DATALESS MODELLING OF LARGE DATABASES

A Project Report

Submitted in partial fulfilment of the

requirements for the Degree of

Master of Engineering

in

Internet Science and Engineering

by

Gourav Kumar Sakargayan

Computer Science and Automation

Indian Institute of Science

BANGALORE – 560 012

JULY 2008

TO

My Family

Acknowledgements

It is a pleasure to thank all those who made this thesis possible. I would like to thank my guide Prof Jayant R.Haritsa for giving me the opportunity to work with him. Thank you sir, for your valuable guidance, for the patience with which you taught me, for your support and encouragement. I would also like to thank my DSL lab-mates for giving me lively and pleasant company. I would like to thank all my friends in IISc. Thanks, to all those whom I have not mentioned here, but have helped me in some or the other way in completing this work.

Abstract

Query optimization has been one of the major areas of research. A query optimizer, for query optimization, does not look for the relational data, all it need is metadata. Nowadays size of most of the commercial databases has become larger then terabytes. Anticipating this fact benchmarks are also getting bigger. It takes a huge amount of space and time to load benchmark. After generation of metadata, one does not need benchmark data. We deal with this problem by keeping metadata and throwing away data. Here, we present comprehensive methods for all well known databases to make them dataless. A dataless database gives optimizer an illusion of big benchmark without having any data. Our idea saves a huge amount of space and time and without affecting benchmark metadata or query optimization.

Contents

List of Figures

Keywords

Dataless Modelling, Large database, TPC-DS, Space, Time.

Chapter 1

Introduction

Nowadays common databases are of size several terabyte or more. Considering this point, a large amount of research is going on how to manage such a big amount of data. All database engines are making wide amount of changes to make their database engines to cope up with big databases. This introduces the term very large databases in database.

There have been two major areas of research in database systems. One is the analysis of data models into which the real world can be mapped and on which interfaces for different user types can be built. These kind of conceptual models include the hierarchical, the network, the relational, and a number of semantics-oriented models. These models have been reviewed by large number of researchers.

A second area of interest is the safe and efficient implementation of the DBMS. Computerized data have become a central resource of most organizations. Each implementation in database should guarantee that it would keep the data safe. Another measure, here, is efficient implementation of database systems. By efficient implementation of a database we mean that response time for a user query should be optimal. Query response time highly depends on the access path chosen by a database engine to retrieve information. In database engine this work is done by Optimizer.

Optimizer, for query optimization, does not look into the data. For optimization purpose all it need is Metadata. When a explain query comes optimizer looks only for the statistics of tables gives the unique plan for that query and does not execute

it. Explain command is a very useful feature, it reports plan without executing query. It has been largely in use to know the behaviors of query optimizers. Working with plans or explain queries require only metadata but generation of metadata requires data. Database engine generates metadata and store it into system tables so after generation of metadata we do not need data. All the while data sit there, without any use, and takes an enormous amount of space.

Here, we present methods for creating dataless database for all well known database engines. We define term dataless database as a database which gives optimizer an illusion that a fully loaded benchmark resides inside it but in reality the benchmark is empty.

A database with metadata and no data fulfill our two requirements:

1. Metadata.

2. Saving of space.

Chapter 2

Overview

An application is useless if it can not retrieve information efficiently. Databases have become a standard tool for hiding the computer user from details of secondary storage management. Applications store information into the databases, so performance of applications highly depends on database systems. For all queries an application generate, database generates a path to retrieve information. This path of information retrieval is called **PLAN**. There can be a large number plans for a query. Choosing a plan, which should give optimal performance, out of all qualified plans is called optimization.

For the query optimization purpose all database engines store all statistics, related to a database, into system tables. These tables collectively are called Metadata of a database. Cost based optimizer; for optimization purpose, looks for the metadata to get the statistics and based on that it chooses its plan. The most important key to success with the cost based optimizer is to carefully define and manage metadata. In order for the cost based optimizer to make an intelligent decision about the best execution plan for SQL, it must have information about the table and indexes that participate in the query. When the cost based optimizers know the size of the tables and the distribution, cardinality, and selectivity of column values, the cost based optimizer can make an informed decision.

Considering above points, metadata, stored in system tables, becomes highly sensitive. Thatś why in most of the cases metadata is not available directly for users. Dealing

with them is troublesome and risky.

Storage methods of all the database engines are different. Each database engine has its own method to store data. Not only data but they also define their metadata in the way which are very different from each others. So there is no generalized method to deal with metadata. So it is also not possible to define a generalized method which can create a dataless database for all engines.

Considering this point, we propose individual method for each database engine. In this report, we discuss the storage behavior of each database engine and then give a procedure for that engine to create dataless database.

For dataless database we have two modes of creation:

- Load Mode: Load mode creates database by loading benchmark data. In this mode, we load benchmark data and generate metadata. After generation we throw away data.

- Transfer Mode: This mode uses the metadata of benchmark loaded on the other system. This mode creates dataless database without loading benchmark data and saves a lot of time.

Chapter 3

Oracle

Oracle stores data in operating system files. These datafiles belong to a logical unit called tablespace. Oracle relies on user to create new datafiles. It does not create a new datafile and store data into already available datafiles. A size of datafile depends on operating system. However a user can create and add a new datafile to tablespace. [3] Growth of a datafile can be configured. A datafile grows when new data comes but does not shrink when we delete data. This are called space reclaiming problem. This feature becomes an obstacle because our goal is to get the space back from database engine when we delete data.

3.1 Tablespace

For Oracle tablespace is logical storage unit. It is a building block of oracle database. An Oracle database consists of one or more logical storage units, called tablespace, which collectively store data for database. Each tablespace in an Oracle database consists of one or more files called datafiles, which are physical operating system files. A database's data is collectively stored in the datafiles that belongs one of the tablespace of the database. Datafile belongs to exactly one tablespace. The simplest Oracle database would have one tablespace and one datafile but a database can have more than one tablespace having more than one datafiles in it.

3.2 Space Management

The size of table space is collectively the sum of size of all datafiles in tablespace. The size of a database is the collective size of the tablespaces that constitute the database. Enlargement in database can be done in three ways:

- Add a datafile to a tablespace
- Add a new tablespace
- Increase the size of a datafile

Here, we use third option with auto grow facility for datafile. This mode expands size of file as data comes.

3.3 Metadata utilities

Oracle provides a full package of utilities to gather database statistics. With DBMS_STATS we can view and modify optimizer statistics gathered for database objects. The DBMS_STATS utility does a far better job in estimating statistics, especially for large partitioned tables, and generates better statistics than Oracle ANALYZE. The utility **exec dbms_stats.gather_schema_s** creates database statistics with a rich number of options. Unlike the other databases Oracle give liberties to update its system tables to change metadata. This can be easily done by DBMS_STATS options. We can save statistics of any instance of database into a user table later we can restore this, stored in user table, instance of database over the current instance of database.

1. Install Oracle: Run setup.exe and follow the wizard.
2. Start→Programs→Oracle→OraHome→Application Development→SQLPlus.
   Login as system/manager (or any other valid user/password).
3. Create a new tablespace with name Test_DB using the following command:
   create tablespace Test_DB datafile Test_DB.dbf size 100m autoextend on next 1m autoallocate;
4. *Update Test_DB as the default table space alter database default tablespace Test_DB;*
5. *Issue command:*
   - *exec dbms_stats.create_stat_table ('system' 'SAVED_STATS');*
6. *Load Benchmark*
7. *Now we gather schema statistics by following two commands.*
   - *exec dbms_stats.gather_schema_stats (ownname⇒'SYSTEM, estimate_percent⇒100, method_opt⇒ for all columns size auto, statid⇒'PREVIOUS);*
   - *exec dbms_stats.gather_schema_stats (ownname⇒SYSTEM , estimate_percent⇒100, method_opt⇒ for all columns size auto , stattab⇒SAVED_STATS, statown⇒SYSTEM, statid⇒PREVIOUS);*
8. *Delete all the rows in each table. Do not delete tables.*
9. *To save statistics export it into a file using the following command exp username/password file=Test_DB.dmp log=emp.log rows=yes indexes=yes*

   This command exports the table space Test_DB into a file Test_DB.dmp.
10. *Delete table space using the following command: drop tablespace Test_DB including contents and datafiles cascade constraints; It will also delete the file Test_DB.dbf .*
11. *Create table space Test_DB again using following command:*

    create tablespace Test_DB datafile Test_DB.dbf size 100m autoextend on next 1m autoallocate;
12. *Now import Test_DB.dmp file using imp utility imp username/password file=Test_DB.dmp full=yes This creates all bench mark tables and SAVED_STATS table with benchmark tables having zero data.*
13. *Generate all statistics with following command: exec dbms_stats.gather_schema_stats (ownname⇒SYSTEM, estimate_percent⇒100, method_ opt ⇒for all columns size auto, statid⇒PREVIOUS);*
14. *Use the following command to reload all statistics exec dbms_stats.import_schema_stats (ownname⇒SYSTEM ,stattab⇒SAVED_STATS);*

Figure 3.1: Procedure for Oracle

3.4 Export and Import

EXP and IMP are export and import utilities present in the /ORACLE_HOME/bin directory and are installed when oracle is installed. These are executable that allows exporting or importing of data object of a database. These are complementary utilities so it helps to create a logical backup of data objects. Because of the special binary format, files which had been created by the export utility can only be read by import.

3.5 Procedure

Figure 1 illustrates the procedure for Oracle. After installation a tablespace is created and it has a datafile with .dbf. Update our new tablespace (here Test_DB) as default tablespace, which makes sure that each new object gets space into new tablespace. Step 5 creates benchmark table and their constraints which automatically go into default tablespace.

Step 7 has two important DBMS_STATS commands with different options; first command generates all the statistics and populates system metadata tables. Now, for this instance of database, loaded with benchmark, we have all metadata. We issue second command which saves metadata into user table 'SAVED_STATS'. Step 8 deletes all rows in each table without deleting tables. To make a back up copy of database we export our default tablespace (Here Test_DB) because default tablespace has all benchmark objects and SAVED_STATS table.

When we populate benchmark tables size of tablespace, size of .dbf file attached with tablespace (Test_DB.dbf), grows large enough to accommodate benchmark data. But after deleting rows of benchmark table size of tablespace does not shrink automatically. So, to reclaim our space back we delete tablespace and recreate it with same command which procedure has used in step 3. This deletion of tablespace gives space back to the operating system.

Now, procedure imports stored tablespace data object, which create all benchmark tables, though non-populated, and SAVED_STATS table with metadata stored in it. Step

13 generates metadata for non populated benchmark but it helps us to create system tables to store metadata. In last step we load metadata from SAVED_STATS, which is metadata for populated benchmark, using SAVED_STATS utility. This command populates system tables with database instance of populated benchmark which gives illusion to Cost based optimizer that benchmark is fully populated though it is not.

3.6 Transfer Mode

Figure 2 illustrates Oracle-Transfer-Mode procedure. Oracle Transfer mode assumes that we already have a the dump file for benchmark. This dump can be generated fro m other dataless database for same benchmark. Procedure does not load data but import metadata from a given dump file.

1. Install Oracle: Run setup.exe and follow the wizard.
2. Start→Programs→Oracle→OraHome→Application Development→SQLPlus. Login as system/manager (or any other valid user/password).
3. Create a new tablespace with name Test_DB using the following command: *create tablespace Test_DB datafile Test_DB.dbf size 100m autoextend on next 1m autoallocate;*
4. *Update Test_DB as the default table space alter database default tablespace Test_DB;*
5. *Issue command:*
   - *exec dbms_stats.create_stat_table ('system' 'SAVED_STATS');*
6. *Load Benchmark*
7. *Now we gather schema statistics by following two commands.*
   - *exec dbms_stats.gather_schema_stats (ownname⇒'SYSTEM, estimate_percent⇒100, method_opt⇒ for all columns size auto, statid⇒'PREVIOUS);*
   - *exec dbms_stats.gather_schema_stats (ownname⇒SYSTEM , estimate_percent⇒100, method_opt⇒ for all columns size auto , stattab⇒SAVED_STATS, statown⇒SYSTEM, statid⇒PREVIOUS);*
8. *Now import Test_DB.dmp file using imp utility imp username/password file=Test_DB.dmp full=yes This creates all bench mark tables and SAVED_STATS table with benchmark tables having zero data.*
9. *Generate all statistics with following command: exec dbms_stats.gather_schema_stats (ownname⇒SYSTEM, estimate_percent⇒100, method_ opt ⇒for all columns size auto, statid⇒PREVIOUS);*
10. *Use the following command to reload statistics exec dbms_stats.import_schema_stats (ownname⇒SYSTEM ,stattab⇒SAVED_STATS);*

Figure 3.2: Procedure for Oracle-Transfer-Mode

Chapter 4

DB2

DB2 stores information in containers [4]. These containers usually are operating system files. But we can also create containers of other storages. DB2 manage space in logical tablespace. For each new data object it creates a new datafile. So if we manipulate datafile of one data object it does not affect others.

For creation of dataless databases this is a very useful feature of DB2. This helps us lot in creation of a dataless DB2 database.

4.1 Tablespace

Tablespace gives logical layout of set of operating system files that is used to store the records in one or more tables. In crude way, table space is a storage structure containing tables, indexes, large objects, and system data. Tablespace works as one of the building block of database. They allow you to assign the location of database and table data directly onto containers. This can provide improved performance and more flexible configuration.

1. Install DB2 with complete support for Java
   - Run setup.exe and follow the wizard.
   - In the Configure DB2 Instances screen, you can customize the Protocol settings (including port number and service name) and Startup settings of the instance.
2. If the DB2 service is not already running, start it with the command db2start.
3. Open a command window using

   Start→Programs→IBM DB2→Command Line Tools→Command Window
4. Create database using following command
   - *create database Test_Dataless on DRIVE:' using codset IBM-1252 using system catalog tablespace managed by database using (file 'PATH SIZE) temporary tablespace managed by database using (file 'PATH' SIZE);*

   Here, DRIVE and PATH specify physical location of database.
5. Create schema for benchmark tables along with constraints on tables.
6. DB2 creates a separate file for each table. Take backup copy for all tables in benchmark (auxiliary copy) from the location

   DRIVE:\DB2\NODE0000\SQL00001\SQLT0001.0.
7. *Load bench mark data and generate statistics.*
8. *Overwrite file in given location with the backup copy. \DB2\NODE0000\SQL00001\SQLT0001.0*

Figure 4.1: Procedure for DB2

4.2 Containers

DB2 occupies the portion of disk space to create containers. A container can be a directory name, a device name, or a file name.

Storage management in databases is very critical. DB2 has 2 kinds of storage management:

1. System managed storage
2. Database managed storage

In System managed storage mode , all DB2 objects are managed by the operating system using the file system. Database managed storage has already available space in form of containers. We can add any number of containers to database.

DB2 has three types of tables:

1. user tables
2. catalog tables
3. Temporary tables

In system managed storage system takes care of all the storage management problems. We can assign different type of storage management mode to tables of each type. Here we assign Database managed storage mode to temporary tables and catalog tables. For user tables we use system managed storage mode. System managed storage mode creates a container (an operating system file) for each database object and stores data in it. Since we assign system managed storage for user tables it creates a new container.

4.3 DB2 directories and files

When we create a database, information about the database including default information is stored in a directory hierarchy. A directory hierarchy structure is created on the

location specified by create database command. In directory, specified in create database command, a subdirectory with name NODE0000 is created. Inside NODE0000 directories named SQL0000* (* can be a number which starts with 1, like SQL00001, SQL00002, and so on) stores all the information about a database. These subdirectories differentiate databases created in this instance. With a lot of other subdirectories SQL0000* contain SQLT subdirectories. The SQLT* subdirectories contain the default System Managed Space (SMS) table spaces required for an operational database. There are three default table spaces:

- SQLT0000.0 subdirectory contains the catalog table space with the system catalog tables.
- SQLT0001.0 subdirectory contains the default temporary table space.
- SQLT0002.0 subdirectory contains the default user data table space.

Each subdirectory has a file called SQL*.DAT stores information about each table that the subdirectory or container contains. The asterisk (*) is replaced by a unique set of digits that identifies each table. It also has a file SQL*.INX which contains index table data.

4.4 Procedure

Figure 3 illustrates the procedure for DB2. Step 4 creates a database with a create database command. It assigns database managed storage mode to catalog tables and temporary tables and a system managed storage mode for user tables. Two containers with the given names and sizes are created for catalog tables and temporary table. Create database also creates subdirectory container named SQLT00002 at location

```
/<PATH>/NODE0000/SQL00001/.
```

In following step five we create schema tables along with constraints. Since, user tables are managed by system managed storage mode database engine creates SQL*.DAT

and SQL*.INX file for each table. We keep copy of each SQL*.DAT and SQL*.INX files and call it auxiliary copy.

Now procedure loads data and generates statistics. Loading data populate SQL*.DAT and SQL*.INX files with data and index related information. After step seven Container for catalog tables has metadata. Now we can overwrite all populated SQL*.DAT and SQL*.INX files with auxiliary copy which are non-populated. This gives us our space back with required metadata.

4.5 Transfer Mode:

Figure 4 illustrates DB2-Transfer-Mode procedure. DB2 Transfer mode assumes that we already have a catalog container for benchmark. This catalog container is taken from other dataless database for same benchmark. Procedure does not but overwrite catalog file with auxiliary catalog file. This new catalog file gives all required metadata for benchmark.

1. Install DB2 with complete support for Java
    - Run setup.exe and follow the wizard.
    - In the Configure DB2 Instances screen, you can customize the Protocol settings (including port number and service name) and Startup settings of the instance.
2. If the DB2 service is not already running, start it with the command db2start.
3. Open a command window using Start→Programs→IBM DB2→Command Line Tools→Command Window
4. Create database using following command
    - *create database Test_Dataless on DRIVE:' using codset IBM-1252 using system catalog tablespace managed by database using (file 'PATH SIZE) temporary tablespace managed by database using (file 'PATH' SIZE);*

    Here, DRIVE and PATH specify physical location of database.
5. Create schema for benchmark tables along with constraints on tables.
6. Overwrite catalog container with auxiliary copy.

Figure 4.2: Procedure for DB2-Transfer-Mode

Chapter 5

Sybase

Sybase maps data objects into data devices. These devices are well allocated memory arrays. Sybase manages these devices by creating a logical picture called segments. Each logical segment can have one or more data devices. We need to allocate space for data devices while creating. Later we can increase the size (does not grow automatically) of data device but can not reduce it.

5.1 Data device

Sybase occupies the physical portion of the disk to create data device. In crude way, data device is a operating system file with already allocated space which can be used by database to store data. A data device can be allocated to more than one database. Data device may occupy entire available space on the physical disk, but still has unused space on the device.

5.2 Segment

A segment is logical space layout of disk space allocated to the database. Segments are created within a particular database from the database devices already allocated to that database. A segment can have any number of data devices. A segment belongs to one

1. Install Sybase by running setup.exe and following the wizard.
2. Open Sybase Interactive SQL window using
   - Strat Programs Sybase Adaptive Server Enterprise Interactive SQL
3. Create four database devices for default data (Default_Data), benchmark data (Benchmark_Data), other tables (dummy_Tables) and Log (Default_Log).
4. Create a database Test_Dataless which uses Dafault_Data device for data and Default_Log device for log using following command:
   - *Create database Test_Dataless on Default_Data = size log on Default_Log =size*

   Here, size specifies volume available for database on data device in MB.
5. Create two news segments for benchmark tables and other tables using the following syntax
   - *exec sp_addsegment 'Segment Name','Database name','Database device'*

   For example, the command to create data segment for benchmark data is:
   - *exec sp_addsegment 'Data_Segment','Test_Dataless','Benchmark_Data'*
6. Create the schema of the benchmark tables along with constraints.
7. Move the benchmark tables to the data segment using the following syntax
   - *exec sp_placeobject 'Segment_name', 'Table_Name'*
8. Enable allow updates to system tables using following command
   - *sp_configure "allow updates to system tables ", 1*
9. Create an auxiliary copy of the data device.
10. Increase size of the device Benchmark_Data. Make data device large enough to accommodate data of benchmark.

    For example, to increase the size of data device, the command is
    - *disk resize name='Benchmark_Data',size ='newsize'*
11. Load benchmark and update the statistics for tables.
12. Replace the data device file with the old small auxiliary copy of data device file.
13. Now create any dummy table
14. Move dummy table to other segment.

Figure 5.1: Procedure for Sybase

database only. When we first create a database, database engine creates three segments in the database:

1. System: Stores database's system tables.
2. Logsegments: Stores this database's transaction log.
3. Default: Stores all other database objects.

We can create any number of segments and can move database objects from one segment to another segment.

First we add one or more data devices and allocate space to adaptive database engine. Now create a database which can occupy different amount of space on each of several database devices. Different segments for a database gives logical layout of devices allocated to the database. Database takes space from all the allocated data devices and creates a logical picture of available picture. So here we need a mechanism which creates illusion of physical data which does not exist.

5.3 Procedure

Figure 5 illustrates the procedure for Sybase. Creating two different segments for other tables and benchmark data tables gives us flexibility to deal with them individually because each segment gives a different logical layout. After installation procedure creates 4 data devices (for benchmark data, default, other table and log data) and ensures that initial size for data device for benchmark data should not be large.

Initially a database has two segments we add two more segments to database for benchmark data and other tables. Sybase does not give any facility to set any segment as default segment but we can move tables from one segment to another so after creating schema for benchmark data we move each table from default segment to benchmark

data segment. We make an auxiliary copy of benchmark data device and make it large enough such that it can accommodate benchmark data. Load benchmark and update the statistics for tables.

At this instance, we have benchmark with metadata which is in default segment. To get our space back we overwrite resized big data device with our small auxiliary copy. This gives us our space back. In last two steps, we create dummy table and move it to other segment. This step ensures that data for other tables go into other segment and does not overlap with benchmark data segment.

5.4 Transfer-Mode

Transfer-Mode mode for Sybase is not possible. Sybase does not give any information about mapping between physical to logical picture of data. So, we can not manipulate its logical storage picture without loading benchmark.

Chapter 6

SQL Server

SQL Server uses datafile to store data and groups these files into filegroups. These filegroups present a logical view of datafiles to the SQL Server engine. While creating a new database object only one default filegroup becomes active. A database object goes into default filegroup. This is very helpful feature for us to create dataless SQL Server database.

6.1 Datafile

SQL Server maps tables in a database over a set of operating system files. It never stores data and log information into same file but data and metadata may go to same file. The data files are combined into filegroups.

6.2 Filegroup

A filegroup is a collection of datafiles. Log files are managed separately. Files in a filegroup will not auto grow unless there is no space available on any other file in the filegroup. There are two types of filegroups:

- Primary

1. Install MS SQL Server. Run setup.exe (or Autorun.exe) and follow the wizard for installation of MS SQL Server. Install with SQL Server authentication. You will be asked for the password for username sa.
2. Open Enterprise Manager using Start Programs Microsoft SQL Server Enterprise Manager.
3. Create database using the syntax:
   - *CREATE DATABASE database_name;*
4. Create two new file groups.
   - *ALTER DATABASE Test_Dataless ADD FILEGROUP Test_Group1*
5. Add one file to each file group. Here, Test_File1 and Test_File2 are file names for Test_Group1 and Test_Group2 respectively.
   - ALTER DATABASE Test_Dataless ADD FILE (NAME = Test_File1,FILENAME = 'PATH and Test_File1',SIZE = 5MB, FILEGROWTH = 5MB) TO FILEGROUP Test_Group1;
6. Make one file group default file group. For example, to make Test_Group1default file group the command is
   - *ALTER DATABASE Test_Dataless MODIFY FILEGROUP Test_Group1 DEFAULT*
7. Make back up copy of datafile of default filegroup (here Test_File1).
8. Create benchmark tables. Here, we create only the schema of the benchmark tables along with their constraints.
9. Use command sp_help tables name to check that all benchmark tables are located in file group Test_Group1.
10. Make copy of Test_File1 located at PATH. Now we call it auxiliary copy of Test_File1.
11. Populate all benchmark tables.
12. Update default file group as second file group.
    - *ALTER DATABASE Test_Dataless MODIFY FILEGROUP Test_Group2 DEFAULT*
13. Create statistics for all benchmark tables.
14. Replace Test_File1 with its auxiliary copy.

Figure 6.1: Procedure For SQL Server

- User-defined

Each database has only one primary filegroup, only one default filegroup, and zero or more user-defined filegroups. We can update any file group as Default file group. The primary filegroup contains the entire system object like system tables, system stored procedures, extended stored procedures. We can not moves system objects from the Primary file group so its necessary to keep a primary file group.

6.3 Procedure

Figure 6 illustrates the procedure for SQL Server. Database engine manages space by creating files. These files are organized in form of filegroups. After installation when we create new database, engine creates a primary filegroup. All pages for the system tables are allocated in the primary filegroup. For benchmark tables and other tables we create two new filegroups. step 4 and 5 creates two new file groups and assign a data file to each of them. Set one of the filegroup as the default file group. Now each new database object will get space into the default file group. sp_help reports information about a database object so after creating benchmark schema we can check that it is located in default tablespace or not.

Datafile for default group, now, holds definition of the benchmark tables so make a backup copy of it. Datafile of default filegroup grows to accommodate the data of benchmark. Update second filegroup as default filegroup and generate statistics. We have updated second filegroup as default filegroup now each new database object will go to second filegroup. At this instance system table, in Primary filegroup, has all required metadata of benchmark data. Overwriting, last step, datafile of filegroup1 gives us space back with all metadata.

6.4 Transfer-Mode

Transfer-Mode mode for SQL-Server is not possible. SQL-Server stores metadata inside primary filegroup. Primary filegroup in SQL-Server is very critical and is not accessible for users. So without touching primary filegroup we can not make it dataless.

Chapter 7

PostgreSQL

PostgreSQL storage manager has a great resemblance with DB2 storage manager. It creates a new container for each data object. It helps us a lot because manipulating one data object container does not affect others. It also manages these containers using logical unit tablespace.

Optimization in PostgreSQL, unlike other database engines, is not fully metadata dependent. While optimizing a query, PostgreSQL gets size of table by reading the size of the container of the table in question. It makes impossible to change the size of containers without affecting optimization. Above behavior of PostgreSQL makes our work quite heavy. Since PostgreSQL is an open source, the code PostgreSQL has been modified to create a dataless PostgreSQL database.

7.1 Tablespace

When we create a new database PostgreSQL create a whole directory structure. On the specified path (asks during installation), PostgreSQL creates base container (a directory). This base container act like the home for tablespace. By default all tablespace gets space inside the base container. PostgreSQL creates an individual datafile for each database object. These datafiles resides in tablespace directory and has no extension.

7.2 Containers

PostgreSQL manages memory in the form of containers. Each container belongs to one tablespace and size of it can grow automatically. Container can have any number of datafiles and these datafile stores data objects.

7.3 Metadata

Optimization in PostgreSQL is not fully metadata dependent. It stores metadata in the form of tables pg_statistic is one of the most important metadata table. The table pg_statistic stores statistical data about the contents of the database. Entries are created by ANALYZE and subsequently used by the query planner. There is one entry for each table column that has been analyzed. All the statistical data is inherently approximate, even assuming that it is up-to-date.

By analyzing all the metadata tables we can conclude that there is no table which stores the information related to size of a relation. PostgreSQL does not store any information related to size of table but while generating a plan it read the size of datafile which belongs to that relation.

7.4 Procedure

Figure 7 illustrates the procedure for PostgreSQL. Above fact, that optimization is not fully metadata dependent, makes it impossible, with existing source code, to manipulate the size of datafile without affecting optimization. PostgreSQL is an open source database. The code has been modified such that it makes possible to create a dataless PostgreSQL database using procedure in figure 7.

We have modified PostgreSQL such that we can change its behavior for some particular tables. For tables, which we want to make dataless, optimizer, while generating a plan, does not look for the size of datafiles. This added feature helps us to create dataless database. Using this feature we can easily fake the size of datafiles.

1. Download files from this link [2].

2. Overwrite downloaded three file of PostgreSQL source code.

   ```
   Postgres.c-/src/backend/tcop/postgres.c
   Plancat.c-/src/backend/optimizer/util/plancat.c
   Explain.c-/src/backend/commands/explain.c
   ```

3. Install PostgreSQL with following commands:

   ```
   ./configure
   gmake
   gmake install
   mkdir  PATH/pgsql/data
   PATH/pgsql/bin/initdb -D PATH/pgsql/data
   PATH /pgsql/bin/postmaster -D PATH /pgsql/data
   PATH/pgsql/bin/createdb Test_DB
   PATH/pgsql/bin/psql Test_DB
   ```

   Here PATH can be any user specified location.

4. Create benchmark schema along with constraints.

5. Make a backup copy of datafile of location

   ```
   PATH/pgsql/data/base.
   ```

6. Populate benchmark tables and generate statistics using **ANALYZE** command.

7. For each benchmark table run following command:

   - *Dataless tablename;*

8. Overwrite datafiles at follwing location with our non-populated backup copy.

   ```
   PATH/pgsql/data/base.
   ```

Figure 7.1: Procedure for PostgreSQL.

Procedure shows the link for modified code. From the given link one can download the modified files of PostgreSQL source code. Overwrite these files in PostgreSQL source code at gives locations:

`Postgres.c-/src/backend/tcop/postgres.c`

`Plancat.c-/src/backend/optimizer/plancat.c`

`Explain.c-/src/backend/commands/explain.c`

These locations can be located easily in source code directory. Now procedure installs the PostgreSQL using commands given in the procedure.

After installation create benchmark schema along with constraints. PostgreSQL creates a new datafile for each benchmark tables and these datafiles can be located easily at location given in procedure. Make a backup copy of benchmark datafiles. Populate benchmark tables and generate statistics using ANALYZE command. Now for all the benchmark tables, which we want to make dataless, run this command:

- *Dataless tablename;*

This puts table into dataless mode. Now we can overwrite datafiles at location

`PATH/pgsql/data/base.`

with our non-populated auxiliary copy. Last step in procedure gives space back to user without disturbing metadata.

7.5 Transfer mode

Transfer mode in PostgreSQL is very simple. Here, we assume that we have all data object for metadata tables and call it auxiliary copy. We can take these metadata objects easily from already loaded benchmark system. Now, overwrite metadata object with auxiliary copy and for all the benchmark tables run this command:

- *Dataless tablename;*

Chapter 8

Experimental Results

Figure 8 gives comparison Normal Mode, Load Node and We have successfully created the dataless database for above listed engines. The noticeable feature of our procedure is the size of dataless database does not depend on size of benchmark. Instead in some cases it takes constant space and for other cases it is directly proportional of number of relations in benchmark. Here, we list the results of our experiments for each database:

8.1 Oracle

We have done experiment for Oracle using Oracle 10g and TPC-H benchmark. Here the mapping from populated database to dataless database takes constant space. In our experiment dataless database takes 10 MB which is the minimum size of a datafile.

	Normal Mode		Load Mode		Transfer Mode	
	Space	**Time**	Space	**Time**	Space	**Time**
Oracle	1.7GB	**8Hrs**	10MB	**8Hrs**	10MB	**15Min**
DB2	2.1GB	**17Hrs**	32KB	**17Hrs**	32KB	**15Min**
SQL-Server	2.1GB	**11Hrs**	4MB	**11Hrs**	-	**-**
Sybase	2.4GB	**19Hrs**	4MB	**19Hrs**	-	**-**
PostgreSQL	1.2GB	**3Hrs**	32KB	**3Hrs**	32KB	**15Min**

Figure 8.1: Performance of Load and Transfer Mode.

	Normal Mode		Load Mode		Transfer Mode	
	Space	**Time**	Space	**Time**	Space	**Time**
Oracle	56GB	**190Hrs**	10MB	**190Hrs**	10MB	**15Min**
DB2	88GB	**410Hrs**	96KB	**410Hrs**	96KB	**15Min**
SQL-Server	108GB	**360Hrs**	4MB	**360Hrs**	-	**-**

Figure 8.2: Performance of Load and Transfer Mode for Large Database

8.2 DB2

We have done experiment for DB2 using IBM DB2 and TPC-H benchmark. Here, size of dataless database depends on the number of relation in benchmark. We can get the size of dataless database by multiplying minimum size of datafile by number of relations. For TPC-H, eight relations, and having minimum size of datafile 4KB the size of dataless database was 32KB.

8.3 Sybase

Sybase again map to a constant number. We did experiment for using Sybase Adaptive Server 4.3 and TPC-H benchmark. Minimum size of a data device is 2MB and we create two data device experiments takes 4MB of space.

8.4 SQL Server

We use SQL 2005 and TPC-H benchmark for our experiments. Server SQL Server also gives constant space mapping from populated database to dataless database. Size of minimum datafile in SQL Server is 2MB. For our experiment, we create two datafiles so size of dataless database is 4 MB.

8.5 PostgreSQL

We have done experiment for PostgreSQL using PostgreSQL 8.1.11 and TPC-H benchmark. Here, size of dataless database depends on the number of relation in benchmark. We can use the method for DB2 here also. For TPC-H, eight relations, and having minimum size of datafile 4KB the size of dataless database was 32KB.

Chapter 9

Conclusions

Here, we introduce the problem of large benchmarks for the community which is working on query optimization. To deal with this problem we give methods for creating dataless database. We have also shown that the generalized method to make any database engine dataless is not possible. In absence of generalized method, we dealt with the problem considering one engine at a time. Report shows the different procedure for five well known database engines. We have tested each procedure successfully and got exciting results. We also showed that PostgreSQL optimizer is not a fully metadata dependent optimizer. Source code of PostgreSQL has been changed to make it fully metadata dependent.

References

[1] N. Reddy and J. Haritsa *Analyzing Plan Diagrams of Database Query Optimizers* , Proc. of 31st Intl. Conf. on Very Large Data Bases (VLDB), Trondheim, Norway, August 2005, pgs. 1228-1240

[2] http://dsl.serc.iisc.ernet.in/projects/projects.html

[3] http://www.oracle.com

[4] http://www.ibm.com/db2

[5] http://www.sybase.com/products/databasemanagement

[6] http://www.microsoft.com/SQL/

[7] http://www.postgresql.org