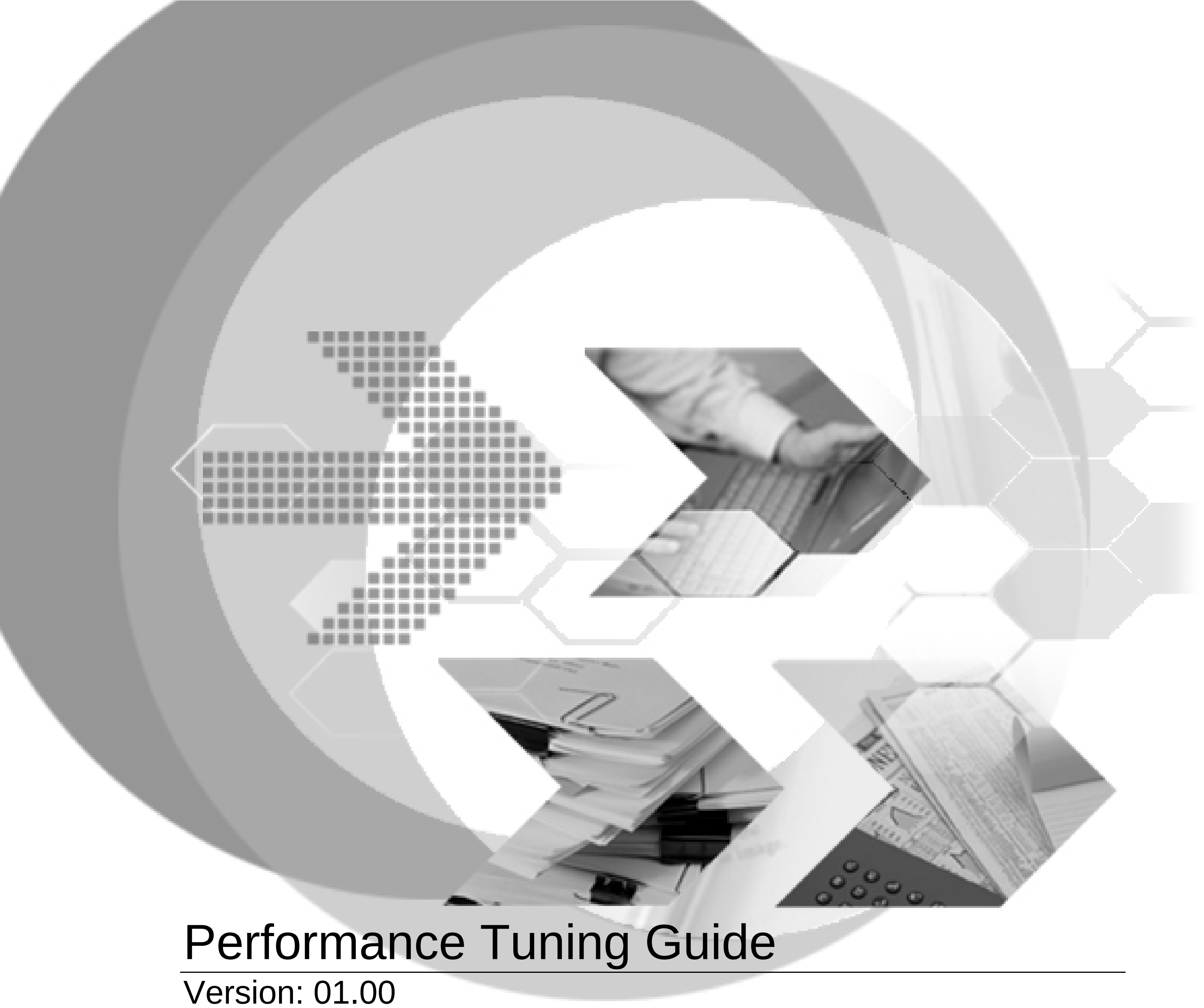


Performance Tuning Guide

Version: 01.00

Document No: 43/DBM43-T01222006-01-PERT
Author: DBMaker Production Team Aug 11, 2005
LingAn Computer Engineering CO.

Print Date: Aug 05, 2005

凌羣電腦
THE SYSCOM GROUP

Table of Content

1. Introduction

When you use any Database to develop a Database application system, one of things you most concerned about is the system performance.

Generally speaking, there are many factors affect the performance of DBMaster. We can see them from the following figure.

Application System	**Query Optimization**
	Concurrent Process
	Application System Architecture
	Database Model Design **(Tablespace、Table、Index、stored command、Stored procedure、Trigger)**
Database System	**Daemon** **(Auto-commit、Checkpoint、Update statistic、Backup server、Replication)**
	Memory Allocation
	Disk I/O **(Database Data partition)**
OS	**(File system, Raid)**
Hardware	**Network**
	I/O
	Memory
	CPU

Figure 2.1

(1) Application

It comprises writing queries that limit the use of stored commands or searches for procedures. Designing good schema or developing an application with better utilities can both significantly increase application performance.

About how to use index to increase the performance of DBMaster, We will discuss it at chapter 5.

Another part of Application is Concurrent Process. Obviously, minimizing lock contention and avoiding deadlocks can increase throughput. Shorter transactions can promote concurrency, but it may be degrade database performance. It will be expounded in chapter 6.

But in this article we only expound the key factors in the right of the Figure 1.1 that affect the performance of DBMaster.

(2) Database System

This is the most important thing that we will discuss in this article. It includes **Disk I/O**, **Memory Allocation** and **Daemon**. Make sure there are enough physical memory for DCCA and few I/O access times. We will discuss the details in the Database Tuning chapter.

(3) OS

Chose a suitable OS can improve the performance of whole system, please select the OS that special designed for the support of application disposal and database as possible as you can, for example the series of Linux D-Class 4 made by xTeam Corp. In addition, about hard disk which support the technical Raid, please select different Raid Level for different data type in DBMaker, you can put data file into Raid 1,3,5, and put journal file into Raid 0,that can guarantee safeness and a high efficiency.

(4) Hardware

It is the basic factor not only affects the performance of DBMaster, but also affects the whole PC's.

- **CPU:** A faster CPU or more CPUs will help executing performance.
- **Memory:** More memory will hold more cached data, so that it will reduce I/O access times.
- **I/O:** Faster hard disks will improve the I/O throughput and more hard disks will promote the I/O concurrency.
- **Network:** Speeding up network transmission will reduce the response time for users. Using only network protocols required will reduce load balancing of the operating system.

  Obviously, enhancing the hardware can greatly improve the overall database system performance.

2. Database Performance Tuning

DBMaster Database Performance Tuning chapter is an aid for people responsible for the operation, maintenance, and performance of DBMaster. This chapter describes detailed ways to enhance DBMaster performance. DBMaster is a highly tunable database system. Tuning DBMaster will increase its performance level to satisfy individual needs. This chapter presents the goals and methods used in the tuning process, and demonstrate how to diagnose a system's performance.

2.1 The Tuning Process

This section provides information on tuning a DBMaster Database system for performance. When considering instance tuning, care must be taken in the initial design of the database system to avoid bottlenecks that could lead to performance problems. Before tuning DBMaster, you must define your goals for improving performance. Keep in mind that some of your goals may conflict. You must decide which of the conflicting goals are most important to you. The list below shows some of the possible goals when tuning DBMaster.

When initial design a Database, you need to consider:

- Allocating memory to database structures
- Determining I/O requirements of different parts of the database
- Tuning the operating system for optimal performance of the database

After the database instance has been installed and configured, you need to monitor the database as it is running to check for performance-related problems.

Before tuning DBMaster, you need to define the following important goals:

- Improving the performance of SQL statements.
- Improving the performance of database applications.
- Improving the performance of concurrent processing.
- Optimizing resource utilization.

After determining the goals, you are ready to begin tuning DBMaster.

Start by performing the following steps:

- Monitor database performance
- Tuning I/O.
- Tuning memory allocation.
- Tuning concurrent processing.
- Monitor database performance and compare with previous statistics.

The methods used to perform tuning in each of these steps may have a negative influence on other steps. Following the order shown above can reduce this influence. After performing all of the tuning steps, monitor the performance of DBMaster to see whether the best overall performance has been achieved.

Before tuning DBMaster, make certain that the SQL statements are written efficiently and the database applications employ good design. Inefficient SQL statements or badly designed applications can have a negative influence on database performance that tuning cannot improve. To write efficient statements and applications, refer to the SQL Command and Function Reference and the ODBC Programmer's Guide.

2.2 Monitoring a Database

This section shows how to monitor information about the status of a database, including resource status, operation status, connection status, and concurrency status. This section also shows how to kill a connection.

2.2.1 The Monitor Tables

DBMaster stores the database status in four system catalog tables: SYSINFO, SYSUSER, SYSLOCK, and SYSWAIT.

The **SYSINFO** table contains database system values including total DCCA size, available DCCA size, number of maximum transactions, and the number of page buffers. It also includes statistics on system actions such as the number of active transactions, the number of started transactions, the number of lock and semaphore requests, the number of physical disk I/O, the number of journal record I/O, and more. Use this table to monitor the database system status, and use the information to tune the database.

The **SYSUSER** table contains connection information, including connection ID, user name, login name, login IP address, and the number of DML operations that have been executed. Use this table to monitor which users are using a database.

The **SYSLOCK** table contains information about locked objects, such as the ID of the locked object, lock status, lock granularity, ID of the connection locking this object, and more. Use this table to monitor which objects are being locked by which connection, and which users are locking which objects.

The **SYSWAIT** table contains information on the wait status of connections, including the ID of the connection that is waiting and the ID of the connection it is waiting for. Use this table to monitor the concurrency status of connections. Once a connection is waiting for those resources locked by an idle or dead connection, you can determine which connection is locking those objects from this table. Then you can kill the idle or dead connection to release the resources.

Browse these four system catalog tables in the same way ordinary tables are browsed.

Example

SQL SELECT command used to browse the SYSUSER table:

```
dmSQL> SELECT * FROM SYSUSER;
```

Refer to DBA manual for more information on these four system catalog tables.

2.2.2 Killing Connections

A connection should be killed when the connection is holding resources and is idle for a long time, or when the resources are urgently required. In addition, all active connections should be killed before shutting down a database. Before killing a connection, browse the **SYSUSER** table to determine its connection ID.

Example 1:

To kill the connection for Eddie, retrieve the connection ID first:

```
dmSQL> SELECT CONNECTION_ID FROM SYSUSER WHERE USER_NAME = 'Eddie';
CONNECTION_ID
=============
352501
```

Example 2:

Then to kill the connection for Eddie use:

```
dmSQL> KILL 352501;
```

2.3 Tuning I/O

Disk I/O requires the most time in DBMaster.

To avoid disk I/O bottlenecks, perform the following:

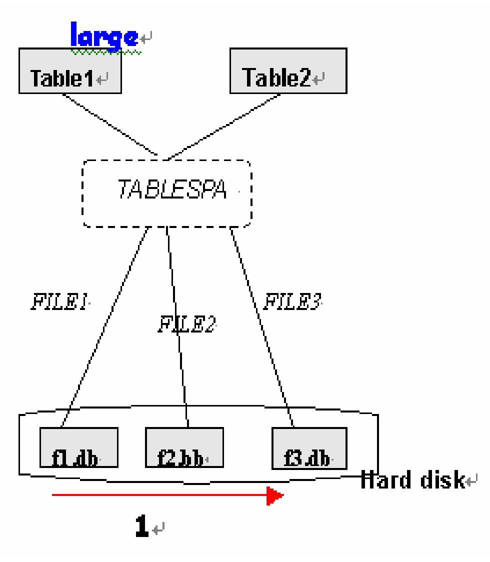
- Determine data partitions.
- Determine journal file partitions.
- Separate journal files and data files onto different disks.
- Use raw devices.
- Pre-allocate space in an auto-extend Tablespace.
- Turn I/O and checkpoint daemon on.

2.3.1 Determining Data Partitions

You can use Tablespace to partition data instead of storing all of the data together. If Tablespace are used properly, DBMaster will have greater performance when performing space management functions or full table scans. Small tables that contain data of a similar nature can be grouped in a single Tablespace, but very large tables should be placed in their own Tablespace.

You can achieve speed improvement in disk I/O by using disk striping. Striping is the practice of separating consecutive disk sectors so they span several disks. This can be used to divide the data in a large table over several disks. This helps to avoid disk contention that may occur when many processes try to access the same files concurrently.

Example:



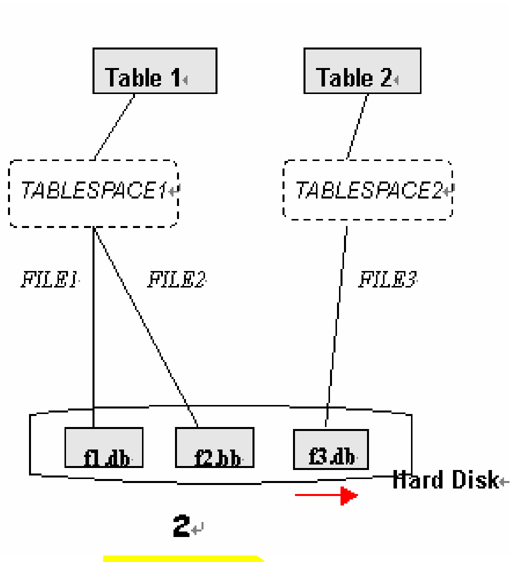


Figure2.1 *Figure 2.2*

We can see from the figure 2.1, if the table1 is too large while the table 2 is small. In fact, in the hard disk table1 might be stored into f1.db and f2.bb. In that case, when you want to select something from table2, you must scan the f1.db, f2.bb first. It is unnecessary to scan too many files.

But in the figure 2.2, that case can't meet, it only needs to scan f3.db, because the Tablespace partitioned reasonably. You can see great improvement on performance obviously.

2.3.2 Determining Journal File Partitions

DBMaster gives the flexibility to use one or more journal files. A single journal file is easier to manage, but using multiple journal files has some advantages as well. If you run DBMaster in backup mode and use the backup server to perform incremental backups, using multiple journal files can improve the performance of incremental backups. Only full journal files will be backed up. In addition, spreading multiple journal files across different disks can increase disk I/O performance.

The size of the journal fill will affect the interval of the backups. You may determine the size of journal files by examining the needs of transactions. However, if you run DBMaster in backup mode and perform backups according to the journal full status, the journal size will also affect the backup time interval. A larger journal file increases the interval between backups.

2.3.3 Separating Journal Files and Data Files

Separating the journal files, data files and the system temp file onto different disks will increase disk I/O performance, permitting files to be accessed concurrently to some degree. If the disks have different I/O speeds, consider which files to put on the faster disks. In general, if you run on-line transaction processing (OLTP) applications often, putting the journal files on the faster disks. However, if you run applications that perform long queries, such as a decision support system, putting data files into faster disks.

2.3.4 Using Raw Devices

If you run DBMaster on a UNIX system, construct raw device files to store DBMaster data and journal files. Since DBMaster has a good buffer mechanism, it is much faster to read/write from a raw device than a UNIX file. For more information on how to create a raw device, refer to the operating system manual or consult your system administrator. The one disadvantage of using raw devices is that DBMaster cannot extend Tablespace on them automatically; so more planning is required when using raw device files.

But we can’t use a raw device on Windows Platform.

2.3.5 PRE-ALLOCATING AUTOEXTEND TABLESPACES

There are two types of Tablespace, Regular Tablespaces and Autoextend Tablespace. Each one has its own advantages; you should choose a reasonable one that you need. DBMaster supports autoextend Tablespaces to simplify Tablespace management. However, if you are able to estimate the required size of a Tablespace, it is better to fix the size when creating the Tablespace.

We can know that Pre-allocate Space in an Autoextend Tablespace will improve performance, as extending pages takes a lot of time. You can extend the pages of a file later by using the alter file command. Pre-allocating the size of a Tablespace can also avoid disk full errors cause by the available space is not enough when DBMaster attempts to extend a Tablespace.

2.4 Tuning Memory Allocation

DBMaster stores information temporarily in memory buffers and permanently on disk. Since it takes much less time to retrieve data from memory than disk, performance will increase if data can be obtained from the memory buffers. The size of each of DBMaster's memory structures will affect the performance of a database. However, performance becomes an issue only if there is not enough memory.

This section focuses on tuning the memory usage for a database. It includes information on how to calculate the required DCCA size, and how to monitor and allocate enough memory for the page buffers, journal buffers and system control area.

Example:

To achieve the best performance, follow the steps in the order shown:

1. Tune the operating system.
2. Tune the DCCA memory size.
3. Tune the page buffers.
4. Tune the journal buffers.
5. Tune the SCA.

DBMaster's memory requirement varies according to the applications in use; tune memory allocation after tuning application programs and SQL statements.

2.4.1 TUNING AN OPERATING SYSTEM

The operating system should be tuned to reduce memory swapping and ensure that the system runs smooth and efficiently.

Memory swapping between physical memory and the virtual memory file on disk takes a significant amount of time. It is important to have enough physical memory for running processes. Measure the status of an operating system with the operating system utilities. An extremely high page-swapping rate indicates that the amount of physical memory in a system is not large enough. If this is the case, remove any unnecessary processes or add more physical memory to the system.

2.4.2 Tuning DCCA Memory

The Database Communication and Control Area (DCCA) is a group of shared memory allocated by DBMaster servers. Every time DBMaster is started, it allocates and initializes the DCCA.

The UNIX client/server model of DBMaster allocates the DCCA from the UNIX shared memory pool. Ensure that the size of the DCCA is not larger than the maximum-shared memory size permitted by the operating system. If the requested size for the DCCA is larger than the operating system limit, refer to the operating system administration manual for information on how to increase the maximum size of shared memory.

2.4.2.1 Configuring the DCCA

The DCCA contains the process communication control blocks, concurrency control blocks, and the cache buffers for data pages, journal blocks, and catalogs. DBMaster maintains the concurrency control blocks and communication status of each DBMaster process in the DCCA. Each DBMaster process accesses the same disk data through the cache buffers in the DCCA.

Setting the appropriate parameters in **dmconfig.ini** before starting the database configures the size of each of the DCCA components.

Example 1:

A sample configuration for the DCCA in the **dmconfig.ini** file:

```
DB_NBUFS = 200
DB_NJNLB = 50
DB_SCASZ = 50
```

DB_NBufs specifies the number of data page buffers (4096 bytes per buffer), **DB_NJnlB** specifies the number of journal block buffers (4096 bytes per buffer), and **DB_ScaSz** specifies the size of the SCA in pages (4096 bytes per page). DBMaster reads these DCCA parameters only when starting a database. To adjust the parameters, terminate the database, modify the values in the **dmconfig.ini** file, and restart the database.

The total memory allocation for the DCCA is the sum of the size of **DB_NBufs**, **DB_NJnlB** and **DB_ScaSz**.

Example 2:

To calculate the total size of the DCCA:

```
size of DCCA = (200 + 50 + 50) * 4 KB = 1200 KB
```

2.4.2.2 Allocating Sufficient DCCA Physical Memory

The DCCA is the resource most frequently accessed by DBMaster processes. It is important to ensure there is enough physical memory to prevent the operating system from swapping the DCCA to disk too often or it will seriously degrade the performance of a database. The page-swapping rate can be measured by using operating system utilities.

Example

To determine the size of memory allocated for the DCCA from the system table SYSINFO:

```
dmSQL> select INFO, VALUE from SYSINFO where INFO = DCCA_SIZE or INFO = FREE_DCCA_SIZE;
      INFO            VALUE
================ ================
DCCA_SIZE           1228800
FREE_DCCA_SIZE      189024
```

DCCA_SIZE - the memory size, in bytes, of the DCCA.

FREE_DCCA_SIZE - the size, in bytes, of free memory remaining in the DCCA.

The free memory in the DCCA is reserved for use by dynamic control blocks, such as lock control blocks.

Usually a larger number of buffers are better for system performance. However, if the DCCA is too large to fit in physical memory, the system performance will degrade. Therefore, it is important to allocate enough memory for the DCCA but still fit the DCCA in physical memory.

2.4.3 TUNING PAGE BUFFER CACHE

DBMaster uses the shared memory pool for the data page buffer cache. The buffer cache allows DBMaster to speed up data access and concurrency control. DBMaster automatically configures the number of page buffers by default. Setting the **dmconfig.ini** keyword **DB_Nbufs** to zero allows DBMaster to automatically set the number of page buffers. DBMaster can dynamically adjust the number of page buffers on systems that allow DBMaster to detect physical memory usage. The number will be no less than 500 pages on Windows 95/98, or no less than 2000 pages for Windows NT/2000/ or Unix. If DBMaster cannot detect the system's physical memory usage, it will allocate the minimum amount.

Adjusting the size of the page buffers will have the greatest effect on performance. The next sections show how to monitor the buffer cache performance and calculate the buffer hit ratios.

To improve buffer cache performance:

1. Update statistics on schema objects.
2. Set NOCACHE on large tables.
3. Reorganize data in poorly clustered indexes.
4. Enlarge cache buffers.
5. Reduce the effect of checkpoints.

2.4.3.1 Monitoring Page Buffer Cache Performance

DBMaster places buffer cache access statistics in the SYSINFO system table.

You can get these values with the following SQL statements:

```
dmSQL> select NUM_PAGE_BUF from SYSINFO;

NUM_PAGE_BUF
============
    200

dmSQL> select NUM_PHYSICAL_READ, NUM_PHYSICAL_WRITE, NUM_LOGICAL_READ, NUM_LOGICAL_WRITE
from SYSINFO;

NUM_PHYSICAL_READ NUM_LOGICAL_READ NUM_PHYSICAL_WRITE NUM_LOGICAL_WRITE
================= ================ ================== =================
   13207              331595            7361              127423

1 rows selected
```

NUM_PAGE_BUF — number of pages used for data buffer cache.

NUM_PHYSICAL_READ — number of pages read from disk.

NUM_LOGICAL_READ — number of pages read from the buffer cache.

NUM_PHYSICAL_WRITE—number of pages written to disk.

NUM_LOGICAL_WRITE—number of pages written to the buffer cache.

You can calculate the page buffer read/write hit ratio with the following formulas:

$$\text{read hit ratio} = 1 - \left(\frac{\text{NUM\_PHYSIC AL\_READ}}{\text{NUM\_LOGICA L\_READ}}\right)$$

$$\text{write hit ratio} = 1 - \left(\frac{\text{NUM\_PHYSIC AL\_WRITE}}{\text{NUM\_LOGICA L\_WRITE}}\right)$$

From the example above, you can calculate the read/write hit ratio:

$$\begin{aligned}\text{read hit ratio} &= 1 - \left(\frac{13207}{331595}\right)\\ &= 0.960\\ &= 96.0\%\end{aligned}$$

$$\begin{aligned}\text{write hit ratio} &= 1 - \left(\frac{7361}{127423}\right)\\ &= 0.942\\ &= 94.2\%\end{aligned}$$

Based on the read/write the hit ratio, you can determine how to improve the buffer cache performance. If the hit ratio is too low, you can tune DBMaster with the methods described in the following subsections.

If the hit ratio is always high, for example higher than 99%, the cache is probably large enough to hold all of the most frequently used pages. In this case, you may try to reduce the cache size to reserve memory for your applications. To make sure you still maintain good performance, you should monitor the cache performance before and after making the modifications.

2.4.3.2 Statistics Values are Outdated

If the read/write hit ratio is too low, it may be that the statistics values of schema objects (tables, indexes, columns) are out of date. The wrong statistics may cause the DBMaster optimizer to use an inefficient plan for SQL statement. If users have inserted large amounts of data into the database after the last time the statistics values were updated, update the values again.

Example 1:

To update the statistics values for all schema objects:

```
dmSQL> update statistics;
```

If a database is extremely large, it will take a lot of time to update statistical values for all of the schema objects. An alternative method is to update statistics on specific schema objects that have been modified since the last update, and set the sampling rate.

Example 2:

To update specific schema objects:

```
dmSQL> update statistics tabel1, table2, user1.table3 sample = 30;
```

After successfully updating the statistical values for schema objects, monitor the performance of the page buffer cache with the method specified in *Monitoring Page Buffer Cache Performance*.

2.4.3.3 Swap Out Cache

DBMaster determines which page buffers to swap with the Least Recently Used (LRU) rule. This keeps the most frequently accessed pages in the page buffers and swaps pages that are used less frequently. However, if a large table is browsed all page buffers may be swapped out just to perform one table scan.

For example, in a database with 200 page buffers, if a table with 250 pages is browsed, DBMaster might read all 250 pages into the page buffers and discard the 200 most frequently used pages. In the worst case, DBMaster must read 200 pages from disk when accessing other data after a full table scan. However, if the table cache mode is set to NOCACHE, DBMaster will place the retrieved pages at the end of the LRU chain when a full table scan is performed. Therefore, 199 of the 200 most frequently used pages are still kept in the buffer cache.

Normally the tables with page numbers that exceed the page buffers should be set to NOCACHE. Tables that are not used frequently or with page numbers close to the number of page buffers should also be set to NOCACHE.

To determine the number of pages and cache mode for a table:

```
dmSQL> select TABLE_OWNER, TABLE_NAME, NUM_PAGE, CACHEMODE from SYSTEM.SYSTABLE where
TABLE_OWNER != 'SYSTEM';
         TABLE...OWNER          TABLE_NAME            NUM_PAGE         CACHEMODE

         ===========           ===========          ===========      ===========

             BOSS                salary                5                  T
             MIS                  asset                45                 T
             MIS                department             3                  T
             MIS                 employee              29                 T
             MIS                 worktime             450                 T
            TRADE                customer             350                 T
            TRADE                inventory            167                 T
            TRADE                 order               112                 T
            TRADE               transaction           1345                T
9 rows selected
```

NUM_PAGE—the number of pages in a table.

CACHEMODE—cache mode of full table scan, 'T' means table scan is cacheable, and 'F' means table scan is non-cacheable.

In the above **sample**, the table **TRADE.transaction** is already set to NOCACHE. The other tables still are cacheable. If there are 200 page buffers, the **MIS.worktime** and **TRADE.customer** tables should be set to NOCACHE, and the **TRADE.order** and **TRADE.inventory** tables should be set to NOCACHE if they are rarely used.

Example 2:

To set the cache mode for a table to NOCACHE:

```
dmSQL> alter table MIS.worktime set nocache on;
```

If there are no valid indexes for a table, or the predicate in a query references non-indexed columns, DBMaster may also perform a full table scan. To prevent this, try to write SQL statements as efficiently as possible, and make use of indexed columns when possible.

2.4.3.4 Poor Clustering of Records

When fetching many records that must be ordered by an index key, or when the predicate references an indexed column, index clustering becomes an important factor that affects the buffer cache performance.

For example, if you execute an SQL statement to select all columns from the customer table and sort it on the primary key **custid** as shown below:

```
dmSQL> select * from customer order by custid;
```

Suppose there are 3500 records in table **customer** distributed over 350 pages, and there are 200 page buffers in your system. If the records are clustered by **custid** and the clustering is very good (arranged sequentially on all pages), DBMaster only needs to read 350 pages from disk. But if the clustering is bad (no sequential records on the same page), DBMaster may have to read 3500 pages from disk in the worst case (every record needs a disk read)! To determine the state of your index clustering, you must update statistics on the table first. Suppose you have built an index called **custid_index** on the **custid** column of table **customer**. Then you can execute the following statements:

```
dmSQL> select CLSTR_COUNT from SYSTEM.SYSINDEX
                where TABLE_OWNER = 'TRADE'
                and TABLE_NAME = 'customer'
                and INDEX_NAME = 'custid_index';

CLSTR_COUNT
===========
    385

1 rows selected
```

CLSTR_COUNT—cluster count, the number of data pages that will be fetched by a fully indexed scan with few buffers.

In the above example, DBMaster at most performs 385 page reads from disk when you scan the full **customer** table and order the results by the **custid** column.

```
dmSQL> select NUM_PAGE,NUM_ROW from SYSTEM.SYSTABLE
                  where TABLE_OWNER = 'TRADE'
                  and TABLE_NAME = 'customer';

NUM_PAGE   NUM_ROW
=========== ===========
    350      4375

1 rows selected
```

NUM_PAGE—the number of pages allocated by a table.

NUM_ROW—the number of records in a table.

With CLSTR_COUNT, NUM_PAGE and NUM_ROW, you can estimate the clustering factor with the following formula:

$$\text{clustering factor} = \frac{(\text{CLSTR\_COUNT} - \text{NUM\_PAGE})}{\text{NUM\_ROW}}$$

In the above example, you can see the clustering factor is 1.7%.

$$\begin{aligned}\text{clustering factor} &= \frac{(385-350)}{9375} \\ &= 0.0017 \\ &= 1.7\%\end{aligned}$$

The clustering factor will be between 0 and 100%. In cases where CLSTR_COUNT is only a little less than NUM_PAGE, you can treat it as zero. If the clustering factor is zero, it means your data is fully clustered on this index. If the clustering factor is too high, for example larger than 20% (what determines a high rate depends on the table size, average record size, etc.), the index has bad clustering. When DBMaster finds an index has bad clustering, the DBMaster optimizer may use a full table scan when you execute an SQL statement even if you think it should be processed by an index scan.

When you find the clustering of a frequently used index is bad, you perform the following procedure to improve index clustering:

- unload all data from the table (order by the index).
- rearrange the unloaded data by order.
- drop indexes on the table.
- delete all data in the table.
- reload the data into the table.
- recreate indexes on the table.

After data reloading, the index should be fully clustered. You should note however, a table can only be clustered on one index. If one table has many indexes, you should maintain index clustering on the most important **index**. Usually, the most important index is the primary key. Since unloading and reloading data takes a great deal of time and storage, you should tune index clustering only on the tables that are very large and frequently browsed.

2.4.3.5 Low Data Page Buffers

If allocated data page buffers are not enough for your database access, you should add page buffers to the DCCA. The following steps show how to modify the number of page buffers:

- terminate the database server.
- reset DB_NBUFS in dmconfig.ini to a larger value.
- restart the database.

After successfully enlarging the data buffers, you should run your database for a period of time and then monitor the buffer cache performance again. If the buffer hit ratio has gone up, adding buffer pages has resulted in a performance improvement. If not, you must again add more pages to the buffer cache or check for other reasons your system performance may be reduced.

2.4.3.6 Checkpoints Occurring Too Often

If the write hit ratio is much lower than the read hit ratio, the cause may be that checkpoints are being processed too often. To determine how many checkpoints have been processed, you can use the following SQL statement:

```
dmSQL> select NUM_CHECKPOINT from SYSINFO;

 NUM_CHECKPOINT
 ==============
        26

1 rows selected
```

When a checkpoint is processed, DBMaster will write all dirty page buffers to disk. Since checkpoints require a lot of CPU time, you can manually perform a checkpoint during periods when the CPU is idle. For example, if you are using UNIX you can set a **cron** job to perform a checkpoint every night. Another advantage of performing checkpoints periodically is to reduce the recovery time taken by DBMaster to start a database after a system crash.

Except when you make a checkpoint manually, DBMaster makes checkpoints automatically when DBMaster runs out of free journal space in NON-BACKUP mode or when an incremental backup is performed in BACKUP mode. To increase the time interval between automatic checkpoints, you can enlarge your journal size.

2.4.3.7 Re-monitor the Cache Buffer Performance

After tuning your system with the above methods, you should re-monitor the cache buffer performance using the following procedure:

- run the database for a period of time to ensure the information in the database is in a stable state.
- reset the statistics values in the SYSINFO system table with the following SQL statement.

```
dmSQL> set SYSINFO clear;
```

- run the database for a period time.
- get the read/write counter from the SYSINFO table and check the hit ratio.

2.4.4 Tuning Journal Buffers

The journal buffers store the most recently used journal blocks. With enough journal buffers, the time required to write journal blocks to disk when updating data and reading journal blocks from disk when rolling back transactions is reduced.

If you seldom run a long transaction that modifies (inserts, deletes, updates) many records, you may skip this section. Otherwise, should determine whether there are sufficient journal buffers for the system. The optimum number of journal buffers is the sum of journal blocks needed by the longest running transactions at the same time.

To estimate the number of journal buffers, perform the following:

1. Make sure there is only one active user in the database.
2. Clear the counters in the SYSINFO table with the following command:

```
dmSQL> set SYSINFO clear;
```

3. Run the transaction that will update the most records.
4. Run the following SQL statement to determine the number of used journal blocks:

```
dmSQL> select INFO, VALUE from SYSINFO where INFO = 'NUM_JNL_BLK_WRITE';
        INFO                              VALUE
       NUM_JNL_BLK_WRITE                   626
       1 rows selected
```

NOTE: NUM_JNL_BLK_WRITE—the blocks used in this transaction. The journal block size used in this example is 512 bytes. In the above example, you need approximately 41 journal buffer pages (1 page=4KB).

Another measurement that can be used to determine the journal buffer utilization is the journal buffer flush rate. The journal buffer flush rate is the percentage of journal buffers flushed to disk when DBMaster writes to the journal. If the journal buffer flush rate is too high (for example, more than 50%), increase the number of journal buffers.

2.4.5 Tuning the System Control Area (SCA)

Cache buffers and some control blocks, such as session and transaction information, have a fixed size, and are pre-allocated from the DCCA when a database is started. However, some concurrency control blocks are allocated dynamically from the DCCA while the database is running, their size is specified by DB_ScaSz.

If a database application gets the error message “database request shared memory exceeds database startup setting”, it means that DBMaster cannot dynamically allocate memory from the SCA area. Usually, this error is due to a long transaction using too many locks. If this situation happens often, solve it with the methods illustrated below.

2.4.5.1 Avoid Long Transactions

A long transaction will occupy many lock control blocks and journal blocks. If there is a long transaction in progress when the above error occurs, analyze whether the transaction can be divided into multiple small transactions.

2.4.5.2 Avoid Excessive Locks on Large Tables

Selecting many records from a large table using an index scan requires many lock resources. To decrease the amount of lock resources used by the transaction, escalate the lock mode before performing the table scan.

For example, if the table's default lock mode is row, escalate the default lock mode to page or table. Although this will reduce the resources used, it will also sacrifice concurrency to some degree.

2.4.5.3 Increase the SCA Size

If both of the above conditions have not occurred, increase the size of the SCA. Reset the value of **DB_ScaSz** in **dmconfig.ini** to a larger value and then restart the database.

2.4.6 Tuning the Catalog Cache

DBMaster stores the catalog cache in the SCA. If schema objects are seldom modified, turn on the data dictionary turbo mode by setting **DB_Turbo**=1 in the **dmconfig.ini** file. When turbo mode is on, DBMaster will extend the lifetime of the catalog cache. This can improve the performance of on-line transaction processing (OLTP) programs.

2.5 Tuning Concurrent Processing

Resource contention occurs in a multi-user database system when more than one process tries to access the same database resources simultaneously. This can also lead to a situation known as a deadlock, which occurs when two or more processes wait for each other. Resource contention causes processes to wait for access to a database resource, reducing system performance.

DBMaster provides the following methods to detect and reduce resource contention:

- Reducing lock contention.
- Limiting the number of processes.

2.5.1 Reducing Lock Contention

When accessing data in a database, DBMaster processes will lock the target objects (records, pages, tables) automatically. When two processes want to lock the same object, one must wait. If more than two processes wait for the other processes to release the lock, a deadlock occurs. When a deadlock occurs, DBMaster will sacrifice the last transaction that helped cause the deadlock by rolling it back. Deadlock reduces system performance. Monitor lock statistics to avoid a deadlock in DBMaster.

To view deadlock statistics:

```
dmSQL> select INFO, VALUE from SYSINFO where INFO = 'NUM_LOCK_REQUEST'
    2>  or INFO = 'NUM_DEADLOCK'
    3>  or INFO = 'NUM_STARTED_TRANX';
            INFO                          VALUE
       NUM_STARTED_TRANX                   33
       NUM_LOCK_REQUEST                    73
       NUM_DEADLOCK                        0
  3 rows selected
```

NUM_LOCK_REQUEST—the number of times a lock was requested.

NUM_DEADLOCK—the number of times deadlock occurred.

NUM_STARTED_TRANX—the number of transactions that have been issued.

In the above example, on average one transaction is in deadlock per 51 (9287/181) transactions and one transaction requests approximately 83 (772967/9287) locks.

If the deadlock frequency is high, examine the schema design, SQL statements, and applications. Setting the table default lock mode lower, such as ROW lock, could reduce the lock contention, but it will require more lock resources.

Another method is to use the browse mode to read a table on a long query if the data does not need to remain consistent after the point in time that it was retrieved. This is useful when viewing the data or performing calculations using the data while not performing any updates. It provides a snapshot of the requested data at a particular point in time, but with the benefit of increased concurrency and fewer lock resources consumed, because locks are freed as soon as the data is read.

2.5.2 Limiting the Number of Processes

DBMaster allows up to 1200 simultaneous session connections to a server. If server resources, (such as memory, CPU power) are not sufficient, limit the maximum number of connections to avoid resource contention. The configuration parameter **DB_MaxCo** affects the maximum number of connections in the database.

When a database is initially created, the journal file is formatted for a specific number of connections. The journal file needs to be able to preserve a transaction information array for each connection. The number of connections available according to the journal file is also known as the *hard connection number*. This value is determined by the value of **DB_MaxCo** when the database is created. The *hard connection number* has a minimum value of 240, a maximum value of 1200, and must be a multiple of 40. If **DB_MaxCo** is set to a value that is not a multiple of 40, then the *hard connection number* is rounded up to a value that is a multiple of 40. The hard connection number is a limitation of the journal file, therefore, to change it the database must be started in new journal mode.

The hard connection number does not directly affect the size of the DCCA. This is determined by a value known as the *soft connection number*. The soft connection number is exactly the value of **DB_MaxCo**. The soft connection number determines the number of connections that the DCCA will support, and consequently the memory usage of the DCCA. The soft connection may be any value less than or equal to the hard connection value. To change the soft connection number, restart the database normally after changing **DB_MaxCo**.

Example 1:

In the following configuration file, the hard connection number for **DB1** is 240. For database **DB2**, it is 1120.

```
[DB1]
DB_MaxCo = 50          ;; the hard connection number = 240
                       ;; the soft connection number = 50
[DB2]
DB_MaxCo = 1100          ;; the hard connection number is 1120
                         ;; the soft connection number = 1100
```

Example 2:

After starting the database successfully, the new hard connection number for **DB1** becomes 280.

```
[DB1]
DB_SMode = 2            ;; startup with new journal file
DB_MaxCo = 280          ;; the new hard connection number = 280
```

Example 3:

Assuming **DB2** has already been created as in example 1, the following entry in the **dmconfig.ini** file will result in a hard connection number of 1120 and a soft connection number of 20.

```
[DB2]
DB_SMode = 1              ;; normal start
DB_MaxCo = 20          ;; the new soft connection number = 20
```

3. Database Model Design

Users should use advanced functions which provided by database as more as possible, rather than implement lots of data disposal process function in application program. You can take some data disposal process out of application program, then use the functions provided by database to carry out purpose.

3.1 Tablespace

Estimation by the capacity of database, you must assign enough disk capacity in advance, in case of high frequency increasing capacity automatically when the capacity of database is not enough.

3.2 Table

According to the data capacity of table, you'd better put the bigger table independently into a single Tablespace, rather than put bigger tables and small ones into the same Tablespace.

3.3 Index

When creating index, you should select the column which most frequently used rather than the least frequently used one, that will improve performance. The reason is that index will improve efficiency when executing the operation scan whereas it will lower the whole efficiency when executing the operation delete and update to maintain index page.

3.4 Constraint

About the constrained disposal process when input data and maintain, please use such functions provided by database as Primary key, Foreign key, default, check etc. That not only enhance the develop efficiency, but also enhance the performance of whole system.

3.5 Trigger

About some cascade processing such as maintaining the consistency of the data or others, please use the Trigger to realize. That can not only improve the speed of software development, but also improve the Performance of the system.

3.6 Stored command

About the SQL command that use frequently in the system, please use Stored Command to replace. That should improve the Performance.

3.7 Stored procedure

About the logical processing which is complex, please use the Stored Procedure to realize.That should improve the Performance of the system.

4. Setting DBMaker Daemon

4.1 I/O and Checkpoint

4.1.1 I/O DAEMON

DBMaster has an I/O daemon to periodically write dirty pages from the least recently used page buffers to disk. This helps reduce the overhead incurred when swapping data pages into the page buffers, and increases performance. One configuration parameter in the **dmconfig.ini** file is used to control the I/O daemon.

DB_IOSvr—enables and disables the I/O daemon. Setting this keyword to a value of 1 enables the I/O daemon, and setting it to a value of 0 disables the I/O daemon.

Example

A typical excerpt from the dmconfig.ini file:

```
[MYDB]
...
DB_IOSVR = 1
```

MYDB database has 400 (DB_NBufs) page buffers in DCCA. Every 10 minutes, the I/O daemon will perform the following steps:

- Scan the least recently used page buffers.
- Collect the dirty pages during scan processing.
- Write these collected dirty pages to disk.

4.1.2 CHECKPOINT DAEMON

DBMaster has a checkpoint daemon (based on the I/O daemon) that periodically takes a checkpoint. This helps reduce the time spent waiting for a checkpoint that occurs during a command, when a journal is full, or when starting or shutting down a database. The checkpoint daemon is actually a sub-function of the I/O daemon, which can perform I/O alone, checkpoints alone, or both together. There is one keyword for use in the **dmconfig.ini** file, which is used to control the checkpoint daemon.

DB_ChTim—specifies the first time a checkpoint daemon should run. The format for this keyword is yyyy/mm/dd hh:mm:ss.

To turn on the **checkpoint** daemon, turn on the I/O daemon using the **DB_IOSrv** keyword. If the I/O daemon is activated without setting **DB_ChTim**, it will automatically take a checkpoint every hour by default after the database starts successfully.

Example:

To start checkpoint daemon and stop the I/O daemon in the **dmconfig.ini**:

```
[MYDB]
...
DB_IOSVR = 1 ; may enable I/O or checkpoint daemon
DB_CHTIM = 2000/1/1 00:00:00 ; the first time the daemon should run
```

In fact, the I/O and checkpoint daemon will expend some I/O resources. After starting the database server, any error messages generated by the I/O and checkpoint daemon are written to the file **ERROR.LOG**.

DB_ChItv (d-hh:mm:ss) — the time interval of Checkpoint daemon, the default value is 1 hour.

4.2 Update Statistics

User can adjust the execution time of update statistics as need. Statistics are automatically updated daily at 3:00 AM by default. If DBMaker is handling other busy transactions at same time, Database will halt the behavior of update statistics, then execute the update statistics again till 03:00 AM tomorrow. If some other busy transactions will hold more resources daily at 03:00 AM. Then the update statistics behavior will not be executed in the course of nature. In this way, statistics data will be out of date and can't reflect the updated status of Database, then can't achieve the purpose of according to 'cost' to optimize SQL. Therefore, user must adjust the other task execution time properly to execute the update statistics.

DB_StSvr

This keyword is used to activate the auto update statistics server. A value of 1 indicates that the server is started. A value of 0 indicates that the server is not running. If the auto update statistics server is activated, it will recalculate database statistics daily at 3:00 AM.

4.3 Backup Server

User can adjust the execution time of backup as need. Adjust principles is to arrange the backup process to the time of lower load, not to dispute server resources with other importance transactions. Thus avoid the opportunity of deadlock reduces system performance.

Backup Schedule

DB_BKSVR—This keyword specifies whether or not a backup server will be started when a database is started. Setting this value to 1 will start a backup server for that database. The default value is 0.

DB_BKFUL—This keyword specifies the percentage that all journal files must be filled to before the backup server is triggered to do an incremental backup. Setting this value to 0 will trigger the backup server whenever a journal file is full. Setting this value between 50-100 will trigger the backup server whenever the total space used in all of journal files exceeds the specified percentage. For example, if there are two journal files of 500 journal blocks each and DB_BKFUL is set to 80, then after every 500x2x0.8=800 blocks are used, the backup server will automatically do an incremental backup. The default value is 0.

DB_BKITV— This keyword specifies the backup time interval. Please refer to **DB_BkTim** described later, no backup schedule if **DB_BkItv** is not set.

DB_BKTIM— This keyword along with **DB_BkItv** specifies the schedule of the backup server. **DB_BkTim** specifies the first time a backup server will perform an incremental backup. Incremental backup will then be performed after every time interval specified in **DB_BkItv**, no backup schedule if **DB_BkTim** is not set.

4.4 Replication Server

User can adjust the execution time of replication as need, adjust principles is to arrange the replication to the time of lower load, not to dispute server resources with other importance transactions, and achieve replication actions using ATR as soon as possible, thus can enhance system performance efficiency.

RP_Btime : Starting time of replication

RP_Iterv : Schedule for the database replication

4.5 Auto-Commit mode

When auto-commit mode is on, DBMaster will automatically issue a commit transaction after each SQL command is successfully executed.So it will spend too much time to I/O. The performance will be poor.

We can understand this factor as following example:

For example, when we insert 10000 ordinary data into table,

(1) Insert into DBMaster **and set autocommit on**

```
Private Sub Command1_Click()
    Dim objConn As New ADODB.Connection
    Dim strSQL As String
    Dim time1 As String
    Dim time2 As String
    time1 = Now
    objConn.Open ("DSN=test;UID=sysadm;PWD=;")
    For i = 1 To 10000
        DoEvents
        strSQL = "insert into test_table values(1)"
        objConn.Execute strSQL
    Next
    objConn.Close
    time2 = Now
    MsgBox time1 & "--->" & time2
    Set objConn = Nothing
End Sub
```

(2) Insert into DBMaster **and set autocommit off**

```
Private Sub Command2_Click ()
    Dim objConn As New ADODB.Connection
    Dim strSQL As String
    Dim time1 As String
    Dim time2 As String
    time1 = Now
    objConn.Open ("DSN=aa;UID=sysadm; PWD=;")
    objConn.BeginTrans
    For i = 1 To 10000
        DoEvents
        strSQL = "insert into test_table values(1)"
        objConn.Execute strSQL
    Next
    objConn.CommitTrans
    objConn.Close
    time2 = Now
    MsgBox time1 & "--->" & time2
    Set objConn = Nothing
End Sub
```

Following this example, we can see the result in this chart:

	DBMaster4.1 (second)	
Default	Set autocommit on	11
Tune	Set autocommit off	6

Auto-commit is the most important factor when you use DBMaster first time. To understand auto-commit better will make great improve on performance of DBMaster.

5. Query Optimization

In this chapter, we will introduce the query optimizer of DBMaster. The query optimizer will make a query of SQL command much faster and efficient by means of choosing the best execution method internally. The contents in this chapter involve the following topics:

- What is query optimization? Why do we need it? When you understand the goal of query optimization, you will find the role it plays in a SQL query.
- What is Query Execution Plan (QEP)? How to read a QEP? When you know the QEP, you will learn how DBMaster executes a SQL query command as well.
- How does the query optimizer operate? When you understand the way the query optimizer searches for a QEP, you can help it to find a more efficient QEP by rewriting an equivalent SQL query instead of the original one.
- What is the cost function? When you know how much time it takes for an operation in QEP, you will learn how the query optimizer chooses a proper operation. Moreover, you can use some commands provided by DBMaster to help the query optimizer find a better operation.
- What are the statistics values? Where has these values been used? When you understand the usage of statistics values of query optimization, you will see the reason why query optimizer chooses such a execution plan.
- How to accelerate the execution speed of a query? When you know how to write an efficient query, you can enhance the execution efficiency by rewriting the query command by yourself.

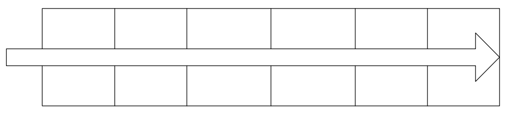
5.1 Indexes

Build indexes on columns in a table make DBMaster finds all required data conveniently. If you built the index reasonable, it will improve the performance.

If you haven't built any index on columns in a table, it will use table scan to acquire rows from a table row by row.

For instance, if a user wants to find all rows in a table to match the condition **age > 50**, it will receive each row from each data page, and then compare each row with the condition to retrieve the desired one.

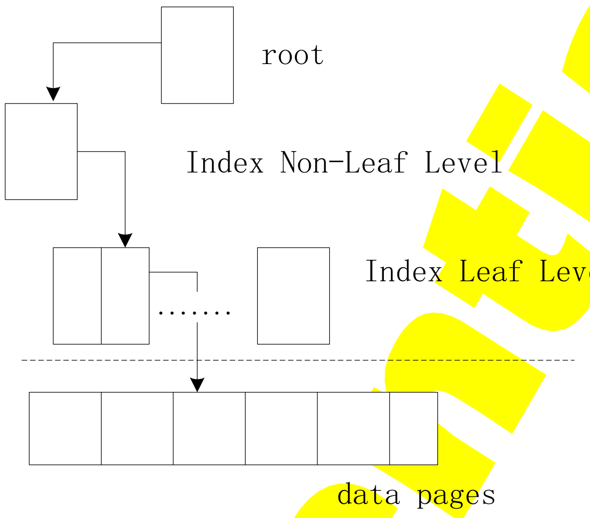
Like the figure shows:



Data pages

But if you build indexes on columns in a table, it will use reference index to scan all required data and get them. The method used in DBMaster is a B-tree, and the precondition to use an index is to build an index on the column with predicate.

The last instance, if the user build index on age columns, like that:



5.2 Statistics Values are Outdated

The statistics value is very important if a table is read frequently. It represents the amount and distribution of data for a table. They provide the information for the cost functions to find the best access plan.

The wrong statistics may cause DBMaster optimizer to use an inefficient plan for SQL statement. If other users insert large amounts of data into a database after the last time you have updated the statistics values, update the values again.

5.3 What is query optimization?

For the commands of Data Manipulation Language (DML) such like SELECT, INSERT, DELETE, and UPDATE, it is a very important stage of query optimization. The reason is that DBMaster may have several execution methods for one SQL query. The goal of query optimization of DBMaster, is to find the most efficient and best execution plan among all of those ones. The main job of the query optimizer is to decide each operation, and the order it operates. To do this, we need to find the most efficient operation from the following basic ones:

- Read the data from a table -- it can be read by sequential scan or by index scan.
- Join tables -- tables can be joined by nested loop join or by sort merge join.
- Sort -- when does a sort be needed? Before some operation or after it? Or can we avoid sorting by some alternative way?

Besides to estimate the number of left rows for a table being operated, the query optimizer needs to consider more factors, in order to find the best choices among so many execution methods.

There is one more point we want to mention here, that is, it is usually the case that the database user gets familiar with the data characteristics rather than query optimizer. In some condition, a database user can find a more efficient way to execute a query than query optimizer.

5.4 Query Execution Plan (QEP)

Query optimizer of DBMaster will estimate all possible execution plans; for each plan, compute the number of rows, how many disk page I/O being needed, and CPU time it takes for a single table. From the above factors mentioned, find a plan with the lowest cost.

On the other hand, query optimizer must decide a query execution plan. A query execution plan is composed of several operation units, and it will choose the best one among all the possible combinations of these operations.

When DBMaster seeks for a query execution plan, it will consider some major operation units:

- Table scan -- or called sequential scan, which means receiving each row from data pages of database by sequential order.
- Index scan -- the order to retrieve data is referenced by the address of data page that is pointed by the index page.
- Nested join -- compare two or more tables row by row, to achieve the goal of merging these tables.
- Merge join -- sort two tables respectively, then compare these two ordered tables row by row, to achieve the goal of merging these tables.
- Sort -- executes sort.
- Temporary table -- in the process of query execution, establish the temporary table.

Let's see the following example

```
dmSQL> SELECT * FROM t1, t2 WHERE t2.c2=3 AND t1.c1=t2.c1 ORDER BY t1.c2;
```

Query Execution Plan 1

```
sort t1.c2
     merge join t1.c1=t2.c1
        index scan t1 on idx1(c1)
        sort t2.c1
           table scan t2, filter t2.c2=3
```

Query Execution Plan 2

```
nested join
     index scan t1 on idx2(c2)
     table scan t2, filter t2.c2=3 and t1.c1=t2.c1
```

5.5 How Does the Optimizer Operate

When the optimizer of DBMaster performs the optimization process to a query, it will follow some rules of the following input information:

- Analyze the query, then cut the where predicate into several factors.
- Search all possible execution sequence and join sequence.
- Decide whether using the nested join or sort merge join.

- Decide whether using table scan or index scan.
- Decide how to sort.

5.5.1 Input of Optimizer

The critical factor whether the optimizer will be successful or not is the precision to estimation. However, there is only finite information for the optimizer. Generally speaking, the estimate time needed by the optimizer occupies only a small part compared with the real execution time. The system catalog tables provide all information that used by the optimizer.

All the information needed by the optimizer come from system catalog tables. To make sure that these information are useful, not out of date, users must use command UPDATE STATISTICS (Please refer to section 5, Statistics). Here we list all the used data in system catalog tables:

- Number of rows in a table
- Number of data pages used by a table
- Average bytes of a row for a table
- Average bytes that a column uses
- The distinct value of each index column
- The second maximum and minimum value for each column, the reason that we do not choose the maximum and minimum value is to avoid that some special large and small value will affect the precision
- Number of index scan pages occupied by the B-tree index
- Number of level (height) of the B-tree index
- Number of leaf pages of the B-tree index
- Cluster count of the B-tree index

The premise that the optimizer uses the information is that we assume that the distribution of data value is uniform. If the distribution of data is skew, and not uniform, the optimizer will choose a worse plan.

5.5.2 Factors

The first job of the optimizer is to examine all expressions in the where predicate. If we decompose these expressions into several small expressions independent with each other, then we call these small expressions to be factors.

Example 1

```
dmSQL> select * from t1, t2 where t1.c1=t2.c1 and t1.c2=3;
```

According to the **where** predicate, the optimizer will decompose the predicate into two factors: "**t1.c1=t2.c1" and "t1.c2=3**".

Example 2

```
dmSQL> select * from t1, t2 where t1.c1=t2.c1 or t1.c1=3;
```

According to the **where** predicate, there is only one factor: "**t1.c1=t2.c1 or t1.c2=3**".

Now let's see the **example 3**,

```
dmSQL> select * from t1, t2 where t1.c1=t2.c1 and (t1.c2=3 or t2.c2=5);
```

According to the **where** predicate, there are two factors: "**t1.c1=t2.c1",** and **"t1.c2=3 or t2.c2=5**".

Example 4

```
dmSQL> select * from t1, t2 where t1.c1=t2.c1 and t1.c2=3 or t2.c2=5;
```

According to the **where** predicate, there is only one factor: "t**1.c1=t2.c1 and t1.c2=3 or t2.c2=5**".

From the above example, we can find easily that when the expression contains binary operation "and", then it can be divided into different factors. But when it contains binary operation "or", the decomposition is not allowed.

Besides to find the factors, the optimizer needs to estimate the selectivity of each factor. The selectivity is the ratio of data filtered by each factor, its value is between 0 and 1. For instance, there are 100 rows in table t1, if there are 5 rows for query

```
dmSQL> select * from t1 where t1.c1=3;
```

then the selectivity of factor "**t1.c1=3**" is 5/100, that is 0.05.

If there are more than one factor in an expression, then the selectivity of this expression is the product of these factors because they are independent with each other.

5.5.3 Join Sequence

The join sequence is the access order of the original table to be merged. Different join sequence will produce different execution sequence and different execution time. But no matter how we execute, we will always the correct result after execution.

Example 1

```
dmSQL> select * from t1, t2 where t1.c1=t2.c1;
```

Query Execution Plan 1

```
nested join
     table scan t1
     table scan t2, filter t1.c1=t2.c1
```

Query Execution Plan 2

```
nested join
     table scan t2
     index scan t1 on t1(c1), filter t1.c1=t2.c1
```

Example 2

```
dmSQL> select * from t1, t2, t3 where t1.c1=t2.c1 and t2.c1=t3.c1;
```

From this query, we can see that there will be 3! (=6) join sequences. All these possible sequences are:

```
(t1, t2), t3
(t1, t3), t2
(t2, t1), t3
(t2, t3), t1
(t3, t1), t2
(t3, t2), t1
```

DBMaster will search all these join sequences, then compute their cost, and choose the best one.

5.5.4 Nested Join and Merge Join

There are two join method supported by DBMaster: they are nested join and merge join.

- Nested join uses nested loop over two layers to accomplish the join purpose. By the analysis of algorithm, its time complexity is n^2.
- Merge join will sort two tables respectively in advance, then merge of these two tables with the sorted order row by row. The time complexity of sort is n x log(n). For the data that has already sorted, the time complexity to perform join is n. Sort merge join can only be used for equal join.

From the view of time complexity, merge join is better than nested join. But there are still exceptions, for example, the difference of the number of rows of two tables are very large.

No matter what, the optimizer will decide the way to perform join by cost functions and statistics values.

5.5.5 Table Scan and Index scan

Table scan means to achieve all rows from a table sequentially row by row. For instance, if a user wants to find all rows in a table that matches the condition **age > 50**, then it will receive each row from each data page, then compare each row with the matched condition to get the desired one.

Another scan type is called index scan, which means to build the index on some columns of a table, then find all needed data by the reference of index. The index method used by DBMaster is B-tree, and the precondition to use an index scan is to build an index on the column we want to use as the predicate. Again, if a user wants to find all data from a table that matches the condition **age > 50**, and there exists an index built on the column **age.** At this time, DBMaster will filter the data by the index, then read desired data from the data page reference by an index, if it uses index scan.

It is determined by cost functions to use table scan or index scan.

5.5.6 Sort

Another important question of query optimizer is to determine how to sort: before join or after it, or try to avoid sorting.

We use the following example to illustrate it.

```
DmSQL> select * from t1, t2 where t1.c1=t2.c1 order by t1.c2;
```

Query Execution Plan 1

```
sort t1.c2
      merge join t1.c1=t2.c1
           index scan t1 on idx1(c1)
           sort t2.c1
               table scan t2
```

Query Execution Plan 2

```
 nested join
      index scan t1 on idx2(c2)
      table scan t2, filter t1.c1=t2.c1
```

In QEP1, the optimizer performs sorting after merging, and in QEP2, it will perform sorting before merging. It is determined by cost functions that which is better.

5.6 Time Cost of a Query

For a database, there are two major parts to perform a query: time to read data from disk, and time to compare the column values. The former takes more time than the latter.

5.6.1 Operation Cost in Memory (CPU Cost)

The database server must process data in memory. It has to read a row into memory, then use filter expression to test. On the other hand, it needs to load data from two tables respectively into memory first, then test their join condition. Besides, the database server also has to collect data of the selected columns from each row.

Most motions in the memory process runs fast. According to the differences of CPUs, database server can handle hundreds, even thousands of comparison for a second. Thus it usually takes a small part of time during the whole query execution for the process in memory.

However, it still takes more time for two types of operations. One of them is sorting, and another is using wild cards in keywords such as “**like”** and “**match”**.

5.6.2 Procession Cost in Disk Access (I/O cost)

It takes much more time to read a row from the disk than to check a row in memory, so one of the main purposes of the optimizer is to reduce the reading amount from the disk.

The basic unit to process disk storage of database server is called page. A page is blocks clustered in a disk spaces, and the size of a page is related with the database server. The size is 4096 bytes for DBMaster. The capacity to contain how many rows in a page is related with the size of a row. There are 10 to 100 rows in a data page in a common case. Besides the entity of an index page contains a key value and a four-byte pointer, therefore there are usually 100 to 1000 entries in an index page.

The database server needs a memory space to store the copies of disk pages read from the disk for processing. Because of the limitation of memory space, some of these pages might be reread in some condition. We call such a memory space to be page buffer. If the needed page happens to be in the page buffer, then the server will not read the row from the disk anymore, and it will rise the performance for this situation. The size of a disk page and the number of page buffer is decided by Database server and the operating system.

The real cost to read a page is variable and hard to be estimated precisely. It is the combination of the following factors:

- Buffers – it is possible that the target page to be in page buffer. The access cost can be almost omitted in this condition.
- Contention – If there are more than one application programs to use the hardware devices such as disk, the request of database server will be delayed at this time.
- Seek time – this is the most time-consuming motion in a disk. It means the elapsed time to move the read/write head to the location of the desired data. It is affected by the speed of disk, and the initial position of disk read/write head. The variation is also large of seek time.
- Latency time – or called rotation delay time. It is related with the speed of the disk, and location of read/write head.

5.6.3 Cost of Table Scan

It is the spent time to scan all data from a table. No matter whether there are predicates in the query or not, it needs to compare all data in the pages. Thus the cost of table scan equals to the number of data pages.

5.6.4 Cost of Index Scan

Index scan means to read data through B-tree index pages. There are two kinds of index scan: one will read data page referenced by B-tree, and the other will read data directly from index leaf, not data page. We call the latter leaf scan.

For example: suppose there is a table t1 with two columns c1 and c2. There is an index built on c1. When we want to execute the following command

```
dmSQL> select * from t1 where c1 > 0;
```

We will use index scan, but read data from the data from data pages referenced by index page.

Another command

```
dmSQL> select c1 from t1 where c1 > 0;
```

We can use leaf scan, and there is no need to read data from the data pages because the leaf pages contain all the desired data.

When we read all data, the cost of index scan is:

cost = B tree level I/O + no. of leaf page I/O + cluster count

When we read all data but only need leaf scan, the cost of index scan is:

cost = B tree level I/O + no. of leaf page I/O

When we read a row, the cost of index scan is:

cost = B tree level I/O + one leaf page I/O + one data page I/O

When we read a row but only need leaf scan, the cost of index scan is:

cost = B tree level I/O + one leaf page I/O

When we read partial data, the cost of index scan is:

cost = B tree level I/O + (no. of leaf page x S) + (cluster count x S)

where S means selectivity.

5.6.5 Cost of Sort

Except reading data from disk into memory, it spends more time on the computing in memory for sorting. The computing cost is proportional to **c x w x n x log$_2$(n)**, where c is the number of columns being sorted, w is the bytes of sorted key, and n is the number of rows being sorted.

5.6.6 Cost of Nested Join

It will need more than two loops to access data pages for nested join. To the nested join, the outer table is different from the inner one. Generally, the cost of nested join is

outer table I/O + inner table I/O x number of rows in outer table

5.6.7 COST OF MERGE JOIN

It is necessary to sort tables before performing merge join. Suppose we have already sorted two tables needed to be merged on the merge keys, then the cost of merge join is the sum of I/O of these two tables. If we do not perform sorting on merge keys, then we still need to add the cost of sorting.

5.7 Statistics

Statistics represents the amount and distribution of data of a table. It provides the information to cost functions to find the best access plan. But the statistics will be out of date as the data in a table being inserted, deleted, or updated. We can execute command "update statistics" to update statistics values to find real statistics at this time, to enhance the efficiency of a query.

5.7.1 TYPES OF STATISTICS

DBMaster will collect the following statistics.

5.7.1.1 For a table

- nPg – number of data pages
- nRow – number of data rows
- avLen – average bytes of a row

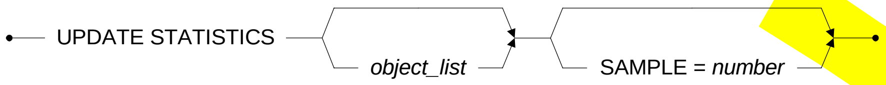
5.7.1.2 For a column

- distVal – number of distinct values
- avLen – average bytes of each column
- loVal – the second minimum value of a column
- hiVal – the second maximum value of a column

5.7.1.3 For an index

- nPg – number of index pages
- nLevel – number of levels of an index tree
- nLeaf – number of leaves of an index tree
- distKey – number of distinct keys
- distC1 – number of distinct keys of the first index column
- distC2 – number of distinct keys of the first two index columns
- distC3 – number of distinct keys of the first three index columns
- nPgKey – number of index pages for each key
- cCount – number of cluster count, which means the number of data pages access through an index

5.7.2 Syntax of UPDATE STATISTICS



The *object_list* Clause

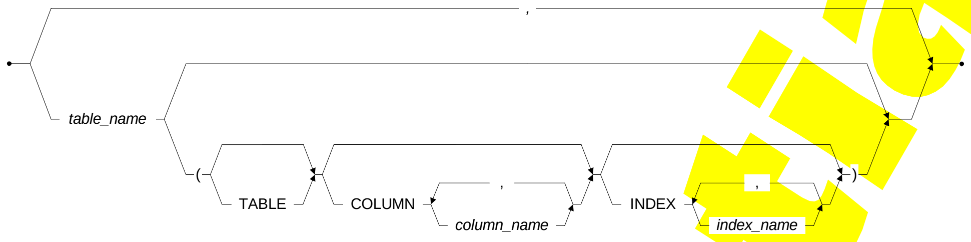


Figure: Syntax for the UPDATE STATISTICS Statement

Where

- owner represents the table owner
- table represents the name of a table
- col1, col2 represent the name of columns
- idx1, idx2 represent the name if indices
- SAMPLE means the sampling rate, it is an integer between 1 and 100.

For instance,

```
dmSQL> UPDATE STATISTICS;
```

This command will update statistics values of all tables, including all columns, indices, and system tables. The default value of sampling rate is 100.

```
dmSQL> UPDATE STATISTICS sample=30;
```

This command will update statistics values of all tables, including all columns, indices, and system tables. The default value of sampling rate is 30.

```
dmSQL> UPDATE STATISTICS jeff.emp;
```

This command will update the statistics values of table jeff.emp.

```
dmSQL> UPDATE STATISTICS jeff.emp (TABLE COLUMN(name, age) INDEX(idx1));
```

This command will only update statistics values of columns name, age and index idx1, of table jeff.emp.

```
dmSQL> update statistics jeff.emp, jeff.dept;
```

This command will update statistics values of tables jeff.emp, and jeff.dept.

On the other hand, DBMaster provides setup of date to update statistics automatically. Thus the database server can update statistics by some prefix date. Please refer to chapter 7 “Managing Schema Objects” for more information about updating statistics automatically.

5.7.3 Load and Unload Statistics

Users can use command “**unload statistics**” to dump the statistics values to an external text file. Besides, users can also use command “**load statistics**” to read statistics values to database from an external text file.

```
dmSQL> unload statistics to file1;   // dump statistics from database to file1
dmSQL> load statistics from file1;   // read statistics from external text file file1
```

An experienced user can enhance the efficiency of query by means of modifying files with statistics, and input it into database.

The following case gives the contents of the external file generated from "**unload statistics**".

```
DBname = TESTDB

TBowner = jeff
TBname = emp
TBpage = 5
TBrows = 30
Tbavlen = 50

COname = age
COtype = INTEGER
COdist = 12
COavlen = 4
COlow = 25
COhigh = 42

IXname = idxage
IXpages = 5
IXlevel = 2
IXleaf = 3
IXdist = 12
IXdistC1 = 12
IXdistC2 = 12
IXdistC3 = 12
IXpgkey = 8
IXcount = 7
```

5.8 Accelerate the Execution of Query

Generally speaking, users can accelerate the execution of query by the following modifications:

- Reading less data rows.
- Avoid sorting, or sort on less data rows, or sort on less data columns.
- Using sequential way to read data.

It is not so intuitive to achieve the above goals. It depends on the design of application programs and database. We will state some techniques and its limitations under some conditions.

5.8.1 DATA MODEL

The definition of data model includes all tables, views, and indices on the database, especially the existence of indices. It describes whether an index be used in the conditions such like join, sort, and views or not.

5.8.2 Query Plan

You can use command "set dump plan on" to check the query execution plan of DBMaster. We list some characteristics of execution plan.

- Index: check the output data, to see whether an index been used or not. If so, how to use it?
- Filter: check the predicate (factors), to see whether the predicate can filter many data or not.
- Query: check query at final, to see whether the access plan is the best or not.

For example, we can use the following command to see the execution plan.

```
dmSQL> set dump plan on;
```

5.8.3 Check the Index

Check whether there exist proper indices on query columns or not. You can use methods mentioned in the following sections to approve the query efficiency.

5.8.4 Adjust Filter Columns

It only uses a small part of information source for an efficient query. Users can use where predicate in select command to control the amount of output information. This is called filter of data.

Here we list some methods to advance where predicate:

5.8.4.1 Avoiding Correlated Sub-queries

Correlated sub-queries means that there is a column appeared in main query and the sub-query in the where predicate. It is different of the result of a sub-query for each data row in the main query. To a sub-query, if the data of columns of each row are different from the one in previous row, then it is equivalent to execute a new query for each row gain from the main query.

If you have found a time-consuming sub-query, first you can do is to check whether it is a correlated sub-query or not. If so, rewrite the query to avoid this condition. If it is not easy to rewrite the query, try other ways to reduce the number of data rows.

5.8.4.2 Avoiding Difficult Regular Expressions

Key word "**like**" provides the comparison of wild card (we call it the regular expression). When we use wild card at the beginning of expression, database server will check each row because it cannot use index to filter any rows. This will make DBMaster sequentially access and check every row in a table.

For example:

```
dmSQL> select * from emp where name like '*st';
```

5.8.5 Re-consider the Query

When you understand what a query really does, you can find another equivalent query to get the same result. We give some suggestions for users to rewrite an efficient query.

- Rewrite joins by views.
- Avoid or reduce sorting.
- Avoid access large table sequentially.

- Use union to avoid sequentially access.

5.8.6 Use Temporary Table to Accelerate Query

It is useful to create a temporary, ordered table to accelerate query. It also can help you to avoid sorting operations on multiple columns, and simplify the operation of optimizer. You can

- Use a temporary table to avoid sorting on multiple columns.
- Replace sorting on non-sequential access.

5.9 Syntax-Based Query Optimizer

You can now manually specify the type of scan to use in a query, and which index to use in an index scan. In addition, the DBMaster query optimizer now automatically determines the most efficient type of scan to use, even if you have not recently updated database statistics. There are five different cases where you can specify the type of index you want to use.

5.9.1 Force Index Scan

You can force an index scan with the following syntax:

```
tablename (INDEX [=] idxname [ASC|DESC])
```

In addition to specifying an index name to scan, you can also specify the value 0 to force a table scan, or the value 1 to force a primary key index scan.

The following example forces a table scan.

```
SELECT * FROM t1 (INDEX=0)
```

The following example forces an index scan on the primary key.

```
SELECT * FROM t1 (INDEX=1)
```

The following example forces an index scan on the index idx1.

```
SELECT * FROM t1 (INDEX idx1)
```

The following example allows the query optimizer to decide what type of scan to use on table t1, but forces an index scan on the index idx1 for table t2.

```
SELECT * FROM t1, t2 (INDEX idx1)
```

5.9.2 Force Index Scan With Alias

You can force an index scan and provide an alias for the table with the following syntax:

```
tablename (INDEX [=] idxname) aliasname
```

The following example forces an index scan on the index idx1, and provides an alias for the table.

```
SELECT * FROM t1 (INDEX idx1) a, t1 b WHERE a.c1 = b.c1
```

5.9.3 Force Index Scan With Synonym

You can force an index scan when using a synonym with the following syntax:

```
synonymname (INDEX [=] idxname)
```

The following example forces an index scan on the index idx1 when using synonym s1.

```
SELECT * FROM s1 (INDEX idx1)
```

5.9.4 FORCE INDEX SCAN WITH VIEW

You can force an index scan when creating a view with the following syntax:

```
viewname (INDEX [=] idxname)
```

The following example forces an index scan on the index idx1 when creating view v1.

```
CREATE VIEW v1 as SELECT * FROM t1 (INDEX idx1)
```

But you can not force an index when selecting a view. The following example is a wrong usage and will return errors.

```
SELECT * FROM v1 (INDEX idx1)
```

5.9.5 FORCE TEXT INDEX SCAN

You can force a text index scan with the following syntax:

```
tablename (TEXT INDEX [=] idxname)
```

The following example forces a text index scan on the text index tidx1.

```
SELECT * FROM t1 (TEXT INDEX tidx1)
```

5.10 Rewrite Query

5.10.1 AVOID SUBQUERIES : REWRITE AS JOIN IF POSSIBLE

- select * from t1 where c1 in (select c1 from t2);
- select t1.* from t1,t2 where t1.c1=t2.c1;

5.10.2 AVOID EXPRESSION AND BUILD-IN FUNCTION IN PREDICATE

- select * from t1 where c1*10=100;
- select * from t1 where c1=10;

5.10.3 AVOID 'OR': REWRITE AS 'IN' IF POSSIBLE

- c1=1 or c1=2
- c1 in (1,2)
- (c1>=1 and c1<=3) or (c1>=5 and c1<=7)
- c1 in (1 to 3, 5 to 7)

5.10.4 AVOID 'BETWEEN' : REWRITE AS 'AND' IF POSSIBLE

- c1 between 3 and 5
- c1>=3 and c1<=5

5.10.5 AVOID TABLE SCAN : REWRITE AS UNIONS IF POSSIBLE

- select * from t1 where (c1=3 and c3>3) or c2=5;
- select * from t1 where c1=3 and c3>3 union select * from t1 where c2=5;

5.10.6 Use temporary table when need

- select * from t1 where c1=3 and c2 like 'a%' order by c3;
- select * from t1 where c1=3 and c2 like 'b%' order by c3;
- select * from t1 where c1=3 order by c3 into temp;
- select * from temp like 'a%';
- select * from temp like 'b%';

5.10.7 Large Data Update

- select * from tb1 into tb2;
- set load on;
- select * from tb1 into temp_tb2;
- set load off;
- It will be a big improvement when data size over 100M.

5.11 How to Read a Dump Plan

The first step for a user to check a slow query is to read its execution plan. To see a dump plan of a query, you only need to execute the following command before you run your query:

```
dmSQL> set dump plan on;
```

But it seems to be very difficult for a normal user to understand the meanings written in each part of the dump plan (that is, execution plan). Therefore we will give a detailed explanation about dump plan in this section.

At the first glance, we will find that a dump plan is composed of several blocks called ON. In other words, query optimizer divides a query into several ON blocks, and each of them is a logic optimization unit. Then optimizer will optimize every ON block. For a simple or joined query, DBMaster usually has only one ON block, but for a complex query such like subquery, DBMaster may generate more than one ON block, including a main block and its sub blocks.

For every ON block, optimizer finds the best execution method based on cost. It will divide a ON block into several PL blocks, and each one PL block represents an operation, such as scan, join etc.

We will describe the information shown in PL block by several examples in the following paragraphs. Before reading these examples, you should be more familiar with the following nouns from previous sections in this chapter:

```
table scan
index scan
nested join
merge join
factor
```

5.11.1 Example of Table Scan

Let's see the following query, and explain the dump plan in detail.

```
dmSQL> set dump plan on;
dmSQL> select * from t1 where c1>1;
```

Here gives the dump plan:

```
----- begin dump plan -----

{ON Block 0}
ON Type     : SCAN

[PL Block 0]
Method      : Scan
Table Name : t1
Type        : Table Scan
Order       : <none>
Factors     : (1) t1.c1 > 1
I/O Cost    : 101.0
CPU Cost    : 25.3
Sub Cost    : 0.0
Result Rows: 330.0

----- end dump plan -----
```

Now we explain the meaning in each line. The first two lines give the information of an ON block.

```
{ON Block 0} - This is a ON block, and its block ID is 0
ON Type: SCAN - ON block type is scan
```

This ON block contains one PL block.

```
[PL Block 0] - This is a PL block, and its block ID is 0
Method: Scan - This PL block will do a scan operation.
Table Name: t1 - Scan on table t1.
Type: Table Scan - Scan type is table scan.
Order: <none> - Scan order, it is no use for a table scan.
Factors: (1) t1.c1 > 1 - This scan will use filter t1.c1 > 1.
I/O Cost: 101.0 - Estimated I/O cost is 101.0 pages in this scan.
CPU Cost: 25.3 - Estimated CPU cost is 25.3 pages in this scan.
Sub Cost: 0.0 - Estimated sum of costs of this PL block's sub-block. In this
example, it has no sub PL block.
Result Rows: 330.0 - Estimated result rows after this scan and filter
```

5.11.2 Example of Index Scan

```
dmSQL> set dump plan on;
dmSQL> select c1,c2 from t2 where c1>1 and c2=2;
```

The following shows the dump plan:

```
----- begin dump plan -----

{ON Block 0}
ON Type     : SCAN
```

```
[PL Block 0]
Method      : Scan
Table Name : t2
Scan Type  : Index Scan on idx21(c2, c1)
Order       : ASC
Index EQFA#: 1
Index FA#  : 2
Index FACOL: 1, 2
Index Cost : 2
Factors     : (1) t2.c2 = 2
            : (2) t2.c1 > 1
I/O Cost   : 2.0
CPU Cost   : 0.6
Sub Cost   : 0.0
Result Rows: 13.0

----- end dump plan -----
```

Similarly, the plan looks like the above one.

```
{ON Block 0} - This is a ON block, and its block ID is 0.
ON Type: SCAN - This ON block type is scan.
```

This ON block also contains one PL block.

```
[PL Block 0] - This is a PL block, and its block ID is 0.
Method: Scan - This block executes scan.
Table Name : t2 - Scan on table t2
Scan Type: Index Scan on idx21(c2, c1) - Scan type is index scan. Apply index
idx12, and the index column is c2, c1.
Order: ASC - Scan order, index scan by ascent order.
index EQFA#: 1 - Equal factor number that can be applied in this index scan, in
this example is t2.c2 = 2.
Index FA#: 2 - Factor number that can be applied in this index scan, in this
example is t2.c2 = 2 and t2.c1 > 1.
Index FACOL: 1, 2 – Factor ID that mapping from index columns. In this example, it
means that the first index column c2 maps to factor (1) t2.c2=2,  and the second
index column c1 maps to factor (2) t2.c1 > 1.
Index Cost : 2 - Estimated index page cost is 2
Factors: (1) t2.c2 = 2
         (2) t2.c1 > 1 - Apply filters t2.c2 = 2 and t2.c1 > 1 in this scan.
I/O Cost: 2.0 - Estimated I/O cost is 2.0 pages in this scan.
CPU Cost: 0.6 - Estimated CPU cost is 0.6 it in this scan.
Sub Cost: 0.0 - Estimated sum of costs of this PL block's sub-block.
Result Rows: 13.0 - Estimated result rows after this scan and filter.
```

5.11.3 Example of Join

The following command will show the plan of a equal join.

```
dmSQL> set dump plan on;
dmSQL> select * from t1, t2 where t1.c2=t2.c2;
```

The plan of a join is much different from the simple scan.

```
----- begin dump plan -----

{ON Block 0}
ON Type     : JOIN

[PL Block 0]
Method      : Join
Type        : Merge Join
Factors     : (1) t1.c2 = t2.c2
I/O Cost    : 8.5
CPU Cost    : 573.8
Sub Cost    : 231.6
Result Rows: 500.0
Sub Block 1: [PL Block 1]
Sub Block 2: [PL Block 2]

[PL Block 1]
Method      : Sort
I/O Cost    : 4.2
CPU Cost    : 274.4
Sub Cost    : 120.0
Result Rows: 1000.0
SUB Block   : [PL Block 3]

[PL Block 3]
Method      : Scan
Table Name : t2
Type        : Table Scan
Order       : <none>
Factors     : <none>
I/O Cost    : 101.0
CPU Cost    : 25.3
Sub Cost    : 0.0
Result Rows: 1000.0

[PL Block 2]
Method      : Sort
I/O Cost    : 4.2
CPU Cost    : 274.4
Sub Cost    : 120.0
Result Rows: 1000.0
SUB Block   : [PL Block 4]

[PL Block 4]
Method      : Scan
Table Name : t1
Type        : Table Scan
Order       : <none>
```

```
Factors     : <none>
I/O Cost    : 101.0
CPU Cost    : 25.3
Sub Cost    : 0.0
Result Rows: 1000.0

----- end dump plan -----
```

In this example, there is more than one PL block. We combine this PL block relationship using their sub block information and draw a tree:

Figure 2

and we replace each node of the tree by the method.

Figure 3

Now, let me describe join and sort block respectively.

```
[PL Block 0] - This is a PL block, and its block ID is 0.
Method: Join - This block is a join.
Type: Merge Join - Join type is merge join.
Factors: (1) t1.c2 = t2.c2 - Apply join filter t1.c2 = t2.c2 in this join block.
I/O Cost: 8.5: Estimated I/O cost is 8.5 pages in this join block.
CPU Cost: 573.8 - Estimated I/O cost is 573.8 pages in this join block.
Sub Cost: 231.6 - Estimated sum of costs of this PL block's sub-block.
Result Rows: 500.0 - Estimated result rows after this join block.
Sub Block 1: [PL Block 1] - This block's first child links to [PL Block 1].
Sub Block 2: [PL Block 2] - This block's second child links to [PL Block 2].
```

The above is the join block. We give the description of sort here.

```
[PL Block 1] - This is a PL block, and its block id is 1.
Method: Sort - This is a sort block.
I/O Cost: 4.2 - Estimated I/O cost is 4.2 unit in this sort block.
CPU Cost: 274.4 - Estimated CPU cost is 274.4 unit in this sort block.
Sub Cost: 120.0 - Estimated sum of costs of this PL block's sub-block.
Result Rows: 1000.0 - Estimated result rows after this sort block.
SUB Block: [PL Block 3] - This block's child block link to [PL Block 3].
```

We have listed the most common cases of dump plans that users will meet. Of course, there are still a lot of changes in dump plan. But they are all consists of the same elements, that is I/O cost, CPU cost, and Result Rows. Once you find that the dump plan is not reasonable, you can use syntax-base optimizer discussed in previous section to try other methods, or you should check whether the statistics values are out of date or not.

6. Redesigning Application

6.1 Redesigning the architecture of application

The user should use the advanced functions of the Database itself as much as possible, and shuoldn't realize so much data processing in the Application. You can pick out the data processing from the Application, then use the Database's function to achive. For example, it is available to use Database funcions such as Primary key,Foreign key ,default and check to realize the restriction processing when inputing and maintaining the data. And use trigger,stored command, stored procedure and replication to realize the same function for the other logical processing.

6.2 Reducing lock contention

Resource contention occurs in a multi-user database system when more than one process tries to access the same database resources simultaneously. This will lead to a deadlock when two or more processes wait for each other. Resource contention causes processes to wait for access to a database resource and reduce system performance.

When accessing data in a database, DBMaster processes will lock the target objects (records, pages, tables) automatically. When two or more processes want to lock the same object, one must wait. If more than two processes wait for other processes to release the lock, a deadlock will occur. When a deadlock occurs, DBMaster will sacrifice the last transaction by rolling it back. Thus deadlock reduces system performance.

There are four methods to Prevent/Avoid Deadlock:

- Set table's lock mode to row
- Reduce unnecessary indexes or index columns
- Access tables in sequence *e.g., two tables t1, t2, always update t1 before t2*
- Shorten the transaction

Note: a multi-process application must process time out or deadlock error handling.

By analyzing the "wait for" graph, DBMaster can automatically detect a deadlock situation. If a deadlock is detected, a victim transaction will be aborted to solve the deadlock problem.

Example:

DBMaster detects a deadlock when transaction T2 issues an X lock on Y. Transaction T2 will be aborted to resolve the deadlock problem and the user executing transaction T2 will receive the error message, **"transaction aborted due to deadlock"**:

```
            T1                                T2
--------------                    ------------
share_lock(Y);
read(Y);
```

```
                                    share_lock(X);
                                    read(X);
exclusive_lock(X);
(T1 waits for T2)                  exclusive_lock(Y);
                                   (T2 waits for T1)

                                  T2 aborted by DBMaster
```

6.3 Limiting the number of connections

DBMaster allows no more than 1200 simultaneous session connection to server. If server resources (such as memory, CPU power) are insufficient, it will limit the maximum number of connections. Obviously, too many connections will reduce the performance of database too. So when you no longer use the Database, please disconnect from the Database in the Application.

6.4 Avoiding duplicate connections

Once there is a connection to a database, more memories of server will be used and the usage ratio of CPU will increase.

So we should avoid connection to database but do nothing. We must also avoid duplicate connections.

As connection, system will assign some resources for it, not only memory but also CPU, disk and so on, thus performance will get poor. As connection after disconnection, firstly system will spend some time in releasing some resources occupied by the previous connection. Secondly system will also apply and assign some resources for the next connection. So performance will be poorer.

As we know that, it's no good for performance of DBMaster to **connect** and **disconnect** frequently in Application system.

6.5 Avoid Long Transactions

A long transaction will occupy many lock control blocks and journal blocks. If there is a long transaction in progress when the above error occurs, analyze whether the transaction can be divided into multiple small transactions. Because that the Long Transactions use a lot of resources, that would reduce the Performance directly, and also easy to cause the Deadlock.