CHAPTER 17

ONGOING MAINTENANCE AND GROWTH

CHAPTER OBJECTIVES

- Understand the critical need for ongoing maintenance of the database system
- Note the routine maintenance activities that must continue
- Examine the importance of regular database backups and recovery procedures
- Study how continuous monitoring and gathering of statistics take place
- Appreciate how performance monitoring enables provision for growth
- Review the use of benchmarks and how these help assess the status of the database system
- Note the various aspects of growth and enhancements in a database environment
- Comprehend the significance of performance tuning
- Learn how to tune the various components of a database system

Let us say that you are part of the project team implementing a database system for your company. Perhaps you are the database administrator. The project team has worked through the database development project cycle from planning through implementation and deployment of the database system. The project is successful, and the system is up and running. Everyone is happy; the project team and the DBA can now stop working on the database system. Is this not true?

From Chapter 14 on database administration, you definitely know that the database development life cycle does not simply terminate as soon as you deploy your database system. Maintenance activities just get started at that point and continue as long as the database system is in operation. The database administra-

Database Design and Development: An Essential Guide for IT Professionals by Paulraj Ponniah
ISBN 0-471-21877-4 Copyright © 2003 by John Wiley and Sons, Inc.

tor has the primary responsibility during this continuous period of maintenance and growth.

Within a few short weeks after deployment, you will note an increase in the usage of the database system. In an operational environment, growth does not materialize in just more transactions in the initial applications. Newer applications get developed to take advantage of the database system. The project team, especially the DBA, must be able to provide for the growth.

After deployment, the database system must be closely monitored. Although the usage pattern and frequency of usage could mostly be anticipated, you will see variations very soon after deployment. You will understand that the project team could not foresee every aspect of the usage. So the DBA must carefully monitor usage on an ongoing basis.

In a typical database environment, new users will need authorizations to access the database. Access privileges may have to be changed for the current authorized users. As and when users leave an organization, their access privileges must be revoked immediately. Security administration becomes an important ongoing responsibility of the DBA.

As soon as a database system is deployed and made ready for use, the system is open to potential failures. Safeguarding against failures is an essential ongoing activity. Proper backup and recovery procedures constitute a basic responsibility in the postdeployment period.

Figure 17-1 provides a comprehensive overview of the ongoing maintenance activities in a database environment.

In the previous chapters, we have touched on these activities. Now in this chapter, we will get into more details on the significant activities. One such activity is ongoing tuning of the database for performance. We will consider how various components are tuned so that the database system may perform at optimal performance.

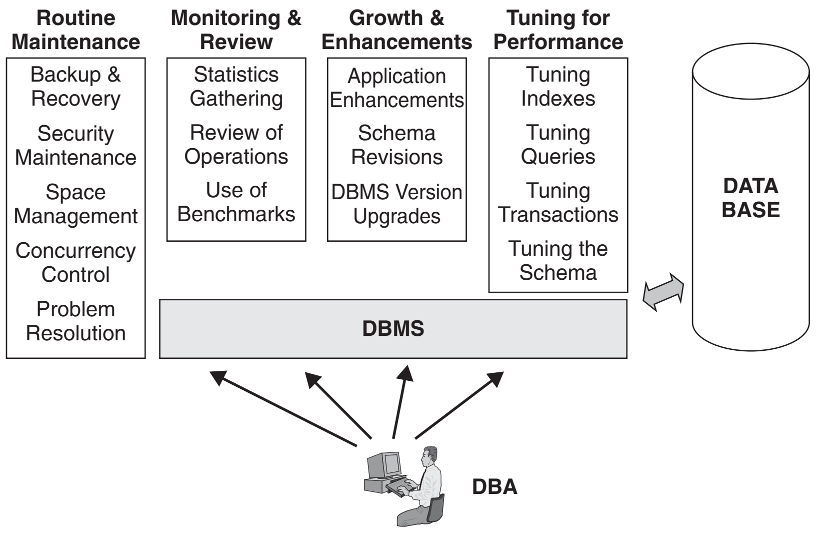

Figure 17-1 Overview of ongoing maintenance.

ROUTINE MAINTENANCE

Just like a mechanical system or any other type of system, to keep the database system working, the database administrator must perform routine maintenance functions. These are not for addressing major problems or recovery from failures. On a day-to-day basis, routine maintenance functions ensure that the components are working well. Routine maintenance functions keep the database performing at the current level without coming to a halt. Let us consider a few key routine maintenance activities.

Backup and Recovery

It does not matter to what extent a database system is safeguarded, in practice failures do occur and they destroy the system partially or completely. Whatever may be the cause of the failure—fire, unintentional or intentional action by people, power failure, or mechanical malfunctions of hardware devices—the DBA must plan and provide for a method to recover and restore the database to a consistent state.

Backing up the database at frequent intervals is obviously an effective, straightforward technique to help in the recovery process. As you know, backup files alone cannot restore the database to the current state if a failure occurs much later than when the latest backup was taken. You know that other files such as the transaction log file are also necessary to bring the database to its most current state.

Recovery System A complete recovery system consists of the following functions and components:

Backup capability. Create backup copies of the entire database at specified intervals. Most database installations create backups at least once a day, usually after the nightly batch update jobs are completed.

Backup files are kept on tape cartridges or optical media other than the disk medium on which databases usually reside. It is customary to keep at least three generations of the backup files, just in case the latest backup file cannot be read during the recovery process. Also, it is advisable to store backup files at an off-site location.

Journalizing capability. Create log files containing details of all transactions since the last full backup of the database. As discussed in a previous chapter, log file records contain before and after images of the database objects that were accessed by individual transactions. The records also keep the timestamp and disposition of each transaction.

Checkpoint capability. Create checkpoint records on the log file to speed up the recovery process. While recovering, the recovery manager module may need to go back only up to the most recent checkpoint record.

Recovery capability. The DBMS should provide a recovery manager module to restore the database to the most recent possible state from the backup and log files.

Figure 17-2 presents the components of the database recovery system.

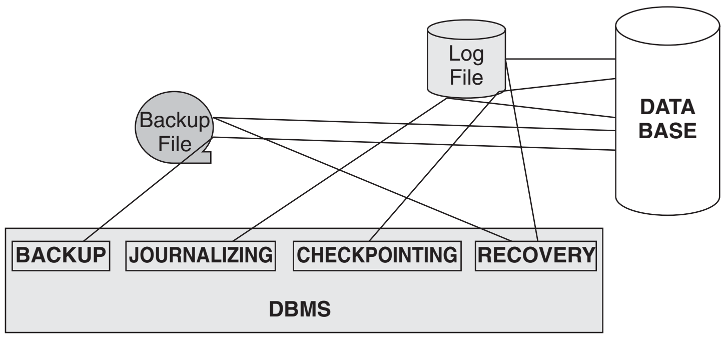

Figure 17-2 Database recovery system.

Routine Backup and Recovery Activities Let us now summarize the routine activities for backup and recovery. Here are the major tasks:

- Schedule regular backups, at least once a day, more often if the nature of your database content warrants it. Find a suitable time window on the computer system to perform the backups. If frequent full backups are not feasible, have a combination of full and partial backups. Partial backups may be hot backups while the database is up and running. Recovery is more difficult with partial backups. Have a good tape management system to make sure that backup tapes are not scratched before their expiration dates.
- Ensure that logging is properly done and that all the necessary details are written in the log records. Partial backups are more difficult for recovery. Monitor the space allocation on disk for the log file so that it does not get filled up before the next archival. Periodically archive the disk log file on other media such as tape cartridges or optical devices. Keep archived log files in safe custody.
- Monitor the frequency of writing checkpoint records. Writing checkpoint records results in system overhead. Adjusting the frequency is a balance between such system overhead and saving time during recovery.
- Review your disaster recovery plan on a regular basis. Have a regular schedule to test the recovery process. This will validate your backup files and the log files. Also, if and when database failure occurs, you will be prepared with a tested procedure.

Recovery Methods How recovery must be performed depends on the nature of the failure, the availability of backup and log files, and the extent of the damage. In failures resulting from disk crashes, depending on how badly the disk gets destroyed, you may have to discard the current devices and use newer or other available ones.

In Chapter 15, Figure 15-31 presented a comprehensive database recovery example. We will now review the applicability of recovery methods to different types of failures. The underlying principle in recovery involves restoring the database to a consistent state as close to the time of failure as possible. The recovery process must be short, fast, and easy.

First, let us explore three standard recovery methods.

Restore and reprocess. This is the simplest of the recovery methods. You take the latest available backup tape and copy it down to the disk extent. From that point, apply or process all the transactions up to the time of failure. Let us say that you create backup tapes every night at 11 P.M. Suppose a failure occurred around 3 P.M. today. You take last night's backup and use it to recreate your database. Then you redo all on-line and batch transactions that executed between 11 P.M. last night up to 3 P.M. today. Although simple and straightforward, this method is not usually acceptable. The time and effort to reprocess all the transactions from the time of the latest backup may be too great. Next, to bring the database to the exact state it was before the crash, you need to reprocess the transactions in the same order they were processed before.

Backward recovery or rollback. In this case, the latest backup tape and the log file are used for recovery. As pointed out in the previous method, the backup tape created last night at 11 P.M. is first used to create the database as it was at that time. Then the recovery manager of the DBMS runs to process the transactions from the log file. All completed transactions found in the log file are applied to the database created by copying the backup file. For transactions in flight or incomplete because of the failure, the before images of data items are applied. All the incomplete transactions must be sorted out and reprocessed.

Forward recovery or rollforward. This is similar to rollback except that, wherever possible, incomplete transactions are completed from the available after images of data items found on the log file. The database is first recreated from the backup file, and then the completed transactions from the log file are applied to the database. The database is then brought forward with any available after images in the log file for the in-flight transactions.

Figure 17-3 shows these three recovery methods. Note the use of the backup and log files in each case.

Applicability to Failure Types When do you apply these recovery modes? Are there any circumstances under which restore and reprocess method can apply? When is forward recovery more appropriate?

Figure 17-4 lists the types of failures and also suggests the recovery method suitable for each type.

Security Maintenance

In Chapter 16 on database security, you studied the significant techniques relating to protecting the database system. As you know, discretionary access controls deal with granting and revoking access privileges to individual users and user groups. Mandatory access control is more stringent, with security classes for database objects and security clearances for users. Here we want to list the types of activities that must be routinely performed as part of the ongoing maintenance.

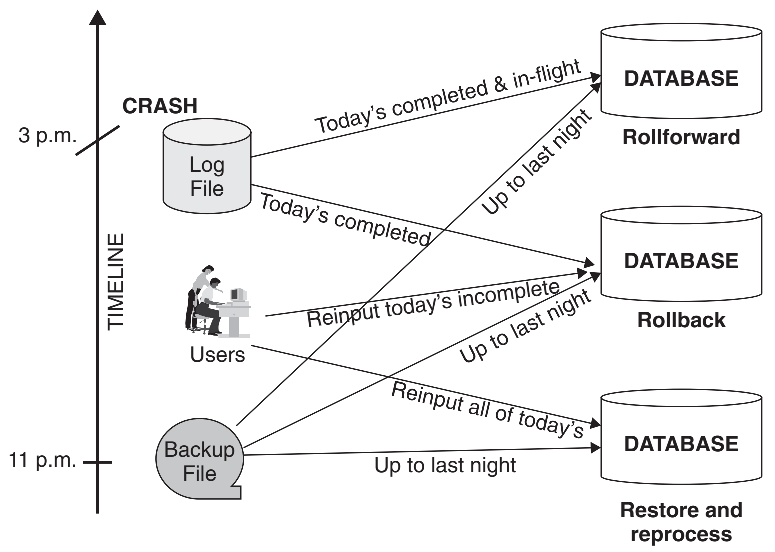

Figure 17-3 Database recovery methods.

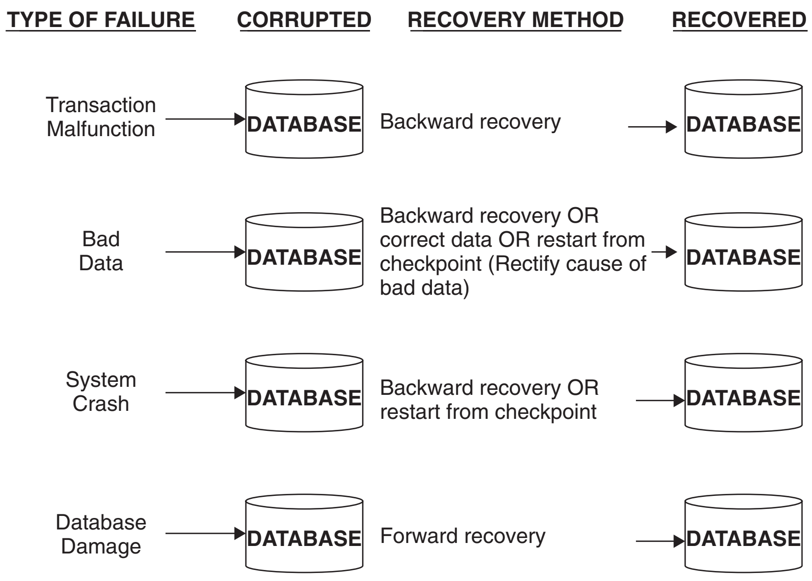

Figure 17-4 Database failures and recovery methods.

Initially, the DBA authorizes users with access privileges. The initial setup usually enables the users to get started with the new database system. However, very soon, changes to the access privileges are warranted. Here are some of the reasons for revisions to access privileges:

- Access privileges must be revoked immediately for people who leave the organization.
- Access privileges must be granted to new employees. This may take the form of a preliminary set of privileges as soon as an employee joins the organization and upgraded privileges soon after the initial orientation and training.
- When staff members move from one region to another or from one division to another, their responsibilities change. Consequently, their access privileges must be revised.
- When employees get promoted to greater responsibilities, their need to access information also changes. Their access privileges need revisions.
- User groups or departments pick up additional responsibilities or shed some current responsibilities. If group authorizations are in place, these will need upgrades or downgrades.
- For organizations using mandatory access controls, database objects may be reclassified because of the changes in the sensitivity of the data content of these objects. In these cases, the security clearances of users may need to be revised.

In large organizations where there are numerous staff movements, security administration takes up substantial time. Special security administrators are employed in such organizations to keep the security authorizations current.

The following is a list of the types of routine security functions in a database environment after deployment:

- Create new user accounts and assign passwords to new eligible employees.
- Grant fresh set of access privileges to new eligible employees.
- Upgrade or downgrade access privileges to existing employees.
- Create, revise, or drop user groups or roles.
- Create views for the purpose of tailoring access privileges to specific users or user groups.
- Deactivate user accounts of terminated employees.
- Revoke access privileges of terminated employees.
- Drop views that are no longer needed for security and other purposes.
- Revise security classes for database objects and security clearances for users if your organization uses mandatory access control.

Space Management

If you are a DBA, every morning you will be monitoring the space utilization of your production database files. You cannot let any production files get completely filled up and stop any transaction from completing for want of space in the middle of the day. Space management is a continuous activity for the DBA.

Some authorized users are also allowed to create test database tables and a few temporary, private tables. Frequently, these tables are not dropped even long after their purposes are served. Review of such obsolete tables and removing them also forms part of routine maintenance.

Here are the routine activities relating to space management:

- Monitor space utilization of all production files regularly.
- Expand disk space for files reaching the end of the allocated space.
- Archive database records not needed for regular, current use. Maintain archived tape files in a safe place.
- Whenever needed, bring back archived tape files, restore them on a special instance of the database, and allow users to use the data.
- Drop one-time, obsolete tables and recover disk space.
- Allocate space quotas to authorized users for creating one-time temporary tables. Revise or remove quotas as and when necessary.

Concurrency Control

Chapter 15 covered concurrent processing of transactions and methods for controlling concurrency from producing adverse effects. You have noted the need for concurrency and also studied the locking techniques needed to prevent problems associated with concurrent transactions. Although database management systems provide adequate lock management, the DBA must still keep monitoring locks and potential deadlock situations. Routinely, the DBA has options to manage and adjust concurrency control techniques.

Usually, a DBMS provides utility scripts to check the current state of a database. When a database system is running slowly and two or more transactions are waiting in deadlock situations, the DBA can run the script and find out what locks are held and which transactions are waiting. This facility usually provides the following information that can be used to resolve deadlocks:

- Transactions that have another transaction waiting on a lock and are not themselves in a wait state
- DDL locks held and requested (DDL locks prevent changes to database definitions while a transaction is executing.)
- DML locks held and requested (DML locks relate to read/write locks in shared and exclusive modes.)
- All locks held or requested
- Transactions that are waiting for database locks and transactions that are currently holding locks
- Locked database objects and the transactions that are currently accessing them

The locking mechanisms of most DBMSs work well. However, if some transactions do not lock at the lowest level of database objects needed by them, deadlock situations could result. Manual intervention by the DBA becomes routinely necessary. After finding out the transactions that are caught in deadlock situations, the DBA has the option of terminating one or more transactions in deadly embrace, thus enabling the waiting transactions to proceed.

Problem Resolution

The DBA is involved in detection and resolution of certain problems on an ongoing basis. Here are a few major problem resolution tasks.

System malfunctions. When the database system stops functioning because of some type of malfunction of the operating system or the DBMS itself, the DBA takes a core dump of the main memory and registers for determining the nature of the problem. The dump is taken before restarting the database system.

Follow-up and coordination. Typically for system malfunctions, the DBA has to seek outside help from the vendors of the operating system and the DBMS. The DBA sends the core dump to the outside vendors. He or she needs to follow up and coordinate the problem resolution with the vendors.

Application of fixes. Usually vendors of the operating system or the DBMS supply code to fix or patch up the system malfunctions. The DBA applies the fixes under appropriate conditions and tests the solutions.

Transaction failures. Whenever any transaction does not execute successfully and aborts, the DBA gets involved in determining the cause of the problem and finding a resolution. Transactions abort for many reasons such as bad data, incorrect coding, or some external conflict.

User support. The DBA and other project team members provide support to the users of the database system by providing clarifications, educating the users on database concepts, and resolving any of their database problems. The DBA is part of the user support loop.

MONITORING AND REVIEW

After deployment of the database system, in addition to performing routine maintenance tasks, the DBA continues to inspect system performance. The project team continues to monitor how the new database system matches up with the requirements and stores the desired information content. The DBA constantly examines the various factors that affect system performance. He or she needs to know the growth trend and whether the database will be able to scale up. The DBA also has to keep the database performing at an optimal level to process transactions efficiently.

How can the DBA and other project team members assess performance levels and be assured of providing for growth? Of course, they need information about how well the database is operating and how smoothly could it accommodate growth. They need statistics about current performance. They need to review various aspects of the design and development of the database system.

Statistics are indicators whose values provide information on how the database is operating. These indicators present growth trends. You will understand how well the database server functions. You will gain insights into how well applications perform in the new database environment.

How do you monitor and collect statistics about the database environment? Two common methods apply to the collection process. Sampling methods and event-driven methods are generally used. The sampling method measures specific aspects of database activity at regular intervals. You can set the duration of the interval. For example, if you want to record the number of active transactions every 20 minutes, this method will produce the numbers three times every hour. The sampling method has minimal impact on system overhead. In the event-driven method, recording of statistics happens only when a specified event takes place, not at regular intervals. If you want to monitor growth of an index table, you can set the monitoring mechanism to record the event every time an update is made to the index table. We will elaborate on these methods a little later. First, you must appreciate the purpose of database monitoring and what monitoring is expected to accomplish.

Purpose of Monitoring

After deployment, actual use of the new database system reveals unanticipated difficulties and factors. During implementation, the project team plans according to the expected usage and access patterns. Not even extensive testing could expose all the variations. Only after the database is sufficiently exercised by the users can the DBA gain real insights.

Actual use for a reasonable period, with execution of all types of transactions, exposes bottlenecks affecting performance. Perhaps the size of the memory buffers constricts the flow of transactions. It is possible that the initial index plan for a particular table hinders quick access instead of speeding up the transactions. All such bottlenecks surface as the database is used more and more. Therefore, ongoing monitoring and collection of statistics becomes important.

At a high level, two major purposes prompt database monitoring and collection of statistics:

- Discerning the growth pattern, estimating additional resources, and planning for future needs
- Tuning the database system for performance

At a more detailed level, database monitoring enables the DBA and other concerned members of the project team to accomplish the following:

- Identify and analyze performance problems in the database system configuration or in the applications using the database system.
- Utilize early warning systems to be alerted to potential problems.
- Discover exception conditions based on defined threshold values.
- Wherever possible, automate corrective actions in response to detected problems.
- Define additional requirements of statistics to be produced in addition to system defaults.
- As and when changes are made to the database system after deployment, assess the effects of the changes.

The Monitoring Process

Above we briefly mentioned two common methods for monitoring the performance and the state of the database system. Now let us get into more details. Let us generalize the two methods as snapshot monitoring and event monitoring. Let us examine how the monitoring process works in each case.

Snapshot Monitoring This method provides a snapshot or picture of the current state of activities in the database system. Suppose you have a problem that is occurring right now. Then you would want a snapshot of the database activities at this moment. If a transaction executes for an inordinately long time, you may want to take snapshots at short intervals to monitor how the transaction is progressing and discover bottlenecks.

You may start the monitoring process and direct the system to take snapshots at specified intervals. At these intervals, snapshots of performance variables are recorded. You may define performance variables on the entire database, a specific table, a disk space for tables, or other similar database objects.

Some systems let you gather cumulative numbers wherever applicable from the start of the monitoring to the current snapshot. Snapshot monitoring systems can also provide comparisons of performance variables at different snapshot points. The comparisons may also be presented in the form of graphs that can display performance trends, peaks, and valleys.

Typical snapshot monitoring systems have graphical interfaces that let you take the following types of actions:

- Set sampling frequency of performance snapshots
- View results of performance computations
- Define threshold values and alert actions
- Generate alerts and view results of alert actions

Event Monitoring With an event monitor, you capture information about transient events you cannot normally monitor through snapshot monitoring. An event monitor lets you monitor events such as deadlocks, completion of a transaction, execution of a particular statement in application code, an insert to a database table, and so on. In addition, you can also receive details about the monitored event. You can know how long a transaction took to complete, how much processor time was used by the transaction, and how many I/Os were necessary.

The details about an event are stored in an event file that can be analyzed by running an analyzer facility, usually as part of the event monitoring system. Details generated by event monitors may be used to manage and tune database systems in the following ways:

- Analyze deadlocks, review locks in participated transactions, and correct lock requests.
- Monitor usage at application level and provide usage accounting.
- Tune long-running transactions by monitoring operations such as reads and writes within the transactions.
- Analyze usage trends, predict future needs, and improve capacity planning.
- Analyze data collected on usage of buffer pools, table areas, and tables.

Gathering of Statistics

While gathering statistics, crucial items to monitor include disk space, free space left for tables, number of users, memory usage, and I/O performance. You should also monitor database usage by individuals, trace data accesses, and database update patterns. Let us list the main types of statistics to be collected. Some of these are automatically collected and stored by the DBMS itself. For others, you have to use monitoring systems.

- Sizes of individual tables
- Number of active tables
- Number of distinct values in table columns
- Number of times a particular query is submitted and completed successfully or unsuccessfully within a period
- Execution times for different phases of queries and transactions
- Storage statistics (tablespaces, indexspaces)
- Number of times DBMS is looking for space in blocks or causes fragmentation
- I/O device performance statistics—total times for read/write operations
- Query processing statistics—execution times, wait times, times for optimization and preparation, I/Os
- Transaction processing statistics—execution times, wait times, times for optimization and preparation, I/Os
- Lock processing statistics—rates of issuing different types of locks
- Logging statistics—log file activity
- Indexing statistics—index creation and accesses
- Input/output usage by user
- Count of authorized users
- Number of read and write operations executed during a set interval
- Number of completed transactions during a set interval
- Audit trail details for selected activities
- Number of page faults during a prescribed interval
- Number of times buffers full condition reached during a given interval
- Profiles of CPU usage during each day

Review of Operations

In addition to monitoring the database system and collecting statistics, the DBA and other team members must continue to review the operations on an ongoing basis. They will be able to spot trends and changes by means of such reviews.

Review the following:

- For each query, tables accessed, attributes used for selection conditions, attributes used for join conditions, and attributes whose values are retrieved
- For each update transaction, type of operation (insert, write, or delete), attributes used in selection conditions, and attributes whose values are changed

- Expected frequencies of individual queries
- Anticipated frequencies of individual transactions
- Time constraints
- Uniqueness constraints in database tables
- Indexes for each tables
- Validity of primary and foreign keys

Use of Benchmarks

As the number of users increases, the size of the database becomes larger, the scope of the applications expands, and the complexity of database operations intensifies, the DBMS tends to become inadequate to provide acceptable performance. Queries and transactions slow down. Conflicts among concurrent transactions escalate, resulting in locking problems. Routine maintenance becomes more and more time-consuming and difficult. You may begin to consider upgrading the DBMS or, perhaps, even replacing it with a more robust one.

How do you go about making decisions to upgrade or replace the existing DBMS? Performance benchmarks provide metrics that can be used for making upgrade or replace decisions. You may apply benchmark tests to the existing DBMS to check how inadequate it has become in the current environment. Also, benchmark tests can be used to justify upgraded versions or new DBMSs considered as replacement.

Benchmarks quantify performance. They allow for variations in database environments. Individual tasks cannot measure performance satisfactorily. You need a collection of tasks to be used to quantify performance. Such a collection or suite of tasks constitutes a performance benchmark. Typically, benchmarks measure performance as transactions per second (tps). Also, they tie in with costs and compute price/performance ratios ($/tps).

Benchmarks defined by the Transaction Processing Council (TPC) are well known and are used more than some other benchmarks from industry. Other proprietary benchmarks proposed by individual vendors are not very useful because they are not universal. Let us examine the TPC benchmarks.

TPC-A. First in a series of benchmarks. Defined in 1989, it measures performance (tps and $/tps) in typical OLTP or operational environments. Simulates a standard bank application with a single type of transaction similar to cash withdrawal and deposit at a teller station, updating three tables, and adding an audit trail record in another table. Measures end-to-end performance of the system including performance at the workstation and on the data communications devices.

TPC-B. Similar to TPC-A, but it measures only the core performance of the DBMS and the OS by themselves. Not a measure for end-to-end performance. Used less widely than TPC-A.

TPC-C. More complex set of database operations than TPC-A or TPC-B. Based on the model of an order entry transaction performing operations to enter order, deliver order, record payment, monitor stock levels, and verify order status.

TPC-D. Not applicable to OLTP systems. Designed for decision support systems such as OLAP systems. Consists of a set of 17 queries in SQL, with some complex SQL features such as aggregation.

When using a particular benchmark, try to understand what it measures, in what context, and in which environment. Is the selected benchmark meaningful to your environment? How well does the benchmark apply to your workload?

GROWTH AND ENHANCEMENTS

Let us assume that the deployment of a database system by your project team is a great initial success. What can you expect in the next six months to a year after deployment? Initial success makes a database system dynamic and vibrant. You can be sure of sharp growth and a demand for additional applications using the database. The benefits of data integration and data sharing will result in greater collaboration among departments.

As users begin to use the initial applications more and more, you will notice substantial increase in data volumes. When the users realize the advantages of the database system, they will demand additional applications. Executives and managers will request new types of applications to provide summary information. Requests for enhancements to the initial applications will come in as a continuous stream.

Business conditions change. Your organization might expand into newer regions, and the users in these new areas would need access to the database. Progressive companies expanding through acquisitions need to have their database enlarged to accommodate additional data. These types of changes also result in revisions and enhancements to the database schema itself.

After a successful deployment of the database system, you must expect growth, revisions, and enhancements. And you must be prepared to manage the growth and enhancements. You must have procedures in place to handle the expansion of the database. We will consider ongoing growth and enhancements by first listing a few practical tips. Then we will cover application enhancements and schema revisions. Finally, we will discuss upgrades to the DBMS.

Ongoing Growth and Enhancements

The underlying principle is to anticipate growth and be prepared. If growth and enhancements are not properly managed, the adverse results will show up as user frustration and loss of productivity. Here are a few tips for managing growth and enhancements:

- Constantly monitor increases in data volumes. Observe the trends and be prepared to have additional disk space at short notice.
- Discourage users, programmers, and analysts from having their own private database tables in the development or test environment. If allowed, very soon these go out of control. All tables must be created through the DBA.
- The entire production database must be under the control of the DBA. Do not permit anyone else to create production tables.

- Have a regular schedule for archiving historical data. Operational systems usually do not need historical data to be available on-line from the database.
- The greater the number of users in an organization, the greater is the need for a fully streamlined procedure for security authorizations. The procedure should ensure strict screening of eligible users, prompt authorizations wherever needed, and immediate revoking of privileges of terminated employees.

Application Enhancements

At the time of deployment of the database system, you have initial applications utilizing data from the database. Users access the database through these applications. Application enhancements include revisions to the initial applications. The revisions may just be enhancing the GUI screens with additional fields or adding new screens. Very often, the enhancements take the form of newer phases of the initial applications. When the database is deployed in an organization for the first time, it is customary to bring the users into a database environment with preliminary versions of the initial applications. Then as the users get more and more accustomed to database concepts, additional versions are implemented.

More involved enhancements relate to brand new applications. For example, the database may be deployed to serve primarily as a repository for an initial order processing application. When this application is pronounced a success, it is quickly followed by an inventory control application. After the inventory control application, another new application may be implemented.

Enhancements have two major requirements. First, application developers need a development or test database environment for testing the changes. This environment is separate from the production database environment. The DBA is responsible for creating and maintaining the test environment. Second, enhancements to the initial applications and development of new applications involve revisions to the database schema. Depending on the nature of the enhancements or the new applications, the revisions could be extensive. We will deal with issues relating to the test environment first. In the next subsection, we will cover schema revisions.

Test Environment We mentioned creating an entire test or development database environment for the purpose of implementing enhancements to applications. Why do you need a separate environment? Does this not involve much effort and resources to have a separate environment operational? Why not accommodate the testing of revisions in the production database environment itself?

You will find that a separate environment becomes necessary not just for enhancements. A separate environment is needed even for resolving problems that arise from time to time. Imagine that an application program that has been in use for some time aborts suddenly. You need to diagnose the problem, fix it, and have the user resume processing. The program execution must have tried to go through one of the paths not taken until now. You need test data with proper conditions to determine the reason for the problem and resolve it. You cannot add suitable test data in the production database and corrupt it. You need a separate test database environment.

Let us list a few significant issues relating to the test or development database environment.

Initial creation. Create the initial test database system by copying the production schema. The test system will start out as a replica of the production system.

Data from production. Move a sufficient number of rows from production tables to corresponding test tables. It is not practical to have the entire contents of the production database copied into the test database. However, you must have enough data to support adequate testing of application enhancements. Have a suite of special programs to select related records from various files. For example, if you select 1000 rows from the customer table, you must select the related orders, not random ones, to populate the orders table. Also, the suite of programs will usually have programs to copy full reference and code files. For example, you would want to copy the entire product file because the selected orders may contain many sets of products.

Test data. In addition to selected data moved from the production database, from time to time, special test data need to be created to test specific conditions. Have programs ready for creating special test data.

Test database operation. The production database is up and running every day during the designated hours. Should the test database be also up and running every day? Most companies discover the necessity of keeping the test database up every day to support the testing of application enhancements. Similar to the production job that is used to start the production database every day, a test start-up job is created to point to the test database files.

Test database maintenance. While test versions of application programs access the test database, very often, program malfunctions corrupt the database. As the test versions of the programs are still being tested, they are not free from error conditions. After some time, the test database gets so corrupted that it becomes useless for further testing. Therefore, periodically, you have to reload the test database from the production database and recreate any special test data.

Duration of test system. Is there a life span for the test database? Should it be kept indefinitely? Almost all organizations with database systems carry a test system indefinitely. However, the composition and data content are likely to keep changing.

Test program versions. Ensure that the test program versions are restricted to access only the test database and never the production database. Only after the programs are fully tested and approved, should they be promoted to access the production database.

Recovery. When failures occur in the test environment, should the test database be restored? Usually, this is not done except when you want to test your recovery procedure. Otherwise, simply refresh the test database with selected production data and proceed.

Schema Revisions

Application enhancements generally require revisions to the database schema. Depending on the type and extent of application enhancements, schema revisions may comprise the following kinds of revisions:

- Changes to data types or sizes of individual attributes
- Additions of new columns to individual tables
- Additions of new tables
- Deletions of any redundant columns
- Changes to uniqueness constraints
- Inclusion of new foreign keys

The revisions to the database schema are first made in the test database environment, tested, and only then applied to the production schema. Figure 17-5 illustrates how schema revisions are implemented.

Here are a few practical suggestions with regard to schema revisions:

Approval for revisions. All revisions to database schema must be subject to serious consideration and approval. Scrutinize the revisions and ensure that they do not violate any standards. Extensive revisions must be treated with great care as if they are part of the overall design.

Procedure for revisions. Have a formal procedure establishing the method for requesting, testing, validating, and approving the revisions. Include the users concerned as part of the process. Identify the responsibilities and assign relevant tasks to the DBA, programmers, analysts, and user representatives. Each revision must follow the prescribed procedure.

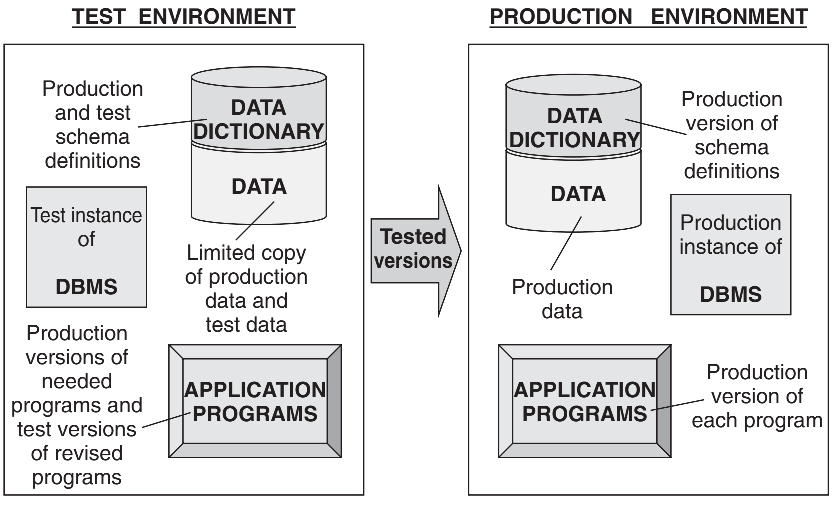
TEST ENVIRONMENT
PRODUCTION ENVIRONMENT
Production and test schema definitions
DATA DICTIONARY
DATA
Test instance of DBMS
Limited copy of production data and test data
Production versions of needed programs and test versions of revised programs
APPLICATION PROGRAMS
Tested versions
DATA DICTIONARY
DATA
Production version of schema definitions
Production instance of DBMS
Production data
APPLICATION PROGRAMS
Production version of each program

Figure 17-5 Database schema revisions.

Testing revisions. Revisions are tested through test versions of application programs. Have queries ready to view data changes in the database. Ensure that referential integrity and other constraints are not violated.

Revisions promoted to production. When test versions of application programs are promoted to production, the corresponding schema revisions must also be moved to the production environment. Have a good approval procedure for the move to production. For major revisions involving creation of new tables, you will be required to populate the new tables with initial data.

Fallback procedure. When you promote test versions of application enhancements and the corresponding schema revisions to production, if you run into unforeseen problems after the move to production, have a fallback procedure to revert to the previous state of the schema. However extensive testing could have been, there is always a slight chance of revisions not working properly; safeguard against such mishaps with a good fallback procedure.

DBMS Version Upgrades

From time to time, your database vendor will announce upgrades to the DBMS. Some of these newer versions contain fixes to problems noticed in the earlier versions. Most of the newer versions provide enhancements and better ways of managing the database versions. Many of the recent upgrades to relational DBMS focus on the merging of database and Internet technologies. Whether or not enhancements in the upgraded versions are directly beneficial to your environment, eventually you will be forced to move up to the latest version. Vendors usually discontinue supporting older versions of DBMSs after a designated period.

Version upgrades cause potentially serious interruption to the normal work unless they are properly managed. Good planning minimizes the disruption. Vendors try to pressure their customers into upgrades on their schedules based on their new releases. If the timing is not convenient for you, resist the initiatives from the vendor. Schedule the upgrade at your convenience and based on when your users can tolerate interruption.

The DBA takes primary responsibility in installing DBMS version upgrades. But others from the database and application project teams as well as user representatives assist with the testing. Here is a brief list of major tasks associated with DBMS version upgrades:

- Coordinate with vendor for assistance during the upgrade process
- Ensure that new version is fully tested by the vendor and free from any errors
- Prepare test database system to install new version
- Configure the new version with proper initialization parameters
- Revise database start-up job, if necessary
- Create a test plan and review the plan with other team members and users
- Compile list of programs to be tested with test conditions and assign responsibilities for testing
- Verify and approve test results

- Install new version in production and perform a dry run
- Run selected programs and queries in production before turning over new version for use

TUNING FOR PERFORMANCE

During the design and implementation phases, all the facts and statistics are not available. The designers and developers go by the information obtained while gathering requirements. As users begin to use the database after deployment, new usage patterns emerge. Over time, usage patterns change. Also, the size of the database increases. Ongoing tuning for performance becomes an essential activity.

In a database environment, problems and bottlenecks affect the performance at four levels: (1) the hardware level, primarily the data storage devices, (2) the level of the operating system that enables the actual interface with the storage devices, (3) the level of the DBMS that manages the database, and (4) the level of the applications that execute transactions and queries. Therefore, any effort at tuning must include considerations at all these four levels. The DBA coordinates the tuning activity, taking primary responsibility for tuning the DBMS. The technical support personnel perform the tuning of the operating system and deal with hardware configurations. Applications staff such as programmers and analysts get involved in applications tuning.

Figure 17-6 illustrates the nature of tuning the database environment at different levels.

Tuning the database environment involves looking at various aspects of performance. The following is just a sample of the factors to be considered while tuning for performance:

- Database configuration parameters
- Disk block utilization parameters
- Memory buffer pools

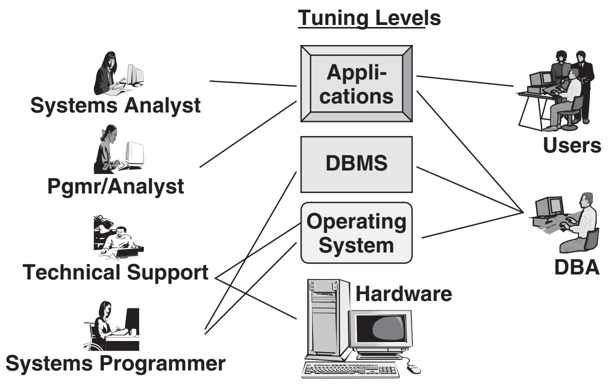

Figure 17-6 Tuning the database environment.

- Prefetching of data pages
- Locking patterns
- Logging overhead
- Memory allocation
- Disk space allocation
- Storage device read/write mechanisms

Goals and Solution Options

Three broad goals prompt the tuning effort in a database environment:

- Improve overall system throughput
- Reduce response times for transactions and queries
- Speed up applications

Bottlenecks Performance is generally affected by constrictions or bottlenecks in one or a few components. Removing bottlenecks produces the greatest benefit. A database environment is like a complex queuing system. Bottlenecks in queuing systems can slow down and even halt movement. In the same way, bottlenecks in a database system can slow down the execution of queries and transactions. Before undertaking a formalized tuning effort, first try to discover bottlenecks and eliminate them.

Figure 17-7 presents a database environment as a queuing system and demonstrates how bottlenecks can make transactions and queries sluggish.

Tuning Options Performance tuning is an elaborate topic. Getting into too detailed a discussion is beyond our scope here. We will consider three areas for per-

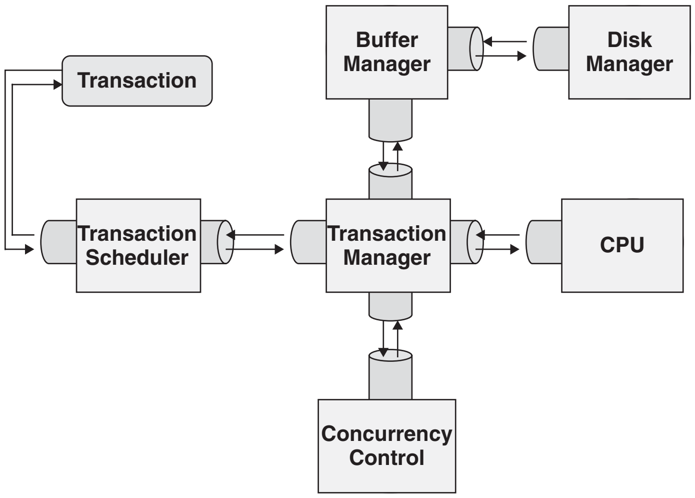

Figure 17-7 Potential bottlenecks in the database environment.

formance tuning: indexes, transactions and queries, and schema. Before that, let us first state some obvious examples of tuning options at the four levels.

Hardware level. This is the lowest level at which tuning activities may be performed. If the bottleneck is at the storage devices, some solution options are to add disks or use improved configurations such as RAID technology. If main memory buffer size is a restriction, try to add more memory. If the processor is a bottleneck, upgrade to a faster processor.

Operating system level. Look into page sizes and block sizes. Consider the block usage parameters and overhead to maintain overflow blocks. Review page faults and page swapping.

DBMS level. Scrutinize and adjust buffer sizes, checkpoint intervals, and deadlock resolution methods.

Applications level. Review the schema, tune indexes, and improve data access schemes. At every possible opportunity, employ stored procedures for which syntax checking, validation, optimization, and compilation have all been done already.

Tuning Indexes

As you know, proper indexes speed up processing considerably. Indexes are the most effective means to improve performance. After deployment, with users settling down in their usage patterns, and with the availability of statistics, indexes may be tuned successfully. Some queries may run long for lack of indexes; certain indexes may not be utilized at all; a few indexes may cause too much overhead because the attributes used for indexing are too volatile.

DBMSs provide trace facilities to sketch out the flow of operations in a query or transaction. Use the trace on a query or a transaction to determine the indexes used. Verify whether the indexes are appropriate. If not, revise the indexing.

The following is a list of general guidelines for tuning indexes:

- If queries are running slow, create appropriate indexes.
- If transaction updates are problems, reevaluate the indexes and ensure that there are not too many indexes suggesting inefficient execution paths to the optimizers.
- Make sure that proper indexing technique is defined for each index. For queries with selections based on ranges of values, B-tree indexing is appropriate. Hash indexing is best for selections based on exact values.
- Periodically reorganize and rebuild indexes to minimize overflow chains and reclaim empty space. Without reorganization, indexes, especially B-tree indexes, carry too much wasted space because of deletions.
- Optimizers depend on updated statistics about the database content such as number of rows, number of distinct values for attributes, and so on. When there is a choice of indexes for a table, the optimizer selects the right index based on

the statistics. Therefore, run the database utility program frequently to keep the statistics reasonably up to date.

- Drop redundant indexes.
- Add new indexes as suggested by changes in access patterns.
- For a multiattribute index, verify and ensure that the order of the attributes as defined in the index is appropriate for most of the processing using that index.

Tuning Queries and Transactions

When users run applications, they access the database through queries and transactions. Queries perform read operations; transactions execute update, insert, and delete operations. When transactions perform update, insert, and delete operations, most of the time they deal with single records. However, queries may read one record or a complete group of records based on a set of selection criteria. The selection may also involve join operations. Therefore, tuning queries constitutes the most significant effort in the overall performance tuning.

As you have seen above, the query optimizer provided in the DBMS intercepts each query and optimizes the query execution with a proper plan of execution. Usually, you can get an indication that a query does not perform efficiently when you note the following:

- The query issues too many disk I/Os.
- The query plan shows that the relevant indexes are not being used.

When you observe these symptoms, take appropriate action. Sometimes rewriting long-running queries may speed them up. Some DBMSs allow the use of hints to the optimizer. Hints are suggestions to the optimizer to use an index other than what it normally picks for a query when more than one index is available for a database table.

Do not hesitate to run the database utility program provided for asking to explain the plan used by the optimizer. Verify that the query is using the index you expect it to use. Also, make sure the optimizer of your DBMS works well for all types of queries and transactions. Some optimizers are found to perform poorly under the following conditions:

- Selection condition has null values
- Selection condition involves arithmetic or string expressions

Let us close with a few comments on the impact of concurrency on the overall performance.

- In an environment with a high volume of transactions, excessive locking is expected. Such excessive locking hurts performance.
- Duration of locking also affects performance because of transactions waiting for locks to be released.

- Placing of data and indexes across multiple disks, as in a RAID configuration, helps speed up concurrent transactions.
- In environments with a high volume of transactions, B-tree indexes themselves may be the bottlenecks. Change indexing techniques in such cases.

Tuning the Schema

Sometimes, you will find that the design of the conceptual and the physical schema does not meet the objectives of the evolving workload. In that case, the schema itself must be examined. You may consider changing the file organizations and allocation of database records to physical files. Changing the conceptual or logical schema requires substantial effort and time.

Here are a few options to be considered while examining the logical schema for the purpose of revisions:

- Decide to settle for third normal form (3NF) relations instead of further normalization into Boyce-Codd normal form (BCNF). The more you normalize a data structure, the more decomposed the relations become. Data access to retrieve a piece of information will have to extend to extra relations. For all practical purposes, 3NF proves to be adequate.
- Data retrieval patterns may indicate that some parts of a table are more frequently accessed than others. Splitting the table appropriately speeds up processing in such cases. Opt for horizontal or vertical partitioning.
- Denormalize in certain situations. For example, if the DEPARTMENT table and the EMPLOYEE table are in a one-to-many relationship and the DepartmentName attribute is required in most queries, it is worthwhile to denormalize the EMPLOYEE table. Repeat the DepartmentName attribute in the EMPLOYEE table. Such controlled redundancies improve efficiency.

CHAPTER SUMMARY

- The database development life cycle does not terminate as soon as you deploy the database system. Ongoing maintenance just gets started and continues as long as the database is in operation.
- The project team members, especially the DBA, must provide for growth and keep the database system tuned for top performance.
- Routine maintenance includes the following: regular backups and preparation for recovery from possible failures, administration of security, storage space management, concurrency control, and problem resolution.
- In large organizations, security maintenance is a major ongoing activity.
- The DBA and other responsible professionals need to monitor the database environment continually for growth planning and tuning the database for performance.

- Two common methods used for monitoring the database environment and collecting statistics: (1) snapshot monitoring by taking samples of performance indicators at defined intervals, and (2) event monitoring by capturing details about transient events.
- Monitoring systems collect statistics on a large variety of indicators such as table sizes, number of active database tables, execution times of different operations in transactions, storage allocations, lock processing details, database usage by authorized users.
- In addition to monitoring and collecting statistics, also review the status of basic components of the database system—indexes, transaction frequencies, uniqueness constraints, and so on.
- Benchmarks quantify performance of the database environment, particularly the DBMS. Benchmarks defined by the Transaction Processing Council are commonly used.
- Growth in a database environment takes the form of increase in data volume, multiplication of the number of users, enhancements to applications, and schema revisions.
- The DBA and the project team must create and maintain a test database environment for ongoing application development and for testing error corrections.
- On a regular basis the DBA, with the assistance of other team members, tunes the database for efficient performance. Tuning requires usage of statistics about the working of the various components.
- Tuning options relate to four levels of the overall environment: hardware, operating system, DBMS, and application. Three significant areas for tuning: indexes, queries and transactions, and the schema itself.

REVIEW QUESTIONS

1. "The DDLC does not terminate with the deployment of the database system." Explain.
2. What are the major activities of routine maintenance in a database environment?
3. Name the three standard recovery methods. Briefly describe any one method.
4. List any six types of routine security functions.
5. What are the types of details available to the DBA to resolve deadlocks? Generally, what actions does the DBA take?
6. What is snapshot monitoring? Briefly list the types of statistics you can gather through snapshot monitoring.
7. What are benchmarks? How are benchmarks useful in a database environment?
8. List any five major tasks associated with DBMS version upgrades.
9. What are the major factors for consideration while tuning the database system for performance?
10. Name any five general guidelines for tuning indexes.

EXERCISES

1. Match the columns:

1.	restore and reprocess	A.	continues after deployment
2.	DDL locks	B.	reduce recovery time
3.	schema tuning technique	C.	part of concurrency control
4.	space management	D.	measured by benchmark tests
5.	checkpoints	E.	simple recovery method
6.	DDLC	F.	uses database statistics
7.	staff transfers	G.	denormalization
8.	monitoring locks	H.	revisions to access privileges
9.	query optimizer	I.	prevent untimely schema changes
10.	transactions per second	J.	drop obsolete tables

2. As the DBA of a local department store, discuss the components of a backup and recovery system. Write a detailed procedure for regular backups and for testing recovery from different types of failures.
3. Six months after deployment of the database system in your organization, users notice sharp increases in the response times as the day progresses into the afternoon. Describe the steps you would take to diagnose the problems. What types of statistics would you look into? Suggest some solution options.
4. Assume that your organization has over 5000 employees who need access privileges to the database system. As the project manager, you are responsible for setting up an ongoing security administration process. Prepare a detailed plan, outlining all aspects and defining responsibilities.
5. Prepare a detailed task list for installing a version upgrade of the DBMS. Indicate the sequence of the tasks, the responsibilities, and user involvement.