User Manual

ODYSSEUS/OOSQL

Version 5.0

Manual Release 1

Aug. 2016

Advanced Information Technology Research Center (AITrc)

KAIST

Contents

1. Directory Structure of ODYSSEUS/OOSQL

Directory/File	Details
OOSQL	Object file and header files for Linux
document	Document
source	Source code
example	Example file

The source code is written in C and C++, and contains approximately 355,000 lines.

2. Compiling ODYSSEUS/OOSQL

The environment for source code compilation on the Linux OS is as follows:

- 32bit
  - Platform: Linux 2.6
  - Compiler: gcc 4.1.2 Compiler
- 64bit
  - Platform: Linux 2.6
  - Compiler: gcc 4.4.7 Compiler

Compilation is performed in the ODYSSEUS/OOSQL source code directory using the procedure outlined below.

1) Install the ODYSSEUS/COSMOS binary and header files in a suitable directory. There are three versions of ODYSSEUS/COSMOS (32bit, 64bit supporting large databases, 64bit not supporting large databases). Refer the ODYSSEUS/COSMOS User Manual for details.
   (ODYSSEUS/COSMOS binary and header files for Linux are needed to compile ODYSSEUS/OOSQL.)
2) Extract the compressed ODYSSEUS/OOSQL source code file to a suitable directory. After extraction is complete, check to make sure that the three subdirectories (./LOM, ./GEOM, ./OOSQL), the setup file, and the Make.sh file have been created.
3) Modify the setup file, which stores the various settings for compiling the ODYSSEUS/OOSQL source code. The types and definitions of the environment variables configured in the setup file are listed in the table below. Since LOM, GEOM, and OOSQL compile the source code file and create the binary file in each subdirectory, the location of the source code is identical to the location of the

binary. Therefore, it is recommended that the O_ROOT and O_KAOSS environment variables be configured with care, and the default settings be used for the remaining variables. O_OOSQL_EXPORT designates the copy path of the binary file after ODYSSEUS/OOSQL has been compiled. SWIG, BISON, and PYTHON designate the locations of the respective applications. (SWIG 1.3.29, Bison 1.28, and Python 2.4 binary and header files for Linux are needed to compile ODYSSEUS/OOSQL.)

Environment Value	Default Value	Definition
O_ROOT	Configured manually	Source code root directory
O_SERVER	$O_ROOT	ODYSSEUS/OOSQL source code root directory
O_KAOSS	$O_ROOT/COSMOS	Location of ODYSSEUS/COSMOS bianry
O_LOM_COMMON	$O_ROOT/LOM	Location of LOM source code
O_LOM_SERVER	$O_ROOT/LOM	Location of LOM binary
O_GEOM	$O_ROOT/GEOM	Location of GEOM source code/binary
O_OOSQL_COMMON	$O_ROOT/OOSQL	Location of OOSQL source code
O_OOSQL_SERVER	$O_ROOT/OOSQL	Location of OOSQL binary
O_COMMON	$O_ROOT/OOSQL	Location of OOSQL utilities
O_DLLBROKER	$O_LOM_SERVER/RPCdll/dllbroker	Location of LOM communication module(dllbroker)
O_DLLSERVER	$O_LOM_SERVER/RPCdll/dllserver	Location of LOM communication module(dllserveer)
O_OOSQL_EXPORT	Configured manually	Location where binary and header will be copied once OOSQL compilation is complete
SWIG	Configured manually	Location of SWIG binary
BISON	Configured manually	Location of Bison binary
PYTHON_HEADER	Configured manually	Location of Python header files

4) Open Makefiles in subdirectories of LOM and OOSQL, and modify SERVERFLAG depending on the version of ODYSSEUS/COSMOS (32bit, 64bit supporting large databases, 64bit not supporting large databases) to use. Disable the flag, DSUPPORT_LARGE_DATABASE2, for ODYSSEUS/COSMOS 32bit and 64bit not supporting large databases, and enable the flag for ODYSSEUS/COSMOS 64bit supporting large databases.

5) Execute the script file, Make,sh, to compile ODYSSEUS/OOSQL. Make.sh compiles LOM, GEOM, and OOSQL, sequentially. After compilation is complete, check to make sure that the ODYSSEUS/OOSQL object file (./OOSQL/liboosql.so) has been created in the OOSQL subdirectory. If it has not, compilation has not been carried out properly. Check for compile errors, resolve the issues, and perform the compilation again.

3. DATABASE Schema Creation

Especially, you should seek an expert advice when creating database schema because the schema is an important element in the performance of the search system.

Objectives

Write a schema to describe the data to search in a database.

Contents

Using SQL, you write a database schema suitable for the designed user interface.

Core Contents

1) When designing the database schema for a search system, you should select proper field types such as numeric, string, and text type. Among these, the text field always creates an index, however, the numeric field and the string field can optionally create the index. You need to pay attention to create the proper indexes.

2) Indexes for numeric and string field have to be created at the field with a low selectivity. A low selectivity means that there are few data of the same value like ID number, registration number, or name. A high selectivity refers to the cases such as publishing country, language, publishing year, and so on. If an index is created at the field with a high selectivity, it takes long time in searching due to lookup the index itself.
   (**Selectivity** = the number of documents as the search result / the total number of documents in the database)

3) For creating an index for the field with a high selectivity, you should use a text index. For instance, when you implement a library information search system, the year of publication can be used to search despite of its high selectivity. If a string index is used for the field, the search speed is reduced radically due to a high selectivity. However, using a text index, the search can be executed at a rapid pace. Even in case of constructing a text index for the numeric field, a range search for the field is possible by using '**BETWEEN** in **MATCH'** in **SELECT** statements. For example, use the following query to search for the book whose year of publication is between 2000 and 2010.

SELECT * FROM BOOK WHERE MATCH(YEAR, BETWEEN("2000", "2010"))>0

4) When you write a database schema, it is important to decide the type and the index for the column. The reason is that those can have an influence upon the performance of the whole search system. To decide the type and index for the column is also important when you move the schema, which has been used for the existing database management system or IR search system, into ODYSSEUS.

When you write the schema for a library search system, you have to decide a text attribute with a text index or a VARCHAR or CHAR attribute with B-Tree index for the fields such as ID number, the year of publication and so on.

When deciding on those matters, you can use rules of creating the database schema as follows:

Rule 1) Create a B-Tree index for the column that has a low selectivity for the Equality(=) operator. (eg. : Create a B-Tree index for the column like ID number that shows a low selectivity for the Equality(=) operator as the same index used in the commercial database management systems.)

Rule 2) Create a text index for the column that has a medium selectivity for the Equality(=) operator. (eg. : Create a text index for the column such as the publishing year that shows a medium selectivity for the Equality(=) operator.)

Rule 3) Separate database for the column that has a high selectivity for the Equality(=) operator. (eg. : For the columns like field of majors(physics, chemistry, etc.), or type of books(journal, separated volume, etc.), store the data after separating the database according to the value of the field.)

Rule 4) Even though falling under Rule 1 and Rule 3, you should create a text index for the column requiring partial match queries (the search for the document including the input keyword). When the partial match queries are required, create the text index for the field irrespective of the selectivity. (eg. : When you find a book written by many authors by typing only one author name, the author field is defined a text type with a text index.)

5) It is desirable to make the schema used in a search system be only one table. If two more tables are in the schema, there will be needed join between tables in searching. Because the join operation spends much time, you should design the schema in a way all the searches will be executed in one table as possible.

6) You cannot make the schema used for the search system be a single table in all cases. Though it is good to make a single table for 1:1 mapping, you had better refrain from searching documents without the join for 1:n and m:n mapping. For example, a book can have several names such as the full name, auxiliary name, and so on. If you create the schema to be one table, the field of table is to increase. That is, in case of 1:n mapping, you have to add n fields to the schema, and need n times OR operations in searching. For this case, you should divide the table into two, and it is needed the join to search tuples. Writing a schema to be in two tables does not always means the degradation of performance. If the number of join results is small in spite of having two tables, it is rather effective to process the join than n times OR operations.

Referential Contents

1) After creating schema, you should assign the volume where each table will be stored. At this time, you have to decide whether several tables will be stored in one volume or each single table will be stored in one volume. Both methods have their advantages and disadvantages as follows:

Method to store several tables in only one volume

- Advantages: Each table share one volume, so the disk space can be efficiently used, no matter how the data increases.
- Disadvantages: When a new data is inserted in the already organized database, the clustering may be not maintained, for the location where new data are stored can be away from the location where the previous data are stored. Also, when a table in the volume is update, every table in the same volume is locked unnecessarily because the whole volume is locked.

Method to store each table in a single volume

- Advantages: Each table is stored in a single volume, so the clustering can be maintained better than when all the tables are stored in a single volume. It makes the search faster.
- Disadvantages: The size of table is limited by the size of each volume, so it does not utilize the disk space efficiently compared to the method of storing all tables in a single volume. Also, join operation between relations in the different volume is not supported.

Therefore, you should consider disk space efficiency, clustering, locking, and relevancy between tables, before you assign the volume where each table will be stored.

4. ODYSSEUS/OOSQL Installation

Many problems have so far risen due to the wrong installation. You can reduce trials and errors by following the below instructions. It is desirable that an expert versed in Linux attends the installation.

Objectives

You install ODYSSEUS/OOSQL, the engine of ODYSSEUS database management system.

Contents

You install ODYSSEUS library, which is the engine of database management system, and copy the related utilities in the assigned directory. To install the ODYSSEUS library, refer to the compilation method explained in Section 2 or use the binary files provided. The installed files are OOSQL library, Header file for programming, relevant utilities, keyword extractor to extract the keyword from documents, and sample programs.

Core Contents

1) On installing ODYSSEUS, you can see library, include file, keyword extractor, utility, and sample program. After the installation, you have to adjust authorization to enable a user to access a directory or a file.

2) You can find the setup file created in the process of ODYSSEUS installation. The file contains the environmental variables necessary for operating ODYSSEUS. You should modify these environmental variables to be suitable for your system. For running ODYSSEUS successfully, you are to correctly declare theses environmental variables, and modify them to be suitable for the application program. The environmental variables are as follows:

ODYS_TEMP_PATH: Directory for the temporary file used in the process of executing ODYSSEUS

ODYS_OODB: Directory appointing the location of ODYSSEUS database

IR_SYSTEM_PATH: Directory containing keyword extractor

COSMOS_LOG_VOLUME: Location of log volume for preventing the damage to database, which is caused by a wrong operation.

COSMOS_COHERENCY_VOLUME: Location of coherency volume for multi-server environment

After installing ODYSSEUS, you need to adjust these environmental variables.

Additionally, you should write an ODYSSEUS library directory path in LD_LIBRARY_PATH so as to execute ODYSSEUS library.

3) Modifying the environmental variables in the setup file dose not complete fixing all the environments for the system operation. For the Linux environment, you have to reflect the contents of setup file in the system by executing “source setup” on the shell. The setup file is based on the tcsh shell of the OS. Hence, if tcsh is the shell currently being used, no changes need to be made. If not, the current shell must be changed to tcsh. In order to prevent the trouble of carrying out this operation every time a user login, it is also recommendable to add to the .login or .cshrc file making the operation automatically carried out.

   If this operation is not carried out automatically, you should run “source setup” before running utility programs or script, or declare the variable at the beginning part of the script to operate the system successfully. Additionally, you should write each program with script, and declare the necessary environmental variables to apply a different environmental variable for a program.

   The operation using scripts can prevent the typing errors, and help to execute all the operations related to database of ODYSSEUS in correct. Therefore, you have to execute the operations using scripts.

4) After installing ODYSSEUS, you have to format the log volume that will be used during running ODYSSEUS. You can use OOSQL_FormatLogVolume command. As a parameter of the command, type an appropriate file path and the size of a log volume to the name of device that will store the log volume. You should specify the size of log volume more than the maximum size of data updated in a transaction.

After creating a log volume, you register the device name of the created log volume on COSMOS_LOG_VOLUME, an environmental variable enabling other ODYSSEUS programs to use the log volume. Unless you create the log volume or specify the log volume location in environmental variables, errors occur in running the programs.

5) The coherency volume must be used where there are updates, insertion or deletion of data in a multi-server configuration. OOSQL_FormatCoherencyVolume is a utility to initialize a coherency volume for a multi-server configuration. For a multi-server configuration, if a buffer is updated in a process, the change is recorded in the coherency volume, and other processes make the contents of the buffer consistent by looking up the changes recorded in the coherency volume. For an OOSQL application program to use the coherency volume, the environment variable $COSMOS_COHERENCY_VOLUME must be set the path where the coherency volume resides.

6) After declaring an environmental variable, you need to check if it is operated in correct. First, type “setenv” on the shell to see and check the declared environmental variables. Secondly, check if the libraries are correctly linked. In case the directory defined in LD_LIBRARY_PATH environmental variable is wrong, an unexpected directory may be linked to a program. Therefore, you should run “ldd[1] <name of .exe file>” to check if the necessary libraries are correctly linked. This kind of error can especially occur when you run programs using the script, so you need to make sure that the environmental variables have been correctly declared in running the program.

Referential Contents

1) If necessary, you can install the library required for running ODYSSEUS not in a single directory but in several directories. For example, you can install liboosql.so in both /usr/lib and ~/OOSQL/lib. Installation in multiple locations like this is followed by trouble that you have to modify all the libraries when updating them. So you need to install the libraries relevant to ODYSSEUS, and execute files in one location, if possible. In case of an inevitable multi-installation, you have to make sure that the libraries and execute files are successfully linked using ldd and which[2].

[1] Provided in the Linux environment, this utility displays the linked directory of dynamic library used by execute file.
[2] Provided in the Linux environment, this utility informs where the execute file is located.

5. DATABASE Volume Construction

Inserting the data to the database be supplied by a search system, this process is divided into database creation, schema creation, data loading, index creation, and database test. Although being automatically executed, this operation demands for much time. You should keep the below instructions in mind to prevent mistakes. Before processing the operation, you need to carefully read 'ODYSSEUS/OOSQL Reference Manual' to know how to apply the utilities for database volume construction.

5.1. Database Creation

Objectives

Allocate the space for database to store data.

Contents

Allocate the space for database with utilities supplied by ODYSSEUS.

Core Contents

1) Apply OOSQL_CreateDB utility supplied by ODYSSEUS to create database. The parameter of the utility is supposed to give the device name and the number of pages to be allocated. When you assign a non-existed device or the one without permission, an error will occur. So you should assign a suitable device.

2) The correctness and performance of database operations depend on the types of devices composing a volume. It is better to use Raw devices. If raw devices were not used, due to unnecessary buffering in the O/S files, main memory is wasted, and consequently, the system can slow down. In particular, the main memory that is used for O/S file buffering can grow as large as the size of the device (say, a few hundred Mbytes ~ a few Gbytes), thrashing can occur due to shortage of remaining main memory.

3) Disk pages are allocated for the database. When the allocated pages are too small, an error occur during inserting data because the data size exceeds the number of allocated pages. So you should allocate the data pages according to the size of the input. The size of allocated pages should be 10 times (6 times when offset is not stored) of the size of the input data. This is for securing a sufficient space for input data and its index.

4) Contrary to the above case, when you allocate the raw device pages exceeding the total number of pages in the raw device to create a volume, the “invalid file format” error occur. Then you should check the number_of_page that has been declared as a parameter of OOSQL_CreateDB. Generally, the size of a page is 4Kbyte, so you can calculate the size of the volume with multiplying 4KByte by number of pages. If the size of the allocated pages is larger than the size of raw device, the error can occur. The size of raw device is determined at disk partition time. The maximum size of raw device is 2GB in the 32bit version and 8EB in the 64bit version.

5) When you want to create database again, you should run OOSQL_CreateDB after running OOSQL_DestroyDB.

Referential Contents

1) Run OOSQL_InitDB to delete all the data and tables in the created database. In order to delete the database already created and insert new data, you should run OOSQL_InitDB instead of OOSQL_CreateDB that was used for creating database in the beginning. And you insert new data.

2) When composing a volume, you can add 4,000 raw devices of 2GB at the maximum. So, considering the extendibility, you need to use 2GB raw device from the beginning. In this way, you can cover up to 8TB data. However, there is no necessity for adding all the available raw devices from the beginning in consideration of the extendibility. When the storage space is insufficient, you can add raw device to a volume using OOSQL_AddDevice, a utility of OOSQL. Refer to OOSQL Reference Manual for the detailed information.

5.2. Schema Creation

Objectives

Create the designed schema in database.

Contents

Change the designed schema into SQL to create a schema in database. Register the keyword extractor in database for the text search.

Core Contents

1) The schema creation that the process define the tables to be used in a search system has three methods: i) to create a schema using isql interactively, ii) to write a schema in the file, and create it using isql or ODYSSEUS/Web-PHP utilities, iii) to write C programs using OOSQL API. Method ii) is recommendable for many implementations, Method iii) can be applied when the schema is perfectly fixed. Method i) is not applied except creating a temporary schema for tests.

2) You should pay attention to the data types of char(10) and varchar(10) when defining a schema. For only 2 byte string, the char(10) data type is filled the latter 8 bytes with null characters keeping up the size of a volume to be 10. However, the varchar(10) data type is filled the last 1 byte with a null character.

3) Register the information on a keyword extractor to find out keywords in a text field. In order to insert the text data in ODYSSEUS, you should register a default keyword extractor to be used for all text fields. However, another type of the keyword extractor may be at need, so you can additionally register another keyword extractor for a specific attribute in a table.

   For registering a keyword extractor for a specific attribute, you have to create the keyword extractor first of all, and write interface to register the keyword extractor in the database.

   Refer to OOSQL/example/null_keyword_extractor program for how to create a keyword extractor. Link the created keyword extractor to the database using OOSQL utilities. InstallDefaultKeywordExtractor, InstallKeywordExtractor, and SetKeywordExtractor are supplied as utilities.

4) The keyword extractor that separates a word without any morpheme analysis is provided. The source code is in OOSQL/example/null_keyword_extractoras. The binary code is in OOSQL/bin/NullKeywordExtractor.so after compilation.

5) A keyword extractor is programmed to change all English letters into small letters before extracting the keyword. Then, the query processor can search the documents containing the given keyword regardless of a capital or small letter. This is why you do not need to care about whether the input data is a capital letter or a small letter in case of your constructing a search system.

   Though, when you have to create a new keyword extractor, you need to consider this problem. When you create a new keyword extractor, you have to add the process of changing all the extracted keywords into small letters; otherwise you cannot find documents to want.

For example, when a keyword in a document is the form of the capital letter, you cannot search the document by querying with a small letter keyword.

5.3. Data Loading

Objectives

Load the data into the database for the search system.

Contents

Load the data into the database storing the designed schema for the search system. The schema has to be created in the database.

Core Contents

1) In order to load data, you should use OOSQL_LoadDB utility in OOSQL.

2) A data file loaded using OOSQL_LoadDB utility is specified the table and attributes names on the top of the file, and is described actual data for each attribute. You should remember that '_ and "_ are used to indicate the char or varchar and the text fields. So you should replace these with \', and \", if the data with '_ and "_ is inserted. For example, when you insert a document having 'The Weather of Korea and 'World'', 'The Weather of Korea and ' is recognized as one document and the rest makes an error. To fix the error, you have to modify the string into 'The Weather of Korea and \'World\''. And '\n' means the carriage return.

3) Another problem similar to the above one is that you may encounter \ \ ' in the document. At this time, \ is processed to be \ \ producing the same results as inserting only ' because the next letter to \ is construed as a special character. So it is regarded that there is ' in the middle of document, and it makes an error. You have to change the string \ \ " and \ \ ' into \ \ \ " and \ \ \ '.

4) You have to write a definition of the class in the loading file, which contains the documents to be loaded. The beginning part of the loading file should have the definition of the class. The form is the follows.

%class <class name> (<name of column1> <name of column2> …)

Unless the definition of the class exists in the beginning part, an error will occur while loading the data. You should remember that the class has to be created in database before loading and the name of column in the file is the same as one defined in the schema. You also remember that the statement of the class defining has to be written in one line. That is, you have to write the whole definition without changing lines.

5) You should check the input file of OOSQL_LoadDB utility using OOSQL_CheckDataSyntax utility before loading. OOSQL_CheckDataSyntax utility verifies the grammar of the input file of OOSQL_LoadDB utility, and show cause and solution of error, if any.

6) When the size of data to be loaded is large, it is effective to load the data at once after storing them in the text file. You should place the class definition in the beginning part of the loading file, and subsequently list the documents. You simply arrange the contents to be input to each column irrespective of the tuple to create the data. Therefore, if any column is missed, an error can be occurred in loading the data. For example, if you load the data on the class having the fields of A, B, and C, an error will occur when no data for B field is.

Problems such as the below can happen.

“a1”
“b1”
“c1”
“a2”
“c2” <- Miss a data for B column
“a3”
“b3”
“c3”

In case the field is stored as the above, the stored results are wrong as the follows.

A	B	C
a1	b1	c1
a2	c2 (no data originally)	a3
b3	c3	

Accordingly, when you create a loading file, you have to give the information for all fields even to the field with no data. For example, you have to insert a black like “ ” when there is no data to be input for text type column, and insert a number like 0 for the number column. In this way, you have to make all the fields expressed in files to properly load data.

5.4. Index Creation

Objectives

Create the index for fast searching and keyword search.

Contents

You can apply the 'create index' of SQL statement to create an index for the string or numeric attributes. You can create the text index for the text field immediately when loading, or create it by batch processing after loading.

Core Contents

1) You can create an index for both the numeric/string field and the text field. You decide whether indexes for the number/letter fields is created or not, while creating an index for the text field is a prerequisite.

2) Create an index for the numeric or letter attribute using the 'create index' of SQL statement. In the same way as schema creation, you can execute SQL statements using isql or ODYSSEUS/Web-PHP utilities, or write a program with OOSQL API. Select one of them and execute SQL statement in the form of 'create [unique|cluster] index *index_name* on *table_name* (*attribute*);'. If the index is improperly created, you should execute '**drop** index *index_name*;' SQL statement to drop the index.

3) You should create the index for the numeric/letter field only in case making the index takes effect. That is, you should create the index only in the field with few tuples having the same values such as ID number or name. For a detailed explanation of this, refer to the Schema Creation part in Sector 5.2.

4) A large amount of data is recorded in the database during the index creation. This process is recorded in "log" to prevent system errors from destroying the database. However, when you create a lot of indexes at once, then the log grows too much larger and can exceed the given volume size. Therefore, you had better create one volume per one index, and then, execute **commit** command to prevent the volume size from growing excessively.

5) To create the index for a text field, you first extract keywords, and then, build text index using the extracted keywords. For further details, refer to ODYSSEUS educational materials, educational videos, and an 'ODYSSEUS/OOSQL Reference Manual.' You should read thoroughly the above materials and fully understand them before starting the operation.

6) When creating the database schema, you should consider the characteristics of keys to be used for creating an index as well as which field to be used for creating an index. For example, in case of a person's name, the exact matching will occur more frequent than the partial matching. In this case, the key used to construct an index should not be changed into a basic type. On the other hand, in case of a book name, the partial matching will occur more frequent than the exact matching, that is, only a part of whole title will be used for searching books. In this case, you have to create an index with the key that can be changed into a basic type. Moreover, in case of the year having the text type, it would be better to create an index with the key having the decimal number type. Therefore, when you construct the database, you have to use appropriate types for indexes or a keyword extractor by identifying the characteristics of keys to be used for creating indexes. Refer to Section 5.2 for the detailed explanation.

7) There are two methods of using 'deferred mode' (the mode of creating the index after inputting the data) for inputting the text data and constructing an index. One is to create it at once using OOSQL_MakeTextIndex, and the other is to use OOSQL_ExtractKeyword, OOSQL_MapPosting, OOSQL_SortPosting, OOSQL_BuildTextIndex, and OOSQL_UpdateTextDescriptor for each stage. These two methods perform the same task. That is, the first method is a script of putting all stages of the second method. For example, in the third stage of OOSQL_SortPosting, if the storage space for temporary files is insufficient to use Linux sorting, then an error will occur. With the latter method, you can preserve the more storage space and execute this program again.

8) Sufficient disk space (sum of posting file size * 2) is needed for temporary files like keyword extraction results in the text index creation process.

9) Text index creation process includes the keyword extraction process. The "/var/tmp" directory of UNIX is used to transfer the data in the process of extracting keywords. The created files at this directory are automatically deleted after extracting keywords for one text. In general, the usable disk space is not large and becomes insufficient if other programs (especially the programs that use the sorting function) use much space of "/var/tmp." Especially, if the data size of the field that you want to extract keywords is large, the available space is reduced very

rapidly. Therefore, you should check if there left a sufficient space in "/var/tmp" before constructing an index.

Notes:

1) Extracting keywords spends one of the most part of time in the process of creating the text index. In some cases, it is necessary to adjust the volume size of the database without modifying the contents of the database, or to change the device that stores the volume of the database. At this time, if you execute the keyword extraction process again, you may spend much unnecessary time to construct the database. In this case, you can construct a new volume without extracting keywords by using the file of which extension is SortedPosting and which stores the results of extracting keyword. For further details, refer to the educational videos and 'ODYSSEUS/OOSQL Reference Manual'.

5.5. Testing the DATABASE Volume

Objectives

Check if the database volume has been normally and correctly created.

Contents

To confirm the database volume is normal and correct, check if some queries are executed normally and correctly.

Core contents

1) Using "isql," an interactive SQL language provided by ODYSSEUS, check if the volume has been correctly created. Basic testing can be done by using "isql". If you want to confirm the volume's status in more detail, write programs using OOSQL API. To run isql, use the following procedure:

```
isql <db name>
```

When ISQL has been invoked successfully, the ODYSSEUS database becomes usable and awaits the user's SQL query. It can create/update/delete tables and tuples, as well as carry out queries for specific tables. All SQL queries must end with a semicolon (;). To end the program, enter "quit" without a semicolon; this commits the currently updated contents to the database and shuts down the ISQL program. Simple SQL sentences for testing can be found in the

example subdirectory of the ODYSSEUS directory. Test SQL sentences can be executed with ISQL as follows:

```
isql test < /odysseus_path/example/test.sql
```

If the result obtained is identical to the test result in the sample file test.result, this confirms that the ODYSSEUS database is operating normally.

test.sql contents

```
CREATE TABLE test1 ( a integer, b varchar(30) );

INSERT INTO test1 VALUES (10, 'abc');

INSERT INTO test1 VALUES (20, 'def');

INSERT INTO test1 VALUES (30, 'ghi');

SELECT * FROM test1;

UPDATE test1 SET b='aaa' WHERE a=30;

SELECT b FROM test1 WHERE a>15;

quit
```

test.result contents

```
First query results:
----------+----------+
         a|         b|
----------+----------+
        10|       abc|
        20|       def|
        30|       ghi|
----------+----------+

Second query results:
----------+
         b|
```

```
----------+
       def|
       aaa|
----------+
```

2) It is very difficult to finish the volume construction at the first attempt without any designing error. You may find the designing errors frequently in the volume test. In this case, you have to reconstruct the volume for the whole data. Thus, if you make the only one process include construction, test, modification, and reconstruction of the entire volume for large database, it takes so much time to finish the volume construction.

   You can solve this problem by the method of constructing a small test volume with a small portion of the data (e.g., 100 objects). It would be better to construct the volume for the whole data after you confirm the test volume construction is successfully finished. In this way, you can avoid waste of time needed by repeating trial and error.

3) You have to set up a log volume before testing a data volume. In general, you spend much time in constructing the volume. Accordingly, if the volume is damaged during the test, you will suffer from much loss of time. The volume damage means that the database falls into the malfunction status due to improper execution of queries, tester's mistakes, and system errors in the process of testing. You can avoid most of these problems by setting up the log volume. Therefore, you'd better set up the log volume before testing in order to avoid the damage caused by improper operations.

4) You should inspect the following matters in the process of testing the database volume.
   - Does each column contain all the necessary data?
   - Are the retrieved tuple data, which is randomly chosen and searched from the database, identical to the input data?
   - For each indexed column, is the search fast if you use the column as a key?
   - Is the number of results from the query identical to the number of results to be actually searched?
   - Can you search with a keyword extracted from the text search?
   - Is the integrated query, which contains various fields, properly executed?
   - Is the **`insert`** command successfully executed for a new random data and correctly reflected the execution result on the database?
   - Is the **`delete`** command successfully executed for a new random data and correctly reflected the execution result on the database?

- Is the **`update`** command successfully executed for a new random data and correctly reflected the execution result on the database?
- Are the other functions necessary for a search system correctly executed?

If there is any doubt about the above questions, be sure to solve it before starting the volume construction. Otherwise, you may fall into a critical situation that you have to create the database volume again, and in case of a large-sized database, you will waste much time.

5) In the testing process, you had better execute all the possible queries to be used for the system operation and record the results and the response times of them. That is, you should try to write queries, which may be input by users, and to executing them to confirm that the results are correct. It is also important to measure the time taken for searching. The time information can attribute to improving the search speed if you find designing errors later. Therefore, you should summarize the results systematically according to the types of queries.

Notes:

1) Even though you execute queries after setting up a log volume, the volume can be damaged if hardware problems such as a disk error occur. In this case, you may fall into a crisis that all the precious dada cannot be recovered not to mention of the waste of time. Therefore, a periodic backup of volumes is essential for preventing this disgraceful event. You should execute this backup process starting from the volume test process.