single multi-byte non-alphanumeric character:

```
SELECT "contains_query" INTO "q"
   FROM "DBA"."queries"
   WHERE "id" = 2;
SELECT “q”, “id”, “value”
   FROM "DBA"."cjkdata"
   WHERE CONTAINS( "value", "q" );
```

   The Interactive SQL window for the utf8 database returns rows with the ids 1 and 4. No results are returned in the Interactive SQL window for the uca database.

Build and test new 2-gram text indexes with the NCHAR collation

1. Execute the following statement in both Interactive SQL windows to delete the ngram1 text index from the uca and utf8 databases:

```
DROP TEXT INDEX "ngram1" ON "DBA"."cjkdata";
```

2. Execute the following statement in both Interactive SQL windows to create the new text indexes using the myNcharNGRAMTextConfig2 text configuration object:

```
CREATE TEXT INDEX "ngram2" ON "DBA"."cjkdata"( "value" )
   CONFIGURATION "DBA"."myNcharNGRAMTextConfig2";
```

3. Execute the following statement in both Interactive SQL windows to verify that the vocabularies for the new text index are identical in the uca and utf8 databases, and that the terms in the vocabulary contain two characters:

```
SELECT "term", "freq"
   FROM "sa_text_index_vocab" ( 'ngram2', 'cjkdata', 'DBA' )
   ORDER BY "freq DESC", "term ASC";
```

   Both Interactive SQL windows should now return 15 rows of data.

4. Execute the following statements in both Interactive SQL windows to search for a character that appears at the last position in a group of characters in the data:

```
SELECT "contains_query" INTO "q"
   FROM "DBA"."queries"
   WHERE "id" = 3;
SELECT “q”, “id”, “value”
   FROM "DBA"."cjkdata"
   WHERE CONTAINS( "value", "q" );
```

   No data is returned because the specified term length is 2 and the character appearing in the last position in a group of characters cannot be searched for directly.

5. Execute the following statements in both Interactive SQL windows to locate a character that appears in the last position in a group of characters:

```
SELECT "contains_query" INTO "q"
   FROM "DBA"."queries"
   WHERE "id" = 3;
SELECT LIST("term", ' OR ' ) INTO "q"
   FROM dbo.sa_text_index_vocab( 'ngram2', 'cjkdata', 'DBA' )
   WHERE term LIKE '%' || "q";
SELECT “q”, “id”, “value”
   FROM "DBA"."cjkdata"
   WHERE CONTAINS( "value", "q" );
```

   These statements demonstrate how text index vocabulary can be used to improve full text searches with NGRAMS. The Interactive SQL windows for both databases should now display rows with the ids 1 and 2.

6. Execute the following statement in both Interactive SQL windows to locate a single character that appears in the middle of a group of characters:

```
SELECT "contains_query" INTO "q"
   FROM "DBA"."queries"
   WHERE "id" = 1;
SELECT “q”, “id”, “value”
   FROM "DBA"."cjkdata"
   WHERE CONTAINS( "value", "q" );
```

   No rows are returned because the search parameter does not match any terms in the dictionary.

7. Execute the following statement in both Interactive SQL windows to search for the same single character that you searched for in step 11:

```
SELECT "contains_query" || '*' INTO "q"
   FROM "DBA"."queries"
   WHERE "id" = 1;
```

```
SELECT “q”, “id”, “value”
   FROM "DBA"."cjkdata"
   WHERE CONTAINS( "value", "q" );
```

Both Interactive SQL windows display 2 rows of data with the ids 1 and 5 because the search parameter matches the dictionary. This statement demonstrates how a prefix search can be used to improve full text searches with NGRAM text indexes.

8. Execute the following statement in both Interactive SQL windows to search for a term with the same length as the indexed terms:

```
SELECT "contains_query" INTO "q"
   FROM "DBA"."queries"
   WHERE "id" = 4;
SELECT "id", "value"
   FROM "DBA"."cjkdata"
   WHERE CONTAINS( "value", "q" );
```

9. Execute the following statements in both Interactive SQL windows to search for a term of length 5:

```
SELECT "contains_query" INTO "q"
   FROM "DBA"."queries"
   WHERE "id" = 5;
SELECT “q”, “id”, “value”
   FROM "DBA"."cjkdata"
   WHERE CONTAINS( "value", "q" );
```

Two rows of data with the ids 1 and 5 are returned for each database. A search for a phrase with four 2-gram terms will return identical results. For example, the query ‘abcde’ is equivalent to “ab bc cd de”.

Build and test new 2-gram text indexes with the CHAR collation

1. In both Interactive SQL windows, execute the following statement to delete the existing text index:

```
DROP TEXT INDEX "ngram2" ON "DBA"."cjkdata";
```

2. In both Interactive SQL windows, execute the following statement to create a new text index using the myCharNGRAMTextConfig2 text configuration object:

```
CREATE TEXT INDEX "ngram2" ON "DBA"."cjkdata"( "value" )
CONFIGURATION "DBA"."myCharNGRAMTextConfig2";
```

3. Execute the following statement in both Interactive SQL windows to verify that the vocabulary for the new text index is different in from that created using the UCA collation:

```
SELECT "term", "freq"
   FROM "sa_text_index_vocab"( 'ngram2', 'cjkdata', 'DBA' )
   ORDER BY "freq DESC", "term ASC";
```

The text index in the utf8 database is now built using UTF8BIN collation. With this collation, multi-byte punctuation characters appear in the vocabulary. The uca database returns 15 rows of data and the utf8 database returns 22 rows of data.

Remove the text index and text configuration objects

To execute the procedures in this document again, you must remove the text index and text configuration objects. To remove the text index and text configuration objects from the sample databases, execute the following statements in both Interactive SQL windows:

```
DROP TEXT INDEX "ngram2" ON "DBA"."cjkdata";
DROP TEXT CONFIGURATION "DBA"."myCharNGRAMTextConfig1";
DROP TEXT CONFIGURATION "DBA"."myCharNGRAMTextConfig2";
DROP TEXT CONFIGURATION "DBA"."myNcharNGRAMTextConfig1";
DROP TEXT CONFIGURATION "DBA"."myNcharNGRAMTextConfig2";
```

Summary

Upon completion of the procedures in this document, you created and tested multiple text indexes and performed full text searches on databases containing CJK data.

Copyright © 2009 iAnywhere Solutions, Inc. All Rights Reserved. Sybase, Afaria, SQL Anywhere, Adaptive Server Anywhere, MobiLink, UltraLite, and M-Business Anywhere are trademarks of Sybase, Inc. All other trademarks are property of their respective owners.