DBMaker

Full Text Search User's Guide

Version: 01.10

Document No: 50/DBM50-T06032010-01-FIDX
Author: DBMaker Support Team, Syscom Computer Engineering CO.
Publication Date: June 3, 2010

凌羣電腦
THE SYSCOM GROUP

Table of Content

1. Introduction

Welcome to the DBMaker Full Text Search User's Guide. This book will guide user understand every functions of DBMaker Full Text Index in detail and know how to use DBMaker Full Text Index.

1.1 About DBMaker Full Text Search

DBMaker is a powerful and flexible SQL Database Management System (DBMS) that provides excellent embedded Text Index capabilities. DBMaker Full-text indexes allow us to search fields containing any text for specific words and phrases. With version 4.0, DBMaker full-text search have been expanded to include exact phrase matching and Boolean search operators, which allows for even more complex control over search results.

DBMaker Full Text Index features include:

- **Powerful query function**
- **Embedded query**
- **Multi-language support convenient and easy to handle**

1.2 Document conventions

This book uses a standard set of typographical conventions for clarity and ease of use. The NOTE, Procedure, Example, and Command Line conventions also have a second setting used with indentation.

CONVENTION	DESCRIPTION
Italics	Italics indicate placeholders for information that must be supplied, such as user and table names. The word in italics should not be typed, but is replaced by the actual name. Italics also introduce new words, and are occasionally used for emphasis in text.
Boldface	Boldface indicates filenames, database names, table names, column names, user names, and other database schema objects. It is also used to emphasize menu commands in procedural steps.
KEYWORDS	All keywords used by the SQL language appear in uppercase when used in normal paragraph text.

CONVENTION	DESCRIPTION
SMALL CAPS	Small capital letters indicate keys on the keyboard. A plus sign (+) between two key names indicates to hold down the first key while pressing the second. A comma (,) between two key names indicates to release the first key before pressing the second key.
NOTE	Contains important information.
Procedure	Indicates that procedural steps or sequential items will follow. Many tasks are described using this format to provide a logical sequence of steps for the user to follow
Example	Examples are given to clarify descriptions, and commonly include text, as it will appear on the screen. Other forms of this convention include Prototype and Syntax.
CommandLine	Indicates text, as it should appear on a text-delimited screen. This format is commonly used to show input and output for dmSQL commands or the content in the dmconfig.ini file

Table 1-1 Document Conventions

2. Overview

In this chapter, we summarize the Text Index in different sides to let user know the basic of DBMaker Text Index. The details of DBMaker Text Index's function and usage will be introduced in other chapters later.

2.1 Concept

A Text Index is a mechanism that provides fast access to rows in a table that contains one or more words or phrases in columns containing text.

Text indexes contain a representation of all the text found in the columns they are based on, but the data is encoded and structured to make retrieval much faster than directly from the table. Once a user creates a text index for a table, its operation is transparent.

The DBMS uses the index to improve full-text query performance whenever possible.

2.2 Application of text index

Text index names must be unique for the table you are creating them on. Text index names have a maximum length of eighteen characters, and may contain numbers, letters, the underscore character, and other symbols. The first character may not be a number.

Text indexes can be built on all character type columns, including CHAR, VARCHAR, LONG VARCHAR, LONG VARBINARY, NCHAR, NVARCHAR, NCLOB, and FILE data types.

A table can have many text indexes and a text index can be built using multiple columns. A user may create text indexes by using either the JDBA Tool or use the *CREATE [SIGNATURE | IVF] TEXT INDEX* dmSQL command.

2.3 Text index types

DBMaker provides two different types of text index: signature and inverted-file. Signature text indexes are more efficient for small amount of data. Inverted-file text indexes usually consume more storage space but their response for queries is faster for large amounts of data.

We can create text indexes on single column or on multi columns. Besides these, we also can create text indexes on media types.

2.4 Operator

The string operators for DBMaker include MATCH, CONTAIN, CONTAINS and LIKE. Only the MATCH and CONTAINS operators can be applied to a text index search.

2.4.1 MATCH

This operator is used on all types of query in text index except on multi-column query.

2.4.2 CONTAIN & CONTAINS

'Contain' is used on single column query, it only can search one word in the record.**'Contains'** is used on multi-column query in text index.

2.4.3 BOOLEAN

The complex Boolean operations can also use in text index. This will be described in chapter **B**oolean **T**ext **S**earch in details.

2.5 Common Index and text index in DBMaker

The common ground for using common indexes or text indexes is to increase the performance of queries by quickly locating specific rows in a table without scanning the entire table. Only a table owner, a DBA, a SYSADM, or a user with the INDEX privilege on the table may create indexes. And you cannot create index on system tables and views.

The following table lists the differences between them:

Common index	Text index
Common indexes are stored in database they built in and managed by database. Without directory.	Signature text indexes are stored in the same tablespace as the column for which the index is being built.
	IVF text indexes are stored in separately directory and exhibit better performance for larger indexes. IVF text indexes managed by database they built in.
DBMaker will automatically update common indexes on tables when a new record is inserted, updated or deleted.	Text index is maintained manually to update records and keep data synchronous.
Index is a mechanism that provides fast access to specific rows in a table based on the values of one or more columns from the table (known as the key).	Purpose of text index is to implement fast query for large unstructured data.
Common index can't be created on large object column, such as file, long varchar, and long varbinary type, etc.	Text index can be created on all character type columns.
Common indexes are created not only on simple columns, but also on Expression columns or User Defined Function (UDF) columns.	Text indexes are created on single column or multiple columns. Text indexes also can search UDFs.
Without this function.	Text index can ignore the words defined as Stopword.

Table 2-1 the difference between common index and text index

3. Create Text Index

Text indexes can be built on all character type columns, including CHAR, VARCHAR, LONG VARCHAR, LONG VARBINARY, NCHAR, NVARCHAR, NCLOB, and FILE types. A table can have many text indexes and a text index can be built using a column or multiple columns. In this chapter we will introduce the method of create text index step by step.

3.1 Create signature text index

3.1.1 SYNTAX

A user can create signature text indexes by using the JDBA Tool, or the *CREATE TEXT INDEX* dmSQL command.

The CREATE SIGNATURE TEXT INDEX syntax is as following:

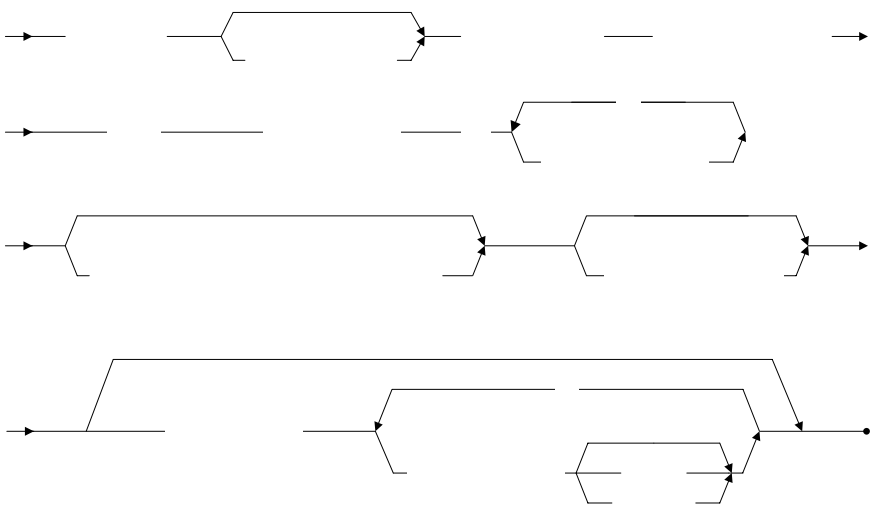


Figure 3-1 CREATE SIGNATURE TEXT INDEX syntax

Example:

```
dmSQL> CREATE TEXT INDEX ix_S ON book(subject);
```

Or use the syntax:

```
dmSQL> CREATE SIGNATURE TEXT INDEX ix_S ON book(subject);
```

NOTE: DBMaker creates signature text indexes if no text index method is specified in the command.

3.1.2 SIGNATURE TEXT INDEX PARAMETERS

DBMaker provides two parameters for conveniently configuring performance and storage size of signature text indexes.

- **Total text size (MB)** - the estimated total size of all source documents in megabytes (MB). The range is 1 ~ 200 and the default is 32. Please note the real total text size is not limited to 200 megabytes; if the size is larger than 200, set to 200. However, we strongly recommend using IVF text index to index very large amount of data for significantly better query performance.
- **Scale** - the expected index size-to-total text size ratio. If a user sets total text size to 20(MB) and expects the text index to use 10MB of storage, then he should set scale to 50 (50%). The larger scale, the better search performance. The range is 10 ~ 200 and the default value is 40 (40%).

A user can use the default setting as text index parameters. To get higher text index performance or to reduce the text index size, change the text index parameters. Set the parameters and monitor the text index performance, and then re-adjust the parameters.

3.1.3 EXAMPLE

Example1:

Create a signature text index use default parameter value.

Environment: create a text index **ix_intro** on the column **intro** of the table **book.**

```
dmSQL> CREATE SIGNATURE TEXT INDEX ix_intro ON book (intro)
dmSQL> def table book;
       create table SYSADM.BOOK (
       AUTHOR  CHAR(10) default null ,
       SUBJECT  CHAR(80) default null ,
       INTRO  LONG VARCHAR(40) default null ,
       CONTENT  FILE )
 in DEFTABLESPACE  lock mode page  fillfactor 100 ;
create text index IX_INTRO on SYSADM.BOOK ( INTRO ) total text size 32 mb scale 40;
```

We can see the default value of text index by def table.

Example2:

We can use the default setting as text index parameters, setting the parameters by manual can get the higher text index performance. The larger scale is, the better search performance.

Environment: create table text, the size of files in this table is near 150Mb, and then create signature text 3 times with same total text size and the scale are 100, 85 and 75. (The details of table text refer to Appendix) test as follow:

```
dmSQL> CREATE SIGNATURE TEXT INDEX ix_con ON text (con)
       TOTAL TEXT SIZE 200 MB
       SCALE 100;
Estimated elapsed time = 386.329000 seconds

dmSQL> select id from text where con match 'kate';
    ID
===========
         1
1 rows selected
```

```
Estimated elapsed time = 171.625000 seconds
dmSQL> drop text index ix_con from text;

dmSQL> CREATE SIGNATURE TEXT INDEX ix_con ON text (con)
       TOTAL TEXT SIZE 200 MB
       SCALE 85;
Estimated elapsed time = 202.109000 seconds

dmSQL> select id from text where con match 'kate';
    ID
===========
          1
1 rows selected
Estimated elapsed time = 181.781000 seconds
dmSQL> drop text index ix_con from text;

dmSQL> CREATE SIGNATURE TEXT INDEX ix_con ON text (con)
       TOTAL TEXT SIZE 200 MB
       SCALE 75;
Estimated elapsed time = 204.703000 seconds

dmSQL> select id from text where con match 'kate';
    ID
===========
          1
1 rows selected
Estimated elapsed time = 200.172000 seconds
```

Example3:

Total text size's range is 1-200M in signature text index. If total text size is larger than 200M, we'd better use IVF text index.

Environment: create table text1 and text2 (refer to Appendix), then create a signature text index **s_con** on the column con of the table text1. Insert records and the size is over 200Mb.

```
dmSQL> create text index s_con on text1(con) total text size 200 MB scale 100;
Estimated elapsed time = 505.547000 seconds

dmSQL> select Id from text1 where con match 'beach';

    ID
===========
          1

1 rows selected
Estimated elapsed time = 413.750000 seconds
```

Create IVF text index ivf_con on the column con of the table text2. The records in it are over 200MB.

```
dmSQL> create ivf text index ivf_con on text2(con);
Estimated elapsed time = 486.375000 seconds

dmSQL> select Id from text2 where con match 'beach';
```

```
    ID
===========
          1


1 rows selected
Estimated elapsed time = 0.391000 seconds
```

It's obvious that query under IVF text index is much faster than under Signature text index if the total text size over 200MB.

3.2 Create IVF text index

3.2.1 SYNTAX

IVF Text indexes can be built on all character type columns, including CHAR, VARCHAR, LONG VARCHAR, LONG VARBINARY, NCHAR, NVARCHAR, NCLOB, and FILE types.

A user may create text indexes by using either the JDBA Tool or the *CREATE IVF TEXT INDEX* dmSQL command.

The CREATE IVF TEXT INDEX syntax is as following:

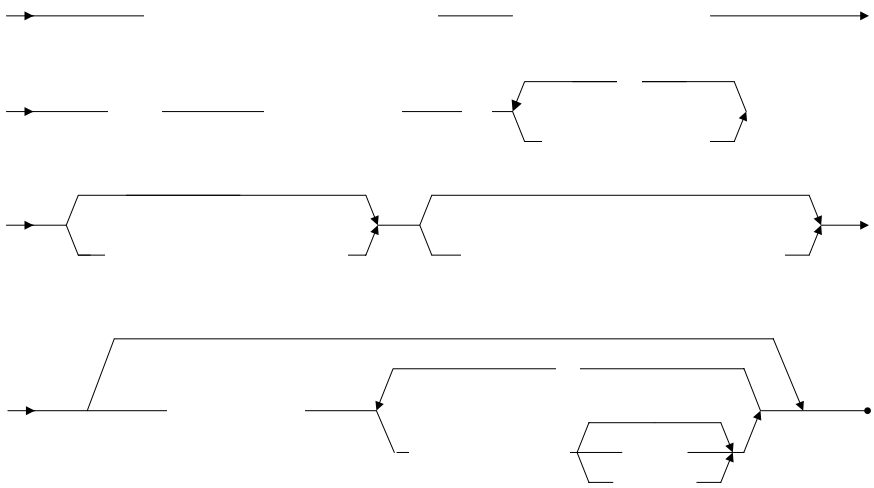


Figure 3-2 CREATE IVF TEXT INDEX syntax

Example1:

Environment: create Invert-file Text Index **ivf_content** on table **book** use dmSQL command:

```
dmSQL> CREATE IVF TEXT INDEX ivf_content ON book(content);
```

Example2:

Environment: create invert-file text index **ivf_info** on column **info** of table **card**:

```
dmSQL> CREATE IVF TEXT INDEX ivf_info ON card (info);
```

3.2.2 IVF TEXT INDEX PARAMETERS

There are two parameters for use in the creating IVF text index command:

- **Storage path** - the logical working directory where the inverted-files will reside in. Users should define the logical directory in the dmconfig.ini file. The default is the value of **DB_DbDir**, the database's home directory. The detail storage management and naming convention of inverted-file index will be described in the next section.
- **Total text size (MB)** - the approximate total size of documents will be indexed in the future. The unit of size is mega-byte (MB). Based on the size, DBMaker will decide how many partitions will be made. It may range between 1 MB to 10000 MB, and the default value is 500 MB.

Example1:

The parameters of IVF text index **ivf_info** created on table **card** were defined with default value.

```
dmSQL> def table card;
create table SYSADM.CARD (
 NUM  INTEGER default null ,
 INFO  LONG VARCHAR default null )
 in DEFTABLESPACE  lock mode page  fillfactor 100 ;
create ivf text index IVF_INFO on SYSADM.CARD ( INFO )
  storage path DB_DBDIR                              //default storage path
  total text size 500 mb  ;                          //default total text size
```

Example2:

Users manually define the parameters:

Step1: Users should define the logical directory in the dmconfig.ini file:

```
       [TEXTDB]
       DB_IVFDIR = C:\DBMaker\5.0\TEXTDB\new
```

NOTE: DB_IVFDIR is a user-defined keyword, not exists in DBMaker system. This keyword specifies the user-defined directory that the inverted-files reside in.

Step2: create IVF text index **ivf_detail** on column **detail** of table **card**:

```
dmSQL> create ivf text index ivf_detail on card(INFO)
       storage path DB_IVFDIR
       total text size 200 mb ;
```

We can find a new file create in the directory we defined. All the information of IVF text index saved in it. (Figure 3-3)

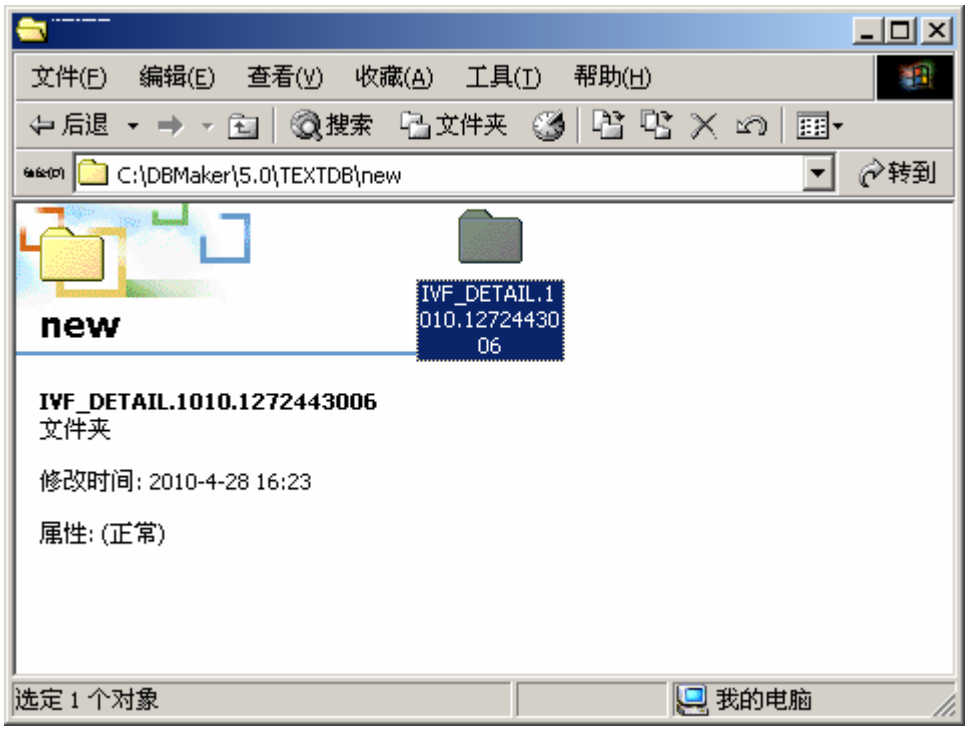


Figure 3-3

3.2.3 STORAGE PATH FOR IVF TEXT INDEXES

In addition to the working directory specified by the Storage path parameter, DBMaker will generate sub-directories in the working directory to manage different inverted-file indexes. Each inverted-file index has a unique time version, so DBMaker can use this property to generate a unique sub-directory to store index files (Figure 3-4). How to name the sub-directory is described later. This is also a limitation: these sub-directories and inverted-files cannot be rolled back when a user drops an inverted-file index.

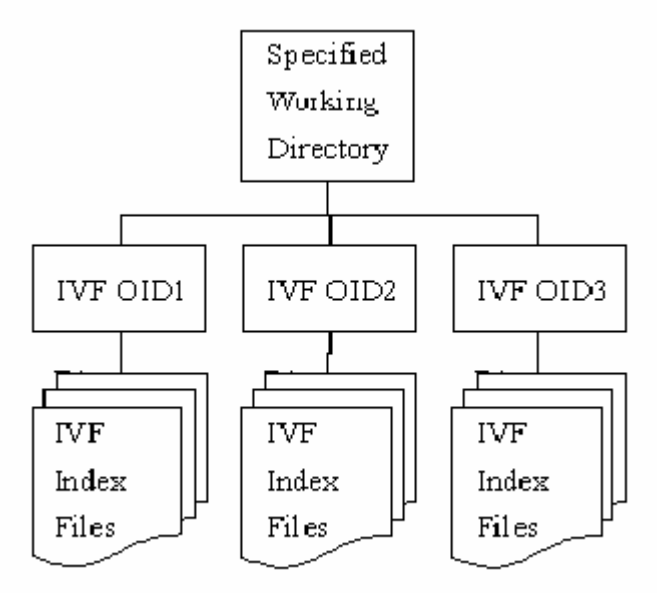


Figure 3-4

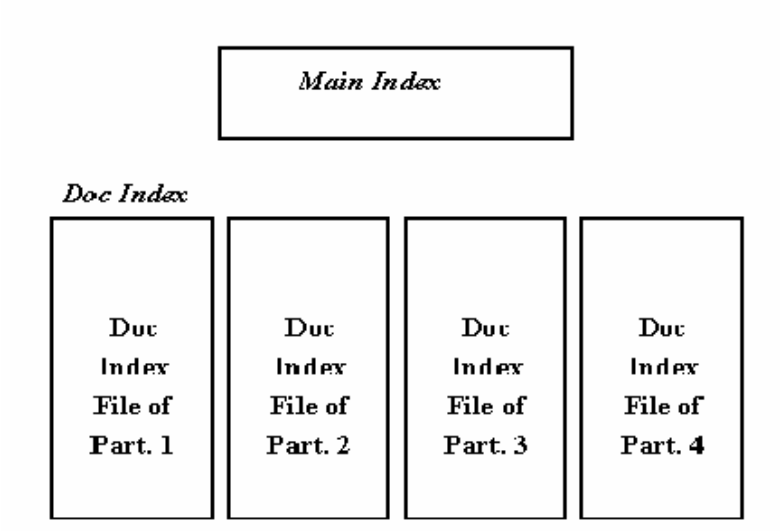


Figure 3-5 Inverted-file structure with four partitions

For example, if \DBMaker5.0\TEXTDB is a specified working directory, and we create an IVF under this working directory with the index name IVF_DOC, and the time version is 1010.1272443421, then a sub-directory IVF_DOC. 1010.1272443421 is created (Figure3-7). All inverted-files will reside in this sub-directory. There are three kinds of inverted-file with different terms: Single-Byte term, Uni-Gram term and Bi-Gram term, and each inverted-file have several partitions determined by text size (Figure 3-8).

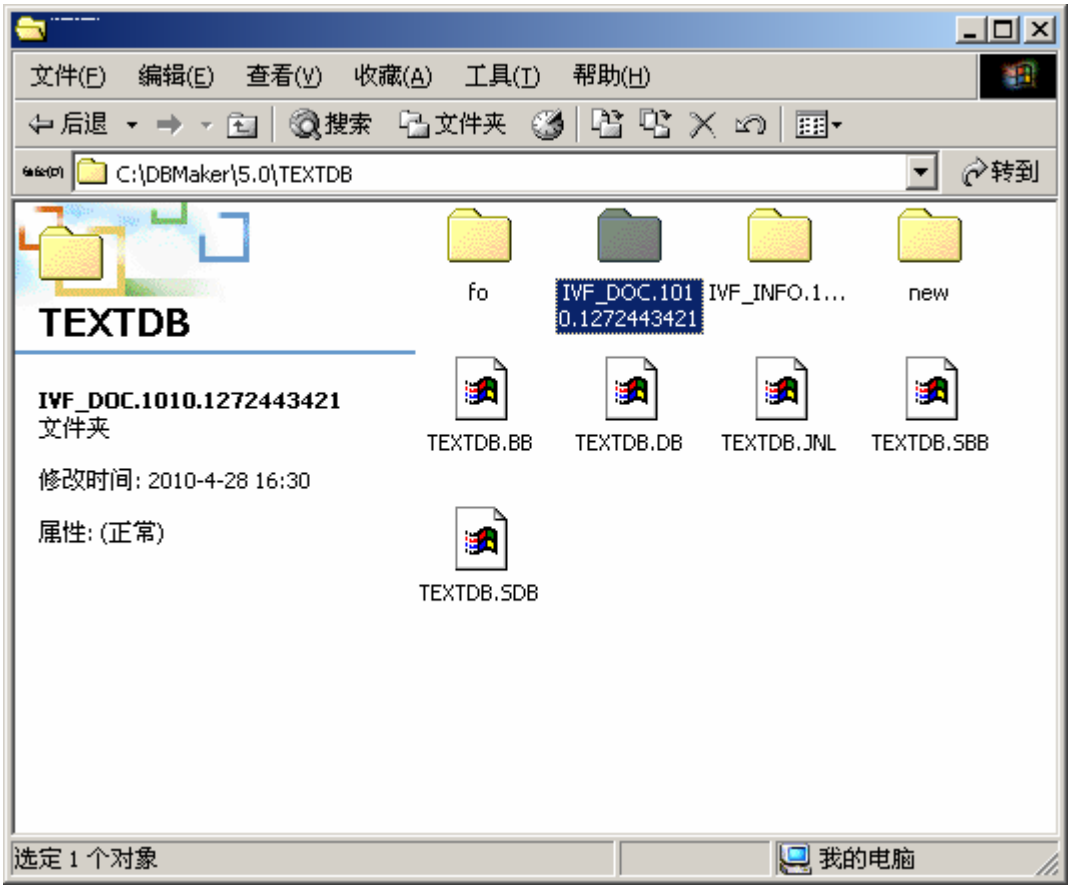


Figure 3-6

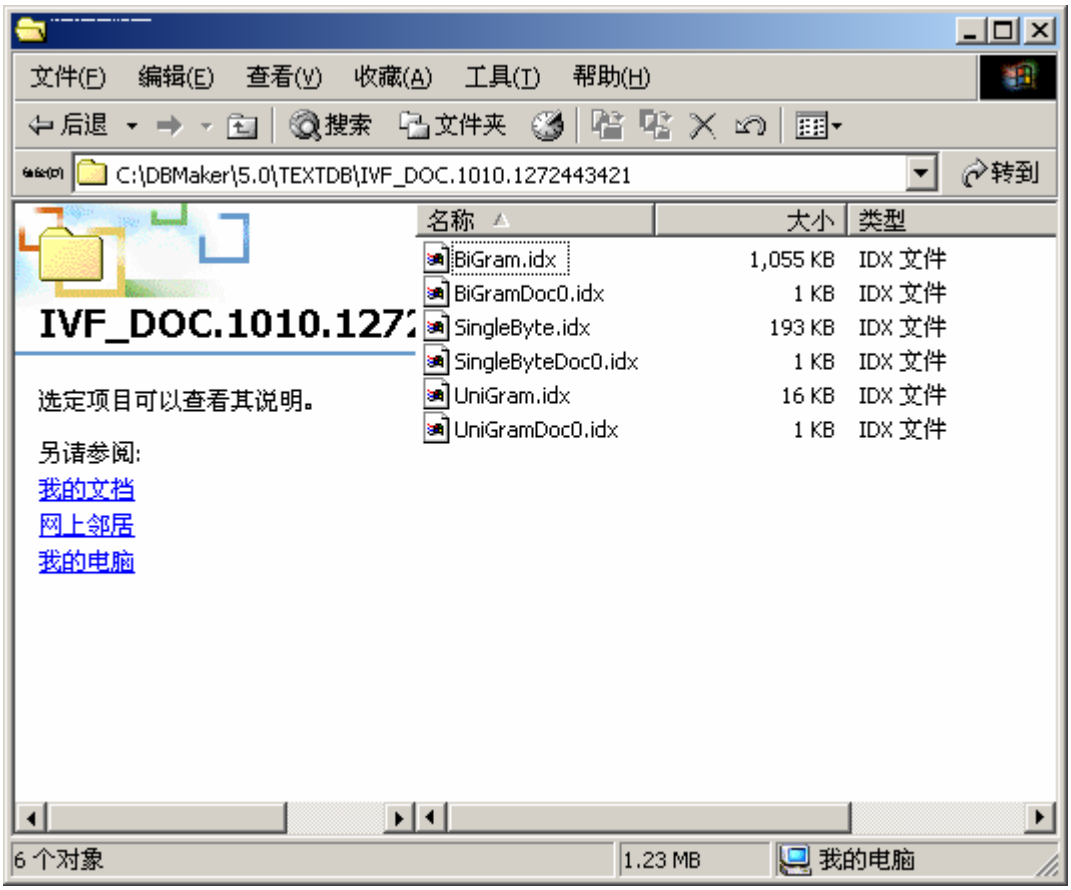


Figure 3-7

3.2.4 MEMORY CONSUME

The creation of inverted-file text indexes has a high memory resource requirement. DBMaker uses a simple rule to decide the maximum memory usage for creating text indexes. If DBMaker can not detect free memory or free memory resources less than 128MB, then the maximum memory usage will be 64MB. Otherwise, it will be half of the free memory resource. Users can specify the approximate upper bound of memory usage manually through dmconfig.ini by setting the keyword entry **DB_IFMem** equal to the number of megabytes (MB) desired.

3.2.5 EXAMPLE

Example1:

If we create IVF text index use user-defined Storage Path, we get different query efficiency with using default value.

Environment: create table text3 and text4, insert same records into 2 tables and create IVF text index on 2 tables, set both Total text size to 1000M, set text3's Storage Path is default value and text4's Storage Path is user defined.

```
dmSQL> create table text3( ID INTEGER, CON FILE);
dmSQL> create table text4( ID INTEGER, CON FILE);
```

The size of all files inserted is about 300Mb.

Then we set the Storage Path for text4:

```
[TEXTDB]
DB_IVFDIR = F:\NEW
```

Execute the command:

```
dmSQL> set showtime on;                    // display the query time
dmSQL> create ivf text index IVF_CON on text3 ( con )
       storage path DB_DBDIR
       total text size 1000 mb  ;
Estimated elapsed time = 580.266000 seconds

dmSQL> create ivf text index IVF_CON on text4 ( con )
       storage path DB_IVFDIR
       total text size 1000 mb  ;
Estimated elapsed time = 599.609000 seconds
dmSQL> select * from text3 where con match 'kate';

    ID                   CON
=========== ================================
          1 d0cf11e0a1b11ae10000000000000000
          2 d0cf11e0a1b11ae10000000000000000

2 rows selected
Estimated elapsed time = 0.438000 seconds

dmSQL> select * from text4 where con match 'kate';

    ID                   CON
=========== ================================
          1 d0cf11e0a1b11ae10000000000000000
          2 d0cf11e0a1b11ae10000000000000000

2 rows selected
Estimated elapsed time = 0.000000 seconds

dmSQL> select * from text3 where con match 'table';

    ID                   CON
=========== ================================
          3 d0cf11e0a1b11ae10000000000000000

1 rows selected
Estimated elapsed time = 0.063000 seconds

dmSQL> select * from text4 where con match 'table';

    ID                   CON
=========== ================================
          3 d0cf11e0a1b11ae10000000000000000

1 rows selected
Estimated elapsed time = 0.016000 seconds
```

Clearly, at the same situation, create IVF text index on text3 is faster than on text4, but if executing query, the result of text3 is slower than text4's. So we can create IVF text index on other free disk to get higher performance.

3.3 Choose factors between STI & IVFTI

Choosing between signature and inverted-file depends on the following factors:

- Index size -- the size of a signature index will not exceed the ratio set by the Scale parameter, which is 40% of total data size by default. The average size of inverted-file indexes is about 1.5 times of the data size, but could grow to 2 or even 3 times, depending on the property of data.
- Response time for queries -- on a modern personal computer with sufficient memory and processing power, users can expect sub-second response time from inverted-file indexes even the data size is gigabytes. Signature indexes will take longer to respond, especially when dealing with larger quantities of data.
- Integration with database -- unlike signature indexes which are stored as BLOB objects, inverted-file indexes are stored as external files, so, for example, users cannot roll back a drop inverted-file index operation.
- Try both types of text index to find which one suits the data's characteristics best. As a rule of thumb, for less than 100 megabytes of data, signature indexes respond to queries reasonably fast and usually take less storage space.

3.4 Creating Text Indexes on Multiple Columns

A text index can be built using multiple columns. Use CONTAINS and the concatenation operator (**||**) to perform multi-column text queries. Users can query on all columns of the index or just part of them. That is, the column list in a match query must be "contained" in the column list of a text index to use the text index. Users are also allowed to use the multi-column query syntax even if no text index is created on the column list, but no text index will be used.

Searching on multiple columns is logically equivalent to merging all columns' data then searching.

Example1:

To create an inverted-file text index **ivf_book** on columns **subject**, **intro** and **content** of the table **book:**

```
dmSQL> CREATE IVF TEXT INDEX ivf_book ON book(subject,intro,content);
dmSQL> SELECT author FROM book WHERE CONTAINS(subject || intro || content, 'text');
       //query all column
  AUTHOR
==========
apple
coco

2 rows selected
```

Query on partial columns:

```
dmSQL> SELECT author FROM book WHERE CONTAINS(subject || content, 'reagan');
AUTHOR
==========
apple

1 rows selected

dmSQL> SELECT author FROM book WHERE CONTAINS(subject, 'reagan');
AUTHOR
```

```
==========

0 rows selected

dmSQL> SELECT author FROM book WHERE subject MATCH 'error';
  AUTHOR
==========
berry
1 rows selected
```

Example2:

In this example, the column **subject** is included in the text index **ivf_book** but **author** is not, so this query will not use any text index.

```
dmSQL> SELECT author FROM book WHERE CONTAINS(subject || author, 'reagan');
       // no text index used;
  AUTHOR
==========

0 rows selected
```

NOTE: Performing the command above, we can also get the result of query. The difference is that we could get the better query performance if used the text index.

Example3:

This example illustrates the behavior of query on multiple columns.

```
dmSQL> CREATE TABLE fruit (sort char(20), others char (20), num serial);
dmSQL> INSERT INTO fruit VALUES('apple orange', 'banana grape', 1);
dmSQL> INSERT INTO fruit VALUES('grape orange', 'strawberry', 2);
dmSQL> CREATE TEXT INDEX ix_fruit on fruit (sort, others);
dmSQL> SELECT num FROM fruit WHERE CONTAINS (sort || others, 'apple');
 NUM
===========

1 rows selected
dmSQL> SELECT num FROM fruit WHERE CONTAINS (sort || others, 'orange & grape');
 NUM
===========
 1
 2
2 rows selected
```

3.5 Creating Text Indexes on Media Types

DBMaker's large object columns can register the media type. For example, a LONG VARBINARY column can know its content is a Microsoft Word file, so that DBMaker can invoke proper functions to perform a full-text search on a Microsoft Word document. DBMaker also provide media UDF for converting some media formats to pure text and a UDF (CHECKMEDIAFORMAT) to query the media format. Table summarizes the different media types available and the associated SQL commands.

MEDIA TYPE	DATA TYPE	FILE TYPE
Microsoft Word™,	MsWordType	MsWordFileType
HTML	HtmlType	HtmlFileType
XML	XmlType	XmlFileType
Microsoft PowerPoint	MsPPTType	MsPPTFileType
Microsoft Excel	MsExcelType	MsExcelFileType
PDF	PDFType	PDFFileType

Table 3-1 Media types and corresponding SQL commands

NOTE: Internally, MsWordType, MsPPTType, MsExcelType and PDFType are treated as a LONG VARBINARY object; HtmlType and XmlType are LONG VARCHAR objects and MsWordFileType, HtmlFileType, XmlFileType, MsPPTFileType, MsExcelFileType and PDFFileType are FILE objects.

3.5.1 FULL-TEXT SEARCH ON MEDIA-TYPE COLUMNS

Users can create a text index and perform a full-text search on the media type, but first the media format must be converted into pure text. DBMaker will not understand new media formats so conversion of the media format into pure text will not be possible nor will full-text search on the media format

DBMaker provides the following media UDFs for converting some media format to pure text:

DOC, XLS, PPT, HTM, PDF.

- DOCTOTXT(BLOB) RETURNS NCLOB;
- XLSTOTXT(BLOB) RETURNS NCLOB;
- PPTTOTXT(BLOB) RETURNS NCLOB;
- HTMTOTXT(CLOB) RETURNS CLOB;
- PDFTOTXT(BLOB) RETURNS NCLOB;

MATCH and CONTAINS can also be used for performing full-text search on media-type columns just as on regular text columns.

Example1:

Converts a PowerPoint document to a temporary BLOB containing the pure text of blob as Unicode.

```
dmSQL> create table tb_ppt(pptfile long varbinary);
dmSQL> insert into tb_ppt values(?);
dmSQL/Val> &e:\udf\pptfile\pfile.ppt;
dmSQL/Val>end;
dmSQL> select PPTTOTXT(pptfile) from tb_ppt;
```

Example2:

Create a table with a MS Word type column, insert some data, and search.

```
dmSQL> create table savefile(id int, sdoc MsWordFileType);
dmSQL> insert into savefile values(1, 'f:\abc.doc');
dmSQL> select id from savefile where sdoc match 'Kate';
ID
===========
1
```

```
1 rows selected;
```

NOTE: we insert user files into table, so we should add keyword **DB_UsrFO** into dmconfig.ini and set its value to 1 before insert, otherwise, an error will be show as:

```
dmSQL> insert into savefile values(2, 'c:\123.doc');
ERROR (316): [DBMaker] cannot link user file object when DB_USRFO = 0
```

A user can also create text indexes on media-type columns.

Example3:

Create a signature text index on the column **sdoc** of the table **savefile**, then search:

```
dmSQL> create text index ix_sdoc on savefile(sdoc);
dmSQL> select id from savefile where sdoc match 'twins';
ID
===========
1
1 rows selected;
```

3.5.2 CHECKING COLUMN DATA'S MEDIA TYPE

It is possible that a media-type column contains data of different types. DBMaker can verify the content during inserting or updating data to media-type columns; DBMaker provides a built-in function CHECKMEDIAFORMAT to check if a column's data match the column's media type. If types match, the function returns 1, otherwise returns 0.

Figure3-8: Syntax for CHECKMEDIAFORMAT

Following is the usage of the CHECKMEDIAFORMAT function:

```
CHECKMEDIAFORMAT (LONG VARBINARY, 'FORMAT') RETURNS INT.
```

DBMaker support the following media type formats: DOC, XLS, PPT, HTM and PDF.

- **DOC**: Microsoft Words Document
- **XLS**: Microsoft Excel Document
- **PPT**: Microsoft Power Point Document
- **HTM**: Hypertext Markup Language
- **PDF**: Portable Document Format

Example:

To check weather the media type format is correct:

```
dmSQL> Create table tb_checkmedia(note long varbinary);
dmSQL> insert into tb_checkmedia VALUES(?);
dmSQL/Val> &E:\DOCS\Media.doc;
dmSQL/Val> end;
dmSQL> select checkmediaformat(note,'doc') from tb_checkmedia;
```

It will returns 0,1, or NULL.

- Returns a 1 when the blob's content matches the specified media format
- Returns a 0 when the blob's content does not match the specified media format.

- Returns NULL when the blob is NULL.

When the table column defined with original media type (system domain) and the media format is not the correct format, the migration may fail. To resolve this problem, either remove the invalid media data or change the data type to the CLOB or the BLOB data type to avoid the step of checking for the correct media format.

3.6 User-Defined Stopword

As opposed to a keyword-based system, full-text retrieval software indexes every word in a document, with the exception of stopwords. Stopwords are those terms that full-text retrieval software is programmed to ignore during the indexing and retrieval processes, in order to prevent the retrieval of extraneous records. Generally, a stopword list includes articles, pronouns, adjectives, adverbs and prepositions (the, they, very, not, of, etc.) that are most common in the English language. You can also apply this rule to Chinese language or any double-byte-encoded text, for example:的, 呢, 啊, 哈, etc.

Note that the user-defined stopword can only use in IVF text index.

3.6.1 SEARCH PATH FOR STOPWORD LIST

DB_StpWd = <string>

This keyword indicates the name of the stopword list definition file that is put in the shared/stopword subdirectory of DBMaker's installation directory. The stopword list definition file is a pure text file, which would affect the text index result in DBMaker. This keyword is used when the database creating and retrieving text index. Without this keyword, database search pre-defined stopword list definition based on LCode.

Default value:

DB_LCode	Stopword List Definition
0 English (ASCII)	en.tab
1 Traditional Chinese (BIG5)	tw.tab
2 Japanese (Shift JIS + Half Corner)	en.tab
3 Simplified Chinese (GB)	cn.tab
4 Latin1 code (ISO-8859-1)	en.tab
5 Latin2 code (ISO-8859-2)	en.tab
6 Cyrillic code (ISO-8859-5)	en.tab
7 Greek code (ISO-8859-7)	en.tab
8 Japanese code（EUC-JP）	en.tab
9 Simplified Chinese (GB18030)	cn.tab

Table 3-2 Default value of stopwords

Valid range: file name of the user-defined stopword list definition file

See also: **DB_LCode**

Where to use: server side (only for creating and searching text index)

3.6.2 DATA RELATIONSHIP

We describe data relationship among configuration file, stopword list and DBMaker as following figure 3-8.

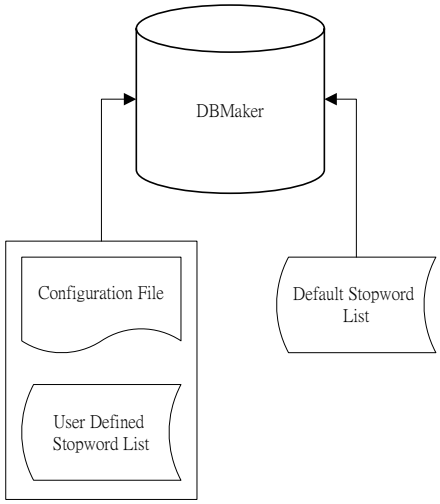


Figure 3-8 Data relationship

3.6.2.1 Default Stopword List

If users do not specify any configuration, DBMaker should load default stopword which has pre-defined file name based on LCODE. This is also compatible for old users.

DBMaker will search the pre-defined file in local directory, and installation directory respectively.

3.6.2.2 User Defined Stopword List

Users can specify their stopword list through configuration file, DBMaker would load the file when user create text index or retrieve from text index.

DBMaker will search the file in local/user specify directory, and installation directory respectively.

3.6.3 USER-DEFINED STOPWORD LIST DEFINITION

3.6.3.1 Keyword of Stopword List File

1. [BEGINE] : All lines after this section will be treated as stopword.
2. [SINGLE : Single byte stopword
3. [DOUBLE : Double byte stopword

3.6.3.2 Basic Rule and File Format of Stopword List

1. All must be printable characters or words.
2. Place characters or words in specific section '[SINGLE' or '[DOUBLE'.
3. No dedicated sequence for section '[SINGLE' or '[DOUBLE'.
4. All lines before section ‘[BEGIN]’ treat as comment.
5. All user defined stopword must be listed after section ‘[BEGIN]’.
6. Lines after ‘//’ treat as comment.
7. Every stopword must be separated by new line, and space will be ignored.

8. All stopwords in section '[SINGLE' must be single byte characters

9. All stopwords in section '[DOUBLE' must be double byte words

3.6.3.3 Stopword Usage

You can specify a stopword list through the configuration file **DB_STPWD**.

DBMaker loads the file when you create a text index or retrieve objects from the text index. That's mean, after reset keyword **DB_STPWD,** we should restart database, then create a newly text index or fully rebuild text index.

DBMaker searches the file in local or user specified directory, and installs the directory.

3.6.3.4 Disable Stopword List

We have two ways to disable stopword list

Rename pre-defined file name, or remove the file.

Define a non-exist stopword list in configuration file.

3.6.3.5 Example

Create a user-defined stopword list as:

Step1: the stopword list definition file is a pure text file, we could create list in a text file, such as WordPad.

Step2: write stopword in this file as rules 3.6.2.2 mentioned. For example:

```
[BEGIN]
[SINGLE-Byte Character Default Dummy Word]

//---- Number ----//
0
1
2
3
4
5
6
7
8
9

//---- English High Frequency Words ----//
A
AN
THE
ONE
SQL
IS

[DOUBLE-Byte Character Default Dummy Word]

//---- Chinese High Frequency Words ----//
一      // 1
```

```
是      // is
下      // down
也      // too
不      // no
的
```

Step3: save the file you set stopword as .tab

Step4: set dmconfig.ini, example:

```
[TEXTDB]
DB_LCode = 2                //this keyword is a choice, if you have defined stopword
StpWd = C:\DBMaker\5.0\stopword\slist.tab
```

Step5: restart database. If we reset the dmconfig.ini, we must restart database to get text index result we wanted.

Step6: create a text index or fully rebuild the text index you have already created.

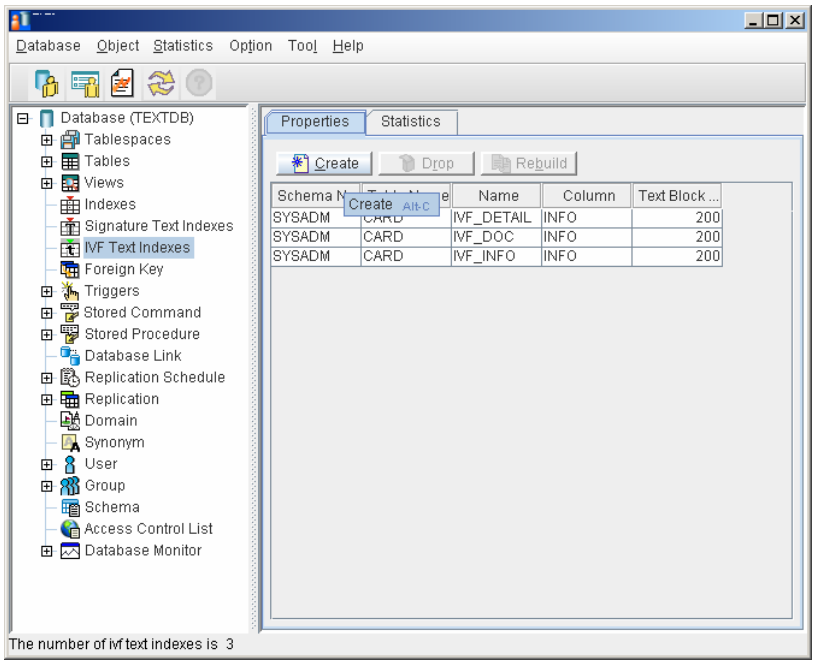
Search result:

```
dmSQL> select id from document where text match 'a';
WARNING (83): common words are included in the search pattern

   ID
========

0 rows selected

dmSQL> select id from document where text match 'a database';
WARNING (83): common words are included in the search pattern

   ID
========
       1
       2
       3

3 rows selected
```

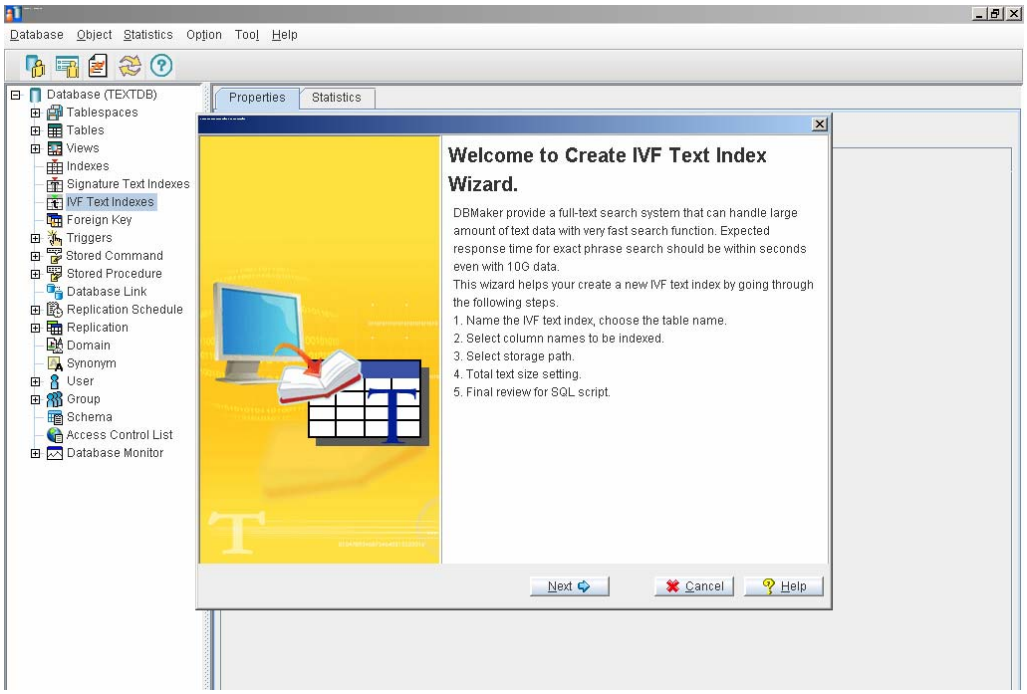
3.7 Create text index by using JDBA Tool

You can use JDBA Tool create IVF text index. Steps as followed:

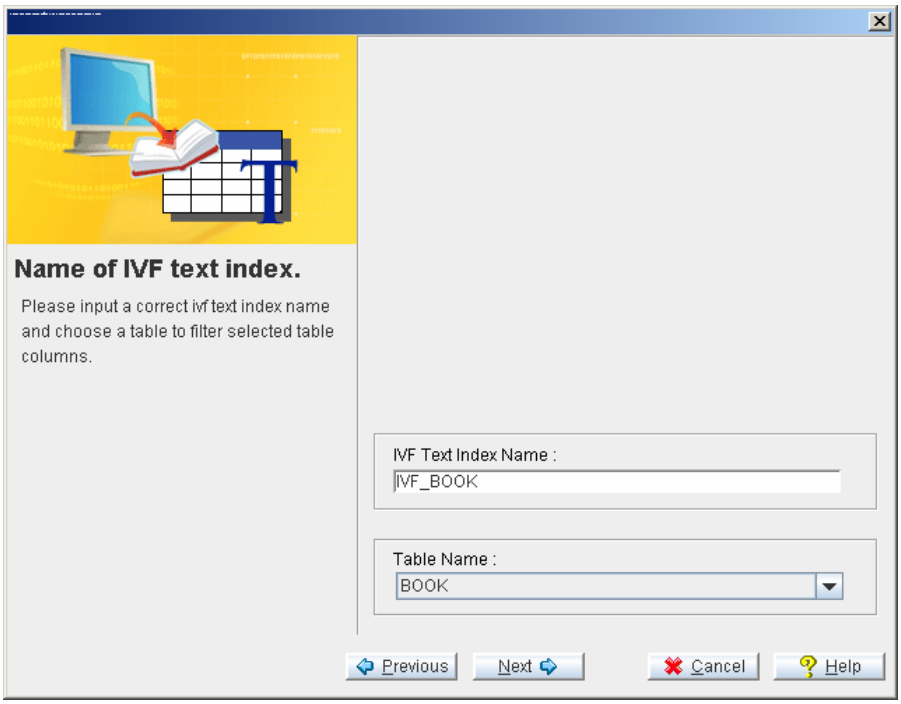
1. Open JDBA Tool; click the IVF text index under database directory. Then you can see all information of IVF text index created in this database via properties and statistics label.



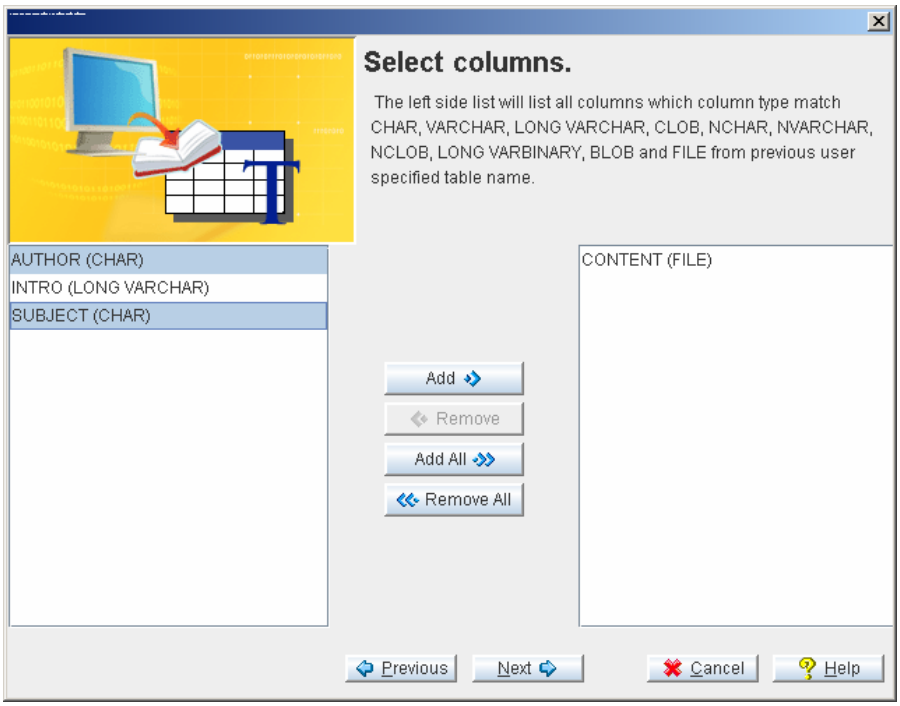
2. Click create button to create a new IVF text index. Then create text index by wizard.



3. Enter the name of text index and choose the table name you want create text index on, otherwise, the next button is disabled.



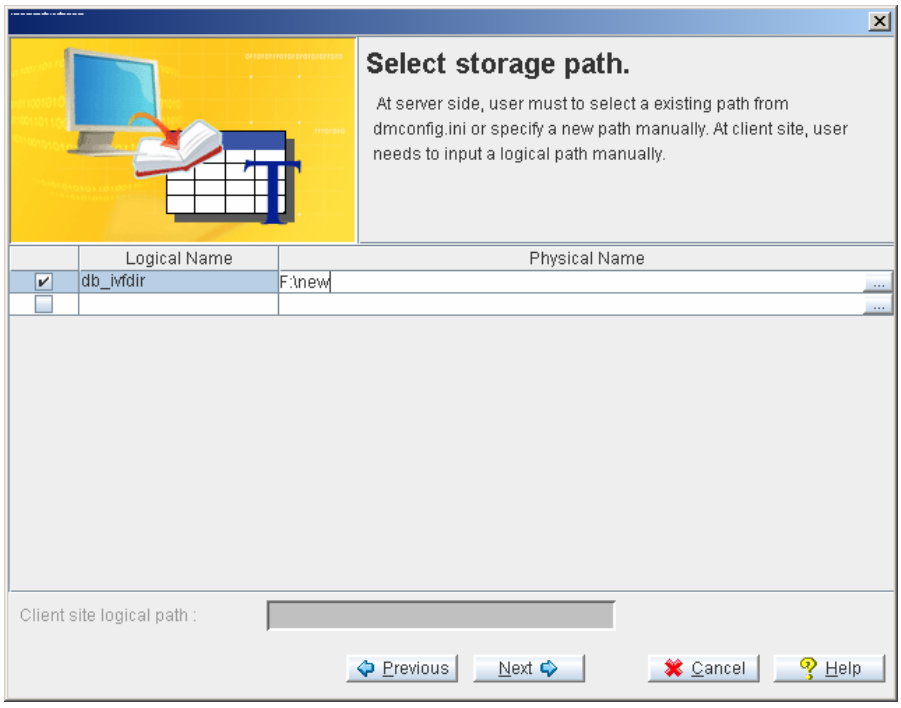
4. Select column from left list to right, you can choose one or more to create single column text index or multiple column text index. If you haven't selected, the next button is disabled.



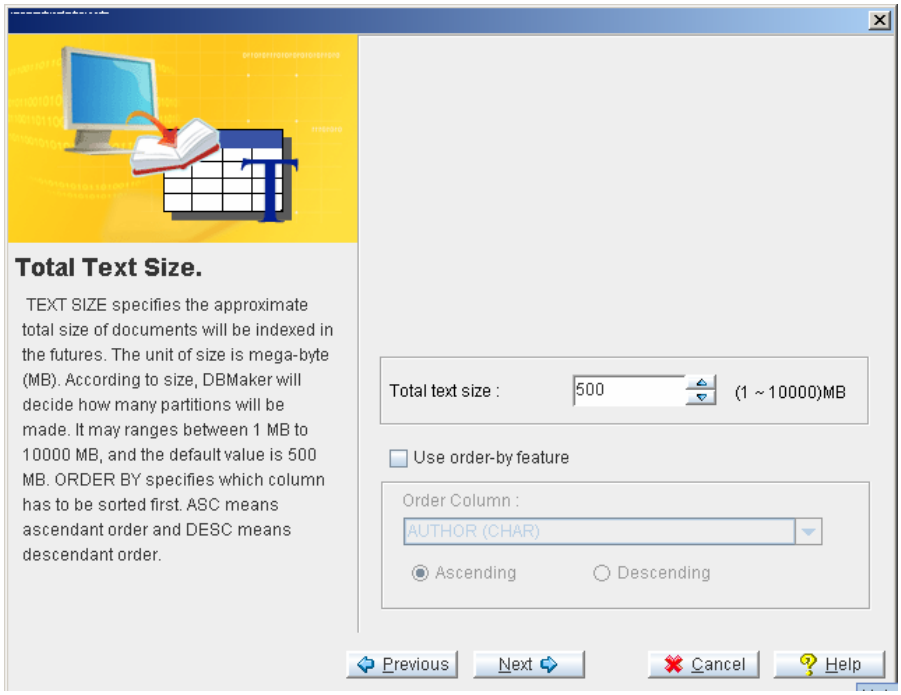
5. You can select an existing path or create new one to store IVF text index file.

   Create a new storage path, first create logical name, and select a path you want to put IVF text index file in as physical name, then make a tick in front frame. Now click next button to go on.

   If you haven't made a tick in front frame, the next button is disabled.



6. Choose total text size, click the **Next** button.



7. System will show the text index's SQL script you created at last, check it, if there isn't any error, click finish button. A successful information will pop up.

4. Rebuild Text index

When loading data into a table, DBMaker does not update any text indexes on that table, thus loading all data before creating a text index. If rows containing matching text entered into a table after the text index were created, the results will not be returned with the full-text query. To include these rows in the search results, rebuild the text index using the REBUILD TEXT INDEX command.

Unlike indexes, the text index will not simultaneously reflect table content if new records are inserted or old records are updated. Therefore, they need to be rebuilt manually. Data updated after the most recent rebuild will not be found during a text index search.

Example:

```
dmSQL> CREATE TABLE song (id int, name varchar(20));
dmSQL> INSERT INTO song VALUES(1,'Endless Love');
1 rows inserted
dmSQL> CREATE TEXT INDEX ix_name ON song(name);
dmSQL> INSERT INTO song VALUES(2,'Love Story');
1 rows inserted
dmSQL> SELECT * FROM song WHERE name MATCH 'love';
    ID              NAME
=========== ====================
          1 Endless Love
1 rows selected                 //2 records match search pattern, but only 1
retrieved.
```

The *REBUILD TEXT INDEX* command rebuilds an existing text index on a table. This updates the text index to include new data that has been added to the database since the text index was originally created. To execute the command to rebuild a text index on a table, you must be the table owner, have DBA or SYSADM security privileges, or have the INDEX privilege for that table.

4.1 Incrementally rebuild text indexes

4.1.1 SYNTAX

When only a few records have been changed, the *REBUILD TEXT INDEX <index_name> INCREMENTAL* command is the most rapid way to rebuild. This command will collect all new and updated records to let query exact.

The REBUID TEXT INDEX syntax is as following:

Figure 4-1 REBULD TEXT INDEX Syntax

The incremental option is the default setting for the REBUILD TEXT INDEX syntax. Incremental appends text into a table after the text index was created, thus making the text available to be returned with full-text query results.

Example:

The following rebuilds the text index named ix_intro on the table book:

```
dmSQL> rebuild text index ix_intro for book incremental;
```

4.1.2 EXAMPLE

Example1:

Use the e.g. above to rebuild a text index incrementally and display all the results of table song:

```
dmSQL> REBUILD TEXT INDEX ix_name FOR song incremental;
dmSQL> SELECT * FROM song WHERE name MATCH 'love';
    ID              NAME
=========== ====================
          1 Endless Love
          2 Love Story
          4 Hi! Be my love
3 rows selected
```

Example2:

Environment: we have already created signature text index ix on table document, executing the query:

```
dmSQL> select id from document where content match 'dbmaker';
    ID
===========
          1
          2
          4
3 rows selected

dmSQL> update document set content='DBMaker is a database.' where id=2;
1 rows updated
dmSQL> select id from document where content match 'dbmaker';
    ID
===========
          1
          4
2 rows selected

dmSQL> rebuild text index ix for document;
dmSQL> select id from document where content match 'dbmaker';

    ID
===========
          1
          4
          2
3 rows selected
```

After updating text index, we have changed the data in table, so the query result isn't correct, thus we should rebuild the text index created in table.

Example3:

Environment: use the table document used in example2; executing delete one record command instead of updating:

```
dmSQL> select id from document where content match 'dbmaker';
    ID
==========
         1
         2
         4
3 rows selected

dmSQL> delete from document where id=4;
1 rows deleted

dmSQL> select id from document where content match 'dbmaker';
    ID
==========
         1
         2
2 rows selected
```

DELETE command can't change the text index's query result. That's show us, not all changes in table can affect the result of text index.

4.1.3 SPECIALTY (RAPID WAY)

Incrementally rebuild text index which is the rapid way rebuilds the updated data incrementally.

This command will collect all new and updated records, build new signature vectors, and append the new vectors to the tail of the text index, But it can not release the signature vector space of old data, so it need more space to save the log.

If the new data less than half of the whole, we can use incrementally rebuild text index command.

4.2 Fully rebuild text indexes

4.2.1 SYNTAX

If a large number of documents are deleted or updated, use the *REBUILD TEXT INDEX <index_name> FULL* command to fully rebuild a text index with its original type (signature or inverted file) and parameters.

The full option rebuilds an entire text index by dropping and rebuilding the index based on a new full-text query.

You can fully build text index by using syntax as follow:

Figure 4-2 REBULD TEXT INDEX Syntax

Example:

```
dmSQL> REBUILD TEXT INDEX ix_name FOR song full;
```

To reset the creating text index parameters of a text index or use a different type of text index, it must be dropped and re-created. So the syntax of the fully rebuild text index is composed by 2 syntaxes, DROP and CREATE.

Example:

```
dmSQL> DROP TEXT INDEX ix_name FROM song;
dmSQL> CREATE IVF TEXT INDEX ix ON song(name);
```

4.2.2 EXAMPLE

Example1:

Environment: fully rebuild text index ix_info for table card:

```
dmSQL> REBUILD TEXT INDEX ix_info FOR card full;
```

After doing all the changes, rebuild text index is a fully rebuild.

Example2:

Environment: rebuilding the signature text index **ix_detail** for the table **card** and changing the text index's type to inverted-file index, we should use fully rebuild text index, executing the query as follow:

```
dmSQL> DROP TEXT INDEX ix_detail FROM card;
dmSQL> CREATE IVF TEXT INDEX ix_detail ON card(detail);
```

Example3:

Environment: changing the parameters of text index in table card by using fully rebuild text index. Rebuild the signature text index **ix_info** for the table **card** with a new **total text size** and **scale.**

```
dmSQL> DROP TEXT INDEX ix_info FROM card;
dmSQL> CREATE TEXT INDEX ix ON card(info)
       TOTAL TEXT SIZE 100 MB
       SCALE 60;
```

4.2.3 SPECIALTY

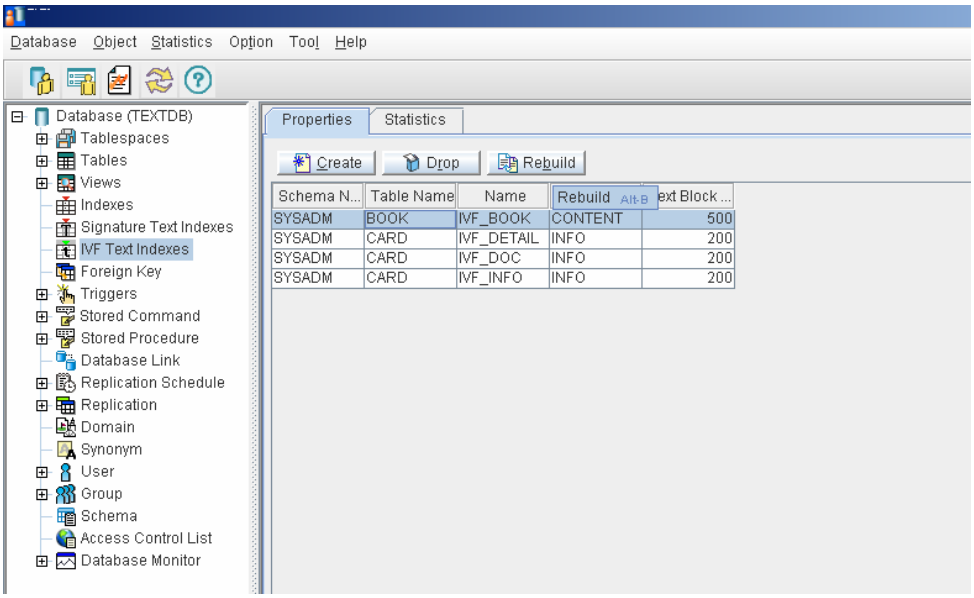
User will find that the space of text index took big and big through using incremental text index rebuild, however, rebuilding text index fully can solve this problem.

Fully rebuild text index can save more space text index occupied, if most records have been updated, we'd better use this way to rebuild text index.

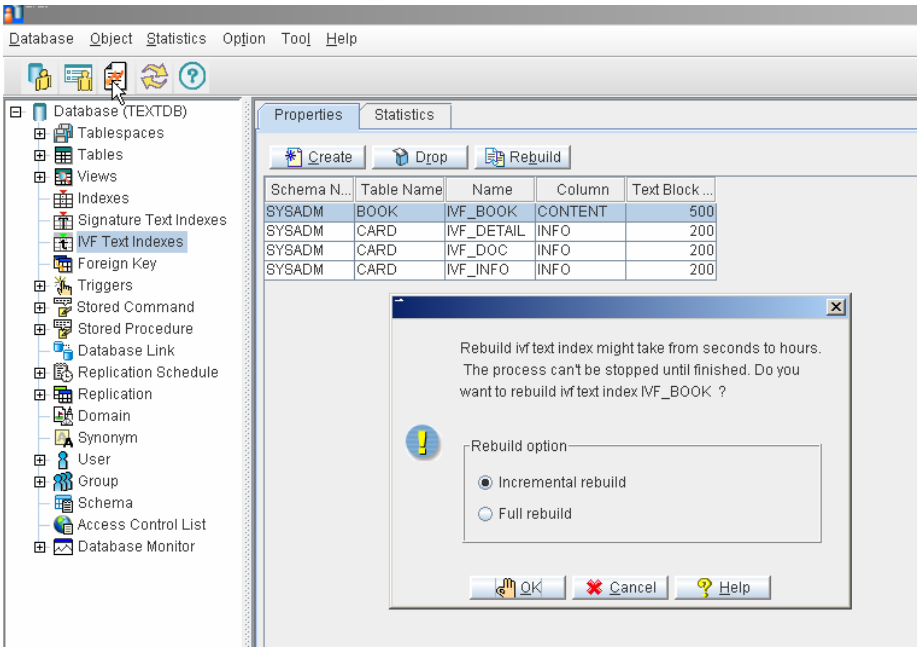
4.3 Rebuild text index by using JDBA Tool

Beside use dmSQL command rebuild text index, you also can use JDBA Tool rebuild text index.

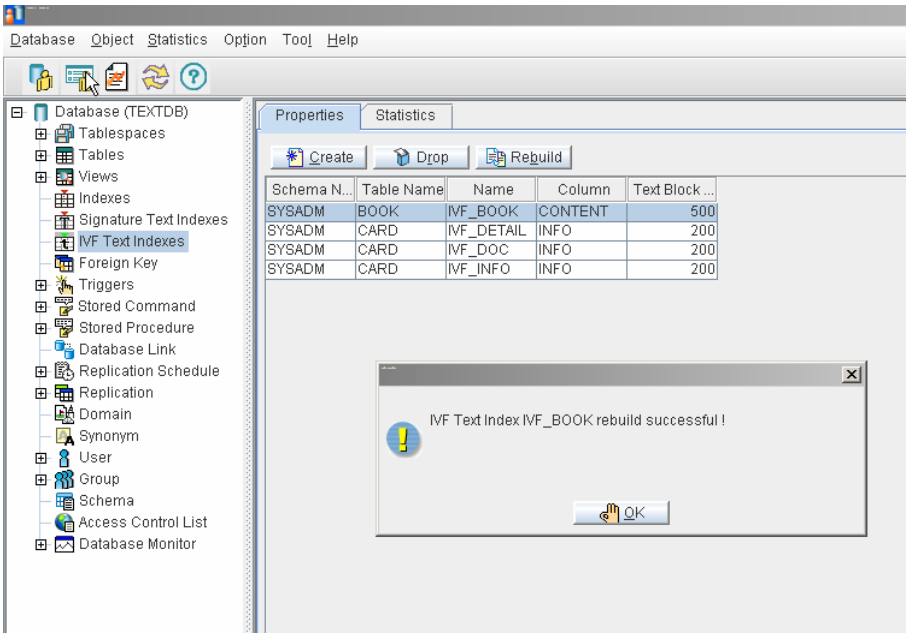
1. Open JDBA Tool; Choose signature or IVF text index under database directory, the select text index you want to rebuild from right table. Click rebuild button, rebuild text index by wizard. If you haven't chosen any text index, the rebuild button is disabled.



2. System will pop up an information window, choose the way you want to rebuild the text index, press OK button.



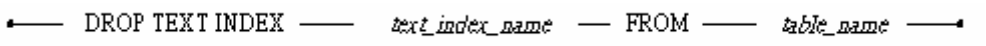
3. Then, there will be popped up a rebuild successful window. Your rebuilding have done.



5. Drop text index

The *DROP TEXT INDEX* command removes an existing text index in a table from the database. To execute the *DROP TEXT INDEX* command to drop a text index from a table, you must be the table owner, have **DBA** or **SYSADM** security privileges, or have the INDEX privilege for that table.

The DROP TEXT INDEX syntax is as following:



Picture 5-1: DROP TEXT INDEX syntax

Text indexes may be dropped using the JDBA Tool or the dmSQL *DROP TEXT INDEX* statement.

Example1:

Use dmSQL command drop text index.

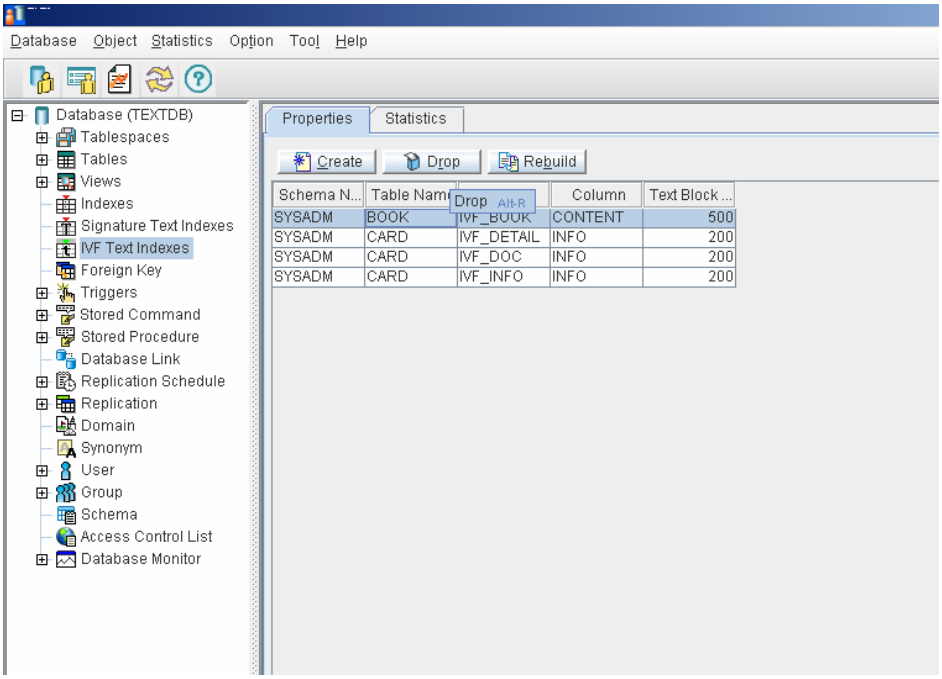
Environment: drop the **ix_S** text index from the table **book:**

```
dmSQL> DROP TEXT INDEX ix_S FROM book;
```

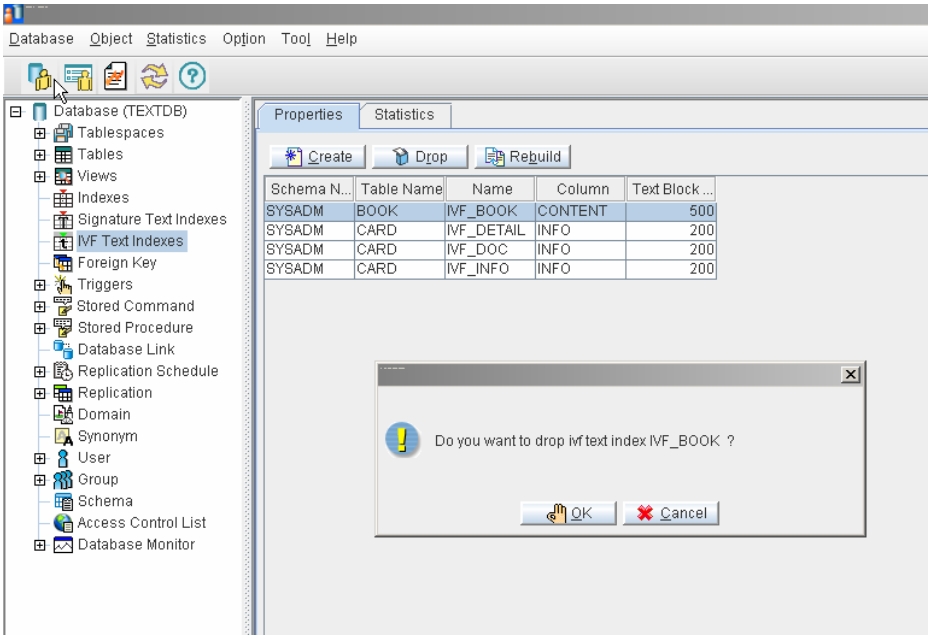
Example2:

Use JDBA Tool drop text index.

1. Choose signature or IVF text index under database directory, the select text index you want to drop from right table, then affirm the command you execute. If you haven't chosen any text index, the drop button is disabled.



2. Click drop button, affirm the command you execute.



One text index has dropped already.

6. Other search

In this chapter we will introduce the skills and search methods used in the Text Search, as well as operation used in example.

6.1 Fuzzy/Near Logic Search

Sometimes users would like to search imprecise patterns. If only exact phrases are allowed, the query 'William Clinton' will not find 'William Jeffery Clinton' and vice versa. A Boolean expression like 'William & Clinton' may return many irrelevant results. DBMaker provides fuzzy & near logic search features that allows users to perform imprecise queries without receiving too many irrelevant results.

6.1.1 FUZZY SEARCH

A phrase led by a '?' (Question mark) will be evaluated as a fuzzy expression, e.g. '?intel pentium'. Words used for a search in a fuzzy expression can be separated by up to four words in the target text. For example, '?intel pentium' will find 'Intel will release its 1GHz Pentium III processor', and '?amd k7 athlon' will find 'AMD has renamed its K7 processor as Athlon'.

Example:

```
dmSQL> select * from textfile;
    ID                                                  CON
========== ========================================================================
         1  DBMaker is a powerful Database
         2  DBMaker is a powerful and flexible SQL Database
         3  DBMaker is a powerful and flexible SQL Database Management System (DBMS)
         4  DBMaker is a powerful and flexible SQL Database that provides excellent
            text index mechanism
4 rows selected
dmSQL> SELECT id FROM textfile WHERE con MATCH '?A SQL  & ? text index';
    ID
==========
         4
1 rows selected

dmSQL> SELECT id FROM textfile WHERE con MATCH '(?dbmaker)  & (? text index)';
    ID
==========
         4
1 rows selected

dmSQL> SELECT id FROM textfile WHERE con MATCH '?(SQL )  & (text index)';
    ID
```

```
==========
ERROR (8345): [DBMaker] invalid match pattern

dmSQL> SELECT id FROM textfile WHERE con MATCH '?(database)  | (text index)';
    ID
==========
ERROR (8345): [DBMaker] invalid match pattern

dmSQL> SELECT id FROM textfile WHERE con MATCH '?database  | text index';
    ID
==========
         1
         2
         3
         4
4 rows selected

dmSQL> SELECT id FROM textfile WHERE con MATCH 'POWERFUL  | ? SYSTEM';

    ID
==========
         1
         2
         3
         4
4 rows selected
```

NOTE: The phrase in a fuzzy expression cannot contain any other operators. Thus the expression '?A SQL & ? text index' will be evaluated as '(?A SQL) & (text index)', and '?(SQL & text)' will cause an error.

A number of words in the query may be missing from the result set. For example '?William Jeffery Clinton' can find 'William Clinton' and 'William J Clinton', but the first word of the query must appear; the query '?William Clinton' will not find 'Bill Clinton'.

Example:

```
dmSQL> select * from textfile where con match '?a java database';
    ID                                                          CON
========== ======================================================================
         1  DBMaker is a powerful Database
         2 DBMaker is a powerful and flexible SQL Database
         3 DBMaker is a powerful flexible Database Management System (DBMS) that
         4 DBMaker is a flexible SQL Database that provides excellent embedded Text
         5 DBMaker supports a Java Database Connectivity compliant interface and
5 rows selected

dmSQL> select * from textfile where con match '? java database';
    ID                                                          CON
========== ======================================================================
         5 DBMaker supports a Java Database Connectivity compliant interface and
1 rows selected
```

Use matching '?a java database' ,we got 5 records; use matching '? java database', just got one. The result is words in fuzzy query may be missing and the first word must appear.

Fuzzy expressions can be combined with other text Boolean operations.

6.1.2 NEAR LOGIC FULL-TEXT SEARCH

A phrase led by a '~'(tilde mark) will be evaluated as a near expression. It ensures that all words in the query string appear in the text. For example, '~amd sales 1ghz athlon' will find 'AMD announced quarterly sales of its 1ghz Athlon chip', but not, 'AMD announced quarterly sales of its Athlon chip'.

Example:

```
dmSQL> SELECT * FROM textfile WHERE con MATCH '~interface dbmaster cobol';

    ID                                                          CON
===========
=========================================================================
          6 compliant interface and DBMaster COBOL interface (DCI
1 rows selected

dmSQL> SELECT * FROM textfile WHERE con MATCH '?interface dbmaster cobol';

    ID                                                          CON
===========
=========================================================================
          5 compliant interface and DBMaker COBOL interface (DCI).
          6 compliant interface and DBMaster COBOL interface (DCI
2 rows selected
```

Near logic full-text search is more precise than fuzzy search. We can use near logic search get the result we wanted.

Near match search expressions can be combined with other text Boolean operations.

6.1.3 FUZZY/NEAR LOGIC MATCHING RULES

The following four rules apply to the matching of a query string to the result string.

1. The first word of the query must appear, e.g. the query '?William Clinton' will not find 'Bill Clinton'.
2. Words can be separated by a preset number of words ("proximity"), e.g. '?intel pentium' will find 'Intel will release its 1GHz Pentium III processor', and '?amd k7 athlon' will find 'AMD has renamed its K7 processor as Athlon'. Currently the number of additional words in the matched result set between words in the query string can be no greater than 4.
3. A number of words in the query may be missing from the result set, e.g. '?William Jeffery Clinton' can find 'William Clinton' and 'William J Clinton'. The maximum allowed number of missing words is determined by the formula:

   max_miss = num_words – round (num_words * threshold).

   The current threshold is 0.75.
4. All words in the query must appear in the original order, e.g. '?amd 1ghz k7 athlon' will find 'AMD will announce 1GHz Athlon', but not 'AMD Athlon, formerly known as K7'.

6.1.4 COMPARE FUZZY SEARCH & NEAR LOGIC FULL-TEXT QUERY

The difference between Fuzzy search & Near logic search:

- A fuzzy match allows users to perform inexact queries without receiving many irrelevant returns. A near match search is similar to a fuzzy search, but more exact.
- For fuzzy search a number of words in the query may be missing from the result set, but near logic search ensures that all words in the query string appear in the text.
- Near search meets the rules 1, 2 and 4 above but does not allow for missed words; all rules mentioned in 6.1.3 suit for Fuzzy search.
- Near search is a special case of fuzzy search.

6.2 Boolean text search

Not only can the MATCH operator search a simple text pattern, but also complex Boolean operations.

6.2.1 BOOLEAN CHARACTERS' SEARCH PATTERN

A user can specify the following Boolean characters in a search pattern:

'&' - AND

'|' - OR

'-' - EXCLUDE

'(' - Left bracket

')' - Right bracket

6.2.2 PRECEDENCE OF BOOLEAN CHARACTERS

The precedence of Boolean characters is: BRACKET > EXCLUDE = AND > OR. When a MATCH pattern contains Boolean characters, all the other characters between Boolean characters are processed as simple search patterns.

For example, if the MATCH pattern is "coffee | tea | apple juice", then the search pattern includes "coffee", "tea" and "apple juice".

6.2.3 EXAMPLE

Example1:

To search for documents that contain the words 'love' and 'friend' from table **song**:

```
dmSQL> select * from song;

    ID              NAME
=========== ====================
          1 Endless Love
          2 Love Story
          3 My dear friend
          4 Hi! Be my love
4 rows selected
dmSQL> SELECT * FROM song WHERE name MATCH 'love & friend';
   ID              NAME
=========== ====================
0 rows selected
```

Example2:

The following searches the documents that contain 'love' or 'friend' and does not include 'endless love'.

```
dmSQL> SELECT * FROM song WHERE name MATCH '(love | friend) - endless love';

    ID              NAME
========== ====================
         2 Love Story
         3 My dear friend
         4 Hi! Be my love
3 rows selected
```

6.3 Full text search UDFs

Text index also can use in searching the results we wanted by using UDF. For example, we can use some functions in DBMaker to get special patterns. For more information about each UDF arguments and returned values, please refer to the *SQL Command and Function Reference User's Guide*,

6.3.1 BLOBLEN

Use BLOBLEN function to return the data length of an input BLOB in table blob1.

```
dmSQL> create table blob1(c1 long varchar, c2 long varbinary, c3 file, c0 serial(1));
dmSQL> insert into blob1 values();
dmSQL> insert into blob1 values(?,?,?);
dmSQL/Val> '','',&d:\release\workspace\empty.txt;
dmSQL/Val>&d:\release\workspace\100char_newline.txt,&d:\release\workspace\100char_newl
       ine.txt,&d:\release\workspace\100char_newline.txt;
dmSQL/Val>&d:\release\workspace\50chinese_nonewline.txt,&d:\release\workspace\50chines
       e_nonewline.txt,&d:\release\workspace\50chinese_nonewline.txt;
dmSQL/Val> end;
```

```
dmSQL> select c0, bloblen(c1) from blob1;
    C0      BLOBLEN(C1)
========== ===========
         1    NULL
         2           0
         3          19
         4         119

4 rows selected
```

6.3.2 HIGHLIGHT

Use HIGHLIGHT function to highlight the source text in table book which match the searching pattern.

```
dmSQL> select highlight(subject,'text index',0,'<<','>>') from book where content
       match 'dbmaker';

HIGHLIGHT(SUBJECT,'TEXT INDEX',0
================================
<<text index>>
```

```
Distributed Data


2 rows selected
```

6.3.3 HITCOUNT

Use HITCOUNT function to return the frequency match the searching pattern DBMaker in table thtable.

```
dmSQL> create table (thtable i serial, c1 long varchar);
dmSQL> insert into thtable values(?,?);
dmSQL/Val> 1,'DBMaker 超库网站包括dbmaker介绍、DBMAKER教学、 DBMaker 常见问答、DBMaker 快讯
等多项服务。';
dmSQL/Val> 2,'ABC abc Abc aBc DabcD dABCd 123Abc 123ABC abc123 ABC123';
dmSQL/Val> 3,'DBMaker provides you with a professional database system
        2> with multimedia capabilities';
dmSQL/Val> 4,'<p><font size="2" class="line120" face="Arial">DBMaker</font>
        2> <font size="2" class="line120">超库
        3> 网站的目的<font face="Arial">DBMaker</font> ';
dmSQL/Val> end;
```

```
dmSQL> select i,hitcount(c1,'DBMaker',0) from thtable where c1 match 'DBMaker';
     I      HITCOUNT(C1,'DBMAKER',0)
=========== ========================
          1                        5
          3                        1
          4                        2


3 rows selected
```

6.3.4 HITPOS

Use HITPOS function show the position information match the searching pattern DBMaker in table thtable.

```
dmSQL> select i,hitpos(c1,'DBMaker',0,1,1) from thtable where
hitpos(c1,'DBMaker',0,1,1) > 0;
     I      HITPOS(C1,'DBMAKER',0,1,1)
=========== ==========================
          1                          7
          3                          7
          4                         54


3 rows selected
```

6.3.5 HTMLHIGHLIGHT

Use HTMLHIGHLIGHT function modified content in which all text matching “DBMaker” will be highlighted with “” and “” before and after.

```
dmSQL> select i,htmlhighlight(c1,'DBMaker' ,1,'<font color="#800040">','</font>') from
thtable where c1 case match 'DBMaker';
    I      HTMLHIGHLIGHT(C1,'DBMAKER' ,1,'','')
=========== ====================================
          1 <BODY><font color="#800040">DBMaker</font> 超库网站包括dbmaker介绍、DBMAKER教
学、 <font color="#800040">DBMaker</font> 常见问答、<font color="#800040">DBMaker</font>
快讯等多项服务。</BODY>
```

```
         3 <BODy><font color="#800040">DBMaker</font> provides you with a
professional database system
with multimedia capabilities</BODy>
         4 <BODy><p><font size="2" class="line120" face="Arial"><font
color="#800040">DBMaker</font></font>
<font size="2" class="line120">超库
网站的目的<font face="Arial"><font color="#800040">DBMaker</font></font></BODy>

3 rows selected
```

6.3.6 HTMLTITLE

Use HTMLTITLE function to find the title (text between *html tags* "<*title*>" and "<*/title*>" in table htmltable) of HTML data.

```
dmSQL> create table htmltable (i serial, c1 long varchar);
dmSQL> insert into htmltable values(?,?);
dmSQL/Val> 1,'<HTML><HEAD><HEAD><TITLE>row1</TITLE></HEAD><BODY>BODY</BODY></HTML>';
dmSQL/Val> 2,'<title>row2 + space<title>';
dmSQL/Val> 3,'<title>row2 + enter
        2> this is title<title>';
dmSQL/Val> 4,'<Title>row3 <titLe>';
dmSQL/Val> end;
```

```
dmSQL> select htmltitle(c1) from htmltable;
                                HTMLTITLE(C1)
================================================================================
row1
row2 + space
row2 + enter
this is title
row3

4 rows selected
```

6.3.7 SUBBLOB

Use SUBBLOB function to get parts of source text match searching pattern from the defined position and length in table blob1.

```
dmSQL> select c0, bloblen(SubBlob(c1, 1, 10)), SubBlob(c1, 1, 10) from blob1;
    C0      BLOBLEN(SUBBLOB(C1, 1, 10))        SUBBLOB(C1, 1, 10)
=========== ========================== ================================
          1                       NULL                            NULL
          2                       NULL                            NULL
          3                         10 313030636861725f6e65
          4                         10 4647b5d8b7bdb5dac8fd

4 rows selected
```

7. Search on Unicode column

Text indexes can be built on all character type columns, beside nchar, nclob and nvarchar data type. This chapter introduces how to handle the data in those columns, create text index on them and search methods.

7.1 Unicode data type

In DBMaker, Nchar, Nclob and Nvarchar data type can contain Unicode characters. Each Unicode character occupies two bytes of storage in UTF16 Little-Endian (LE) encoding.

- **Nchar**

The NCHAR data type is a fixed-length data type. The (size) parameter determines the number of 2 byte characters in the column. The (size) parameter must be entered when creating an NCHAR column, and may range from a minimum of 1 to a maximum of 1996.

Synonyms for the NCHAR data type include NATIONAL CHAR(size), and NATIONAL CHARACTER(size).

- **Nclob**

The NCLOB data type is a variable length data type. The maximum length for an NCLOB column is 1 gigabyte (GB).

Synonyms for the NCLOB data type include NATIONAL CHAR LARGE OBJECT, NCHAR LARGE OBJECT, and NATIONAL LONG VARCHAR.

- **Nvarchar**

The NVARCHAR data type is a variable-length data type. The (size) parameter determines the number of 2 byte characters in the column. The (size) parameter must be entered when creating an NVARCHAR column, and may range from a minimum of 1 to a maximum of 1996.

Synonyms for the NVARCHAR data type include NATIONAL CHAR VARYING(size), NCHAR VARYING(size), NATIONAL VARCHAR(size), and NATIONAL CHARACTER VARYING(size).

7.2 Operation on Unicode column

7.2.1 INSERT

When entering NCHAR, NCLOB or NVARCHAR data, enclose the Unicode character with single quotes (‘’) and prefix the quotes with 'N'.

Example:

Create table uni (The details of table uni refer to Appendix), then insert data into it.

```
dmSQL> insert into uni values('aaa','bbb','ccc','ddd');
1 rows inserted
```

```
dmSQL> insert into uni values(N'apple',N'berry',N'coco',N'ddd');
1 rows inserted
```

When a character string is input to a Unicode column but is not prefixed by 'N', then it will automatically be converted from local code to Unicode.

```
dmSQL> insert into uni values('The physical','across the files that','make up a
database can become quite complex.','overview');
WARNING (63): [DBMaker] data truncated when converting from different type
1 rows inserted


dmSQL> insert into uni values('Welcome to','Database Administrator','flexible SQL
Database Management System (DBMS)','dbmaker');
WARNING (63): [DBMaker] data truncated when converting from different type
1 rows inserted


dmSQL> insert into uni values('@','#','$','&');
1 rows inserted
```

If data is input in hexadecimal format, enclose the hexadecimal string with quotes and append a 'u' character. As following:

```
dmSQL> insert into uni values('6eee'u,'2394'u,'adfb'u,'cccc'u);
1 rows inserted
```

The Unicode column don't support insert file data type. When insert a file into Unicode column, we can't find the context in the file, the system will recognize it as char type data.

Example:

```
dmSQL> insert into uni values('f:\abc.doc','f:\abc.doc','f:\abc.doc','f:\abc.doc');
1 rows inserted


dmSQL> select * from uni where c1 match 'twins';
                    C1
=======================================
0 rows selected

dmSQL> select * from uni where c1 contains 'abc';
                    C1
=======================================
f:\abc.doc
1 rows selected
```

7.2.2 DISPLAY MODE

When executing a query, we can find the data input into table have been converted hexadecimal format.

Example:

```
dmSQL> select c1 from uni;
                    C1
=======================================
5400680065002000700068007900730069006300
570065006c0063006f006d006500200074006f00
6100610061002000200020002000200020002000
6100700070006c0065002000200020002000
6eee200020002000200020002000200020002000
```

```
400020002000200020002000200020002000
6 rows selected
```

NOTE: If NCHAR data is entered into a column that is shorter than the column length, the data will be padded with spaces. The space in hexadecimal format is '2000'u.

Execute the following command can display the original data you put into.

```
dmSQL> select cast(c1 as char(20)) from uni;
CAST(C1 AS CHAR(20))
====================
The physic
Welcome to
aaa
apple
n
@
6 rows selected
```

7.2.3 SEARCH, UPDATE, DELETE

Use text search, update or delete.

Example:

```
//search
dmSQL> select c1 from uni where c1 match'the';
                  C1
========================================
54006800650020007000680079007300690063 00
1 rows selected

dmSQL> select c1 from uni where c1='aaa';
                  C1
========================================
61006100610020002000200020002000200020002000
1 rows selected

//update
dmSQL> update uni set c1='@@@' where c3='$';
1 rows updated
dmSQL> update uni set c1='new record' where c3 contains 'a';
3 rows updated
dmSQL> update uni set c1='welcome!!!' where c3 match N'sql';
1 rows updated

//delete
dmSQL>  delete from uni where c2 ='#';
ERROR (6130): [DBMaker] invalid operation in expression
dmSQL> delete from uni where c2 match '#';
2 rows deleted
dmSQL> delete from uni where c1 contains 'new';
2 rows deleted
```

NOTE: we can use equal mark(=) search, update or delete on nchar and nvarchar column. But there is an error will pop up when you do same operations on nclob column with '='.

Use hexadecimal code search.

Example:

```
//search
dmSQL> select c1 from uni where c1 match '6eee'u;
                  C1
=======================================
6eee2000200020002000200020002000200020002000
1 rows selected

dmSQL> select c1 from uni where c1 match '6100'u;
                  C1
=======================================
0 rows selected

dmSQL> select c1 from uni where c1 contains '6100'u;
                  C1
=======================================
6100610061002000200020002000200020002000
6100700070006c0065002000200020002000200020002000
2 rows selected

//update
dmSQL> update uni set c1='8745'u where c2 contains '6600'u;
2 rows updated
dmSQL> update uni set c2='a database' where
       c1='770065006c0063006f006d006500210021002100'u;
1 rows updated
dmSQL> update uni set c2='butter' where c1 match '66007200750069007400'u;
1 rows updated

//delete
dmSQL> delete from uni where c2 contains '7500'u;
1 rows deleted
```

When searching part text in record, if matching condition is the whole record, we can use the equal mark(=); if it's the words, can use 'match' and 'contains'; if the matching condition is the characters, we just can use 'contain' to query.

7.3 Relationship between local code and Unicode

UTF-32、UTF-16 和 UTF-8 are some encoding character mode in standard Unicode encoding character set. UTF8, UTF16BE and UTF16LE are used widely, UTF means the Unicode Transformation Format, so they just one using mode of Unicode. DBMaker support UTF16LE being encoding format, so we must convert local code to DBMaker code if inputting others Unicode into DBMaker's Unicode column.

- **UTF8:** use 8 bits (one byte) as encoding unit. It's the unfixed length Unicode. U+0000 to U+007F use one byte to encode, U+0080 to U+07FF use two byte to encode, U+0800 to U+FFFF use three byte to encode and U+10000 to U+10FFFF use two byte to encode. The principle of UTF-8 is: the value 0x00 to 0x7F always show U+0000 to U+007F（Basic Latin ）.

- **UTF16BE:** serializes a Unicode value as a sequence of two bytes, in big-endian format. e.g. FE FF (byte-order mark) 00 41 00 42 00 43.
- **UTF16LE:** serializes a Unicode value as a sequence of two bytes, in little-endian format. e.g. FF FE (byte-order mark) 41 00 42 00 43 00.
- **UTF32:** use 32 bits as encoding unit.

UTF8 and UTF16 can easily convert to encoding format, the method is between [0 ,0x007f] convert it to the first byte, then between [0x0080 , 0x07FF] convert it to second byte, the result is 110 xxxxx 10 xxxxxx , that's UTF-8; between [0x0800 , 0xffff] convert it to third byte, the result is 1110 xxxx 10xx xxxx 10xx xxxx , that's UTF-16.

The following byte stream shows a character. The used byte stream decided by serial number in Unicode.

U-00000000 - U-0000007F: 0xxxxxxx

U-00000080 - U-000007FF: 110xxxxx 10xxxxxx

U-00000800 - U-0000FFFF: 1110xxxx 10xxxxxx 10xxxxxx

U-00010000 - U-001FFFFF: 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

U-00200000 - U-03FFFFFF: 111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

U-04000000 - U-7FFFFFFF: 1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

Example:

Encode Unicode character U+00A9 = 1010 1001 as UTF-8:

11000010 10101001 = 0xC2 0xA9

Different encoding format of character:

Unicode encoding	U+0041	U+00DF	U+6771	U+10400
character	A	ß	東	𐐀
UTF-32	00000041	000000DF	00006771	00010400
UTF-16	0041	00DF	6771	D801 DC00
UTF-8	41	C3 9F	E6 9D B1	F0 90 90 80

So it's convenient to convert UTF-8 to UTF-16 when we knowing the rule. We don't need code table, but just converting arithmetic and some functions such as **iconv** which are easy to find.

8. Performance

Performance for text indexes is influenced by many factors, such as memory, disk speed, and CPU speed. Expect hardware resources, performance for text indexes also depends on the following:

- Occasion of using signature and IVF text indexes

  Try both types of text index to find which one suits the data's characteristics best. When comparing signature and IVF text indexes, you not only consider time of creating/updating text indexes and respond time of queries, but also storage space used by text index. Generally speaking, signature indexes will take longer to respond, especially when dealing with larger quantities of data.

- Select appropriate method to update text indexes, and updating text indexes should be done when little activity are processing in current system. After creating text index or rebuilding text index, if you updated data, you must update text indexes to reflect new update to text index.

  But the operation will occupy system resources and may take much time.

- Adjust Signature text index parameters
- Adjust IVF text index parameters
- Prepare enough independent disk space for text indexes directory, so let location of database files and location of text indexes directory be in different disk space.
- adjust frame size can improve performance and save disk space

Example:

In the following example, we will compare performance of using signature and IVF text indexes.

Test ENV:

create table article(num serial, title varchar(100),content file);

The size of all files inserted is about 500MB.

A. Signature

```
dmSQL> create text index ix_content on article(content);
Estimated elapsed time = 32.859000 seconds
dmSQL>  select count(*) from article where content match 'JDBC';
COUNT(*)
===========
        500

1 rows selected
Estimated elapsed time = 6.031000 seconds
```

B. IVF

```
dmSQL> create ivf text index ix_content on article(content);
Estimated elapsed time = 78.563000 seconds
dmSQL>  select count(*) from article where content match 'JDBC';
COUNT(*)
===========
        500

1 rows selected

Estimated elapsed time = 0.156000 seconds
```

Signature text index is stored in Database as BLOB data. For the above example, it took 5535 Frames (each frame = 16K).For IVF text index, it took about 97.4MB disk space. For larger quantities of data, IVF has better performance than Signature.

Adjusting frame size, for smaller frame size, IVF took less disk space and got the better performance.

Appendix Tables used in the samples

The following table shows all the tables that used in the previous samples, including the detail information of tables. It can help user to understand the sample easily.

Table name	Table definition
BOOK	CREATE TABLE SYSADM.BOOK (AUTHOR CHAR(10) DEFAULT NULL , SUBJECT CHAR(80) DEFAULT NULL , INTRO LONG VARCHAR DEFAULT NULL , CONTENT FILE) ;
TEXT	CREATE TABLE SYSADM.TEXT (ID INTEGER DEFAULT NULL , CON FILE);
TEXT1	TEXT1'S STRUCTURE IS THE SAME AS TEXT'S.
TEXT2	TEXT2'S STRUCTURE IS THE SAME AS TEXT'S.
TEXT3	TEXT3'S STRUCTURE IS THE SAME AS TEXT'S.
TEXT4	TEXT4'S STRUCTURE IS THE SAME AS TEXT'S.
CARD	CREATE TABLE SYSADM.CARD (NUM INTEGER DEFAULT NULL , INFO LONG VARCHAR DEFAULT NULL);
DOCUMENT	CREATE TABLE SYSADM.DOCUMENT (ID INTEGER DEFAULT NULL , CONTENT CHAR(100) DEFAULT NULL) ;
SONG	CREATE TABLE SYSADM.SONG (ID INTEGER DEFAULT NULL , NAME VARCHAR(20) DEFAULT NULL) ;
SAVEFILE	CREATE TABLE SYSADM. SAVEFILE (ID INTEGER DEFAULT NULL ,

	SDOC MSWORDFILETYPE DEFAULT NULL);
TEXTFILE	CREATE TABLE SYSADM.TEXTFILE (ID INTEGER DEFAULT NULL , CON CHAR(100) DEFAULT NULL) ;
UNI	CREATE TABLE SYSADM.UNI (C1 NCHAR(10) DEFAULT NULL , C2 NCLOB(20) DEFAULT NULL , C3 NVARCHAR(30) DEFAULT NULL);
FRUIT	CREATE TABLE SYSADM.FRUIT (NUM SERIAL, SORT CHAR(20) DEFAULT NULL, OTHERS CHAR(20) DEFAULT NULL) ;

Appendix Table: Tables used in the samples