PART E

Advanced Topics

This division of the text covers some advanced topics in database system that you should be familiar with.

The objectives of this division are

- to introduce you to the salient issues related to database administration;
- to introduce you to the theory, advantages, and challenges of distributed databases;
- to provide an overview of object databases, and point out the challenges faced by the approach;
- to provide an overview of data warehousing;
- to provide an overview of Web-accessible databases, and the supporting technologies.

The division includes five chapters, each providing an overview of an area of database systems that could be further explored.

The chapters to be covered include the following:

- Chapter 21 — Database Management
- Chapter 22 — Distributed Database Systems
- Chapter 23 — Object Databases
- Chapter 24 — Data Warehousing
- Chapter 25 — Web-Accessible Databases

Please note that for each of these topics, several texts have been written. It will therefore not be possible to cover them in detail. Rather, in each case, an overview is provided, outlining the salient issues.

CHAPTER 21

Database Administration

We have established the importance of a database as a valuable information resource in the organization. This resource must be carefully administered (managed) in order to ensure the continued operation and success of the organization. In this regard, the database administrator is extremely important (database administrators are among the highest paid IT professionals).

This chapter provides an overview of database administration. It discusses (from an administrative view point) the following issues:

- Database Installation, Creation and Configuration
- Database Security
- Database Management
- Database Backup and Recovery
- Database Tuning
- Database Removal
- Summary and Concluding Remarks

Please note that a solitary chapter which overviews database administration will not make you a good database administrator (DBA). To achieve that objective, you will need to apply the knowledge in this course, combined with additional knowledge gained from a special course in database administration. This chapter helps to prepare you for that vocation by providing an overview of database administration issues.

21.1 Database Installation, Creation, and Configuration

Before any work can be done, the database software must be installed. This is usually a straightforward, but time consuming process. For large, sophisticated products such as Oracle and DB2, installation could get complicated, since decisions have to be made about what components to install, where (in terms of directories or folders) to store certain resources, and what environmental or configuration settings to choose. For simpler products like MySQL and Delphi, the installation process is correspondingly simpler.

Next, the database must be created and configured. Depending on the DBMS suite that is in use, database creation and configuration may be very simple or quite complicated. In some systems (for example Delphi), database creation is a simple act of creating a directory (in the Linux environment) or folder (in the Windows environment), and then creating database aliases that point to that directory or folder. In others such as MySQL, there is a highly simplified **Create-Database** statement that allows you to create a database within seconds. On the other end of the spectrum, there are systems such as Oracle, for which you must first complete a course before you know how to properly create a database (review Oracle's version of the **Create-Database** statement in Chapter 11). And there are those products in between the two extremes.

Once you have created the database, there are issues that must be determined as part of the database configuration. These issues include the following:

- Location of the database and its related files
- Communication issues for server and client (for client-server databases) in a multi-user environment
- Physical structure of the database
- Logical structure of the database
- Users of the database and their access rights

Like database creation, the complexity of these issues, and the flexibility with which they can be addressed, will depend to a large extent on the products involved, as well as the complexity of the database itself.

21.2 Database Security

Database security is a very important aspect of database administration. Ideally, the security mechanism must be multi-tiered, controlling access to the system, the system resources, and the system data. The DBA must ensure the following:

- Access to the system is controlled.
- Authorized users must be able to access (insert, modify, retrieve or delete) data that they are authorized to access.
- Authorized users must be restricted to the data and resources that they are duly authorized to access and nothing more.
- Unauthorized users must have absolutely no access.

In order to achieve this, the DBA must be fully conversant with SQL facilities (commands) for managing database users and objects. The information covered in Chapter 13 is particularly relevant here. Some DBMS suites are marketed with a GUI, which is superimposed on the basic SQL interface, and provides a more user-friendly environment for working, by generating SQL code (which can be subsequently accessed and modified) to GUI-based instructions. Examples of this in the Oracle suite include Oracle Enterprise Manager (OEM), iSQL Plus, and Oracle SQL Developer (OSQLD).

To further reinforce the security mechanism of the database, more sophisticated products provide the facility to encrypt data stored in database files. In other products, the conventional wisdom is to rely on the encryption feature provided by the underlying operating system.

21.3 Database Management

As mentioned in Chapter 11, once the database has been created, it must be populated with database objects. Database objects include tablespaces (specific to Oracle), tables, indexes, views, synonyms, procedures, triggers, packages, sequences, users, roles, etc. Most of these database objects are dynamic and will grow or shrink in size, with the passing of time. The database and its objects must therefore be managed.

Database management is a complex matter that covers a wide range of activities; among these are the following:

- Reorganizing existing database tables and indexes
- Deleting unnecessary indexes or moving other objects
- Making alterations to the database itself
- Making alterations to database components (tablespaces, datafiles, tables, procedures, etc.)
- Creating additional database objects (tablespaces, datafiles, tables, users, indexes, procedures, etc.)
- Training users
- Backup and recovery of database objects
- Database performance tuning

Most of these issues have been discussed in previous chapters. The last two issues — backup and recovery, and database performance tuning — deserve some attention. They will be addressed in the next two sections.

21.4 Database Backup and Recovery

In general, the term database backup and recovery refers to the various strategies and procedures involved in protecting a database against data loss, and reconstructing the data should that loss occur. The reconstructing of data is achieved through media recovery, which refers to the various operations involved in restoring, rolling forward, and rolling back a backup of database files. Like database installation, creation and configuration, backup and recovery can be quite simple or very complex, depending on the database environment, and the desired objectives. For the remainder of this section, we shall consider, as case study, backup and recovery in the Oracle environment (it does not get any more complicated than this).

21.4.1 Oracle Backups: Basic Concept

A backup is a copy of data. This backup may include important parts of the database such as the control file, datafile(s), or tablespace(s); alternately, it may involve the entire database. A backup is a safeguard against unexpected data loss and application errors. If you lose the original data, then you can reconstruct it by using a backup.

Backups are divided into physical backups and logical backups. Physical backups, which are the primary concern in a backup and recovery strategy, are copies of physical database files. You can make physical backups with either the Oracle Recovery Manager (RMAN) utility or operating system utilities. In contrast, logical backups contain logical data (for example, tables and stored procedures) extracted with the Oracle Export utility and stored in a binary file. You can use logical backups to supplement physical backups.

21.4.2 Oracle Recovery: Basic Concept

Recovery is the opposite of backup. A database recovery is effected from a database backup, so if the backup was not done, the recovery is not an option. Like backup, recovery may involve a component or section of the database (from a control file, datafile(s) or tablespace(s)), or it may involve an entire database.

To restore a physical backup of a datafile or control file is to reconstruct it and make it available to the Oracle database server. To recover a restored datafile is to update it by applying archived redo logs, and online redo logs, that is, records of changes made to the database after the backup was taken. If you use RMAN, then you can also recover restored datafiles with incremental backups, which are backups of a datafile that contain only blocks that changed after a previous incremental backup.

After the necessary files are restored, media recovery must be initiated by the user. Media recovery can use both archived redo logs and online redo logs to recover the datafiles. If you use SQL*Plus, then you can run the `RECOVER` command to perform recovery. If you use RMAN, then you run the RMAN `RECOVER` command to perform recovery.

21.4.3 Types of Failures

Several circumstances can halt the operation of an Oracle database. The most common types of failure are described in Figure 21-1. Oracle provides for complete recovery from all possible types of hardware failures, including disk crashes. Options are provided so that a database can be completely recovered or partially recovered to a specific point in time.

Failure Type	Comment
User error	User errors can require a database to be recovered to a point in time before the error occurred. For example, a user may accidentally drop a table. To enable recovery from user errors and accommodate other unique recovery requirements, Oracle provides exact point-in-time recovery. For example, if a user accidentally drops a table, the database can be recovered to the instant in time before the table was dropped.
Statement Failure	**Statement failure** occurs when there is a logical failure in the handling of a statement in an Oracle program (for example, the statement is not a valid SQL construction). When statement failure occurs, the effects (if any) of the statement are automatically undone by Oracle and control is returned to the user.
Process Failure	A **process failure** is a failure in a user process accessing Oracle, such as an abnormal disconnection or process termination. The failed user process cannot continue work, although Oracle and other user processes can. The Oracle background process PMON automatically detects the failed user process or is informed of it by SQL*Net. PMON resolves the problem by rolling back the uncommitted transaction of the user process and releasing any resources that the process was using.
	Common problems such as erroneous SQL statement constructions and aborted user processes should never halt the database system as a whole. Furthermore, Oracle automatically performs necessary recovery from uncommitted transaction changes and locked resources with minimal impact on the system or other users.
Instance Failure	**Instance failure** occurs when a problem arises that prevents an instance from continuing work. Instance failure can result from a hardware problem such as a power outage, or a software problem such as an operating system crash. When an instance failure occurs, the data in the buffers of the system global area is not written to the datafiles.
	Instance failure requires **crash recovery** or **instance recovery**. Crash recovery is automatically performed by Oracle when the instance restarts. In an Oracle9*i* Real Application Clusters environment, the SMON process of another instance performs instance recovery. The redo log is used to recover the committed data in the SGA's database buffers that was lost due to the instance failure.
Media (disk) Failure	An error can occur when trying to write or read a file that is required to operate the database. This is called disk failure because there is a physical problem reading or writing physical files on disk. A common example is a disk head crash, which causes the loss of all files on a disk drive.
	Different files can be affected by this type of disk failure, including the datafiles, the redo log files, and the control files. Also, because the database instance cannot continue to function properly, the data in the database buffers of the system global area cannot be permanently written to the datafiles.
	A disk failure requires **media recovery**. Media recovery restores a database's datafiles so the information in them corresponds to the most recent time point before the disk failure, including the committed data in memory that was lost because of the failure. To complete a recovery from a disk failure, the following is required: backups of the database's datafiles, and all online and necessary archived redo log files.

Figure 21-1. *Types of Database Failures*

If some datafiles are damaged during a disk failure, but most of the database is intact and operational, the database can remain open while the required tablespaces are individually recovered. Therefore, undamaged portions of a database are available for normal use while damaged portions are being recovered. This is a very desirable feature, especially for large corporate databases that must be up and running "twenty four by seven."

21.4.4 Database Backups

Because one or more files can be physically damaged as the result of a disk failure, media recovery requires the restoration of the damaged files from the most recent operating system backup of a database. There are several ways to back up the files of a database.

Whole Database Backups

A whole database backup is an operating system backup of all datafiles, online redo log files, and the control file of an Oracle database. A whole database backup is performed when the database is closed and unavailable for use.

Partial Backups

A partial backup is an operating system backup of part of a database. The backup of an individual tablespace's datafiles and the backup of a control file are examples of partial backups. Partial backups are useful only when the database's redo log is operated in ARCHIVELOG mode.

A variety of partial backups can be taken to accommodate any backup strategy. For example, you can back up datafiles and control files when the database is open or closed, or when a specific tablespace is online or offline. Because the redo log is operated in ARCHIVELOG mode, additional backups of the redo log are not necessary. The archived redo log is a backup of filled online redo log files.

21.4.5 Basic Recovery Steps

Because of the way the Oracle Database Writer (DBWn) writes database buffers to datafiles, at any given time, a datafile might contain some tentative modifications by uncommitted transactions and might not contain some modifications by committed transactions. Therefore, two potential situations can result after a failure:

- Data blocks containing committed modifications were not written to the datafiles, so the changes appear only in the redo log. Therefore, the redo log contains committed data that must be applied to the datafiles.
- Because the redo log can contain data that was not committed, uncommitted transaction changes applied by the redo log during recovery must be erased from the datafiles.

To solve this situation, two separate steps are used by Oracle during recovery from an instance or media failure: rolling forward and rolling back.

Rolling Forward

The first step of recovery is to **roll forward**, which is to reapply to the datafiles all of the changes recorded in the redo log. Rolling forward proceeds through as many redo log files as necessary to bring the datafiles forward to the required time.

If all necessary redo information is online, Oracle rolls forward automatically when the database starts. After roll forward, the datafiles contain all committed changes as well as any uncommitted changes that were recorded in the redo log.

Rolling Back

The roll forward is only half of recovery. After the roll forward, any changes that were not committed must be undone. After the redo log files have been applied, then the undo records are used to identify and undo transactions that were never committed yet were recorded in the redo log. This process is called **rolling back**. Oracle completes this step automatically.

21.4.6 Oracle's Backup and Recovery Solutions

There are two methods for performing Oracle backup and recovery: Recovery Manager (RMAN) and user-managed backup and recovery. RMAN is a utility automatically installed with the database that can back up any Oracle8 or later database. RMAN uses server sessions on the database to perform the work of backup and recovery. RMAN has its own syntax and is accessible either through a command-line interface or though the Oracle Enterprise Manager GUI. RMAN comes with an API that allows it to function with a third-party media manager.

One of the principal advantages of RMAN is that it obtains and stores metadata about its operations in the control file of the production database. You can also set up an independent recovery catalog, which is a schema that contains metadata imported from the control file, in a separate recovery catalog database. RMAN performs the necessary record keeping of backups, archived logs, and so forth using the metadata, so restore and recovery is greatly simplified.

An alternative method of performing recovery is to use operating system commands for backups and SQL*Plus for recovery. This method, also called user-managed backup and recovery, is fully supported by Oracle Corporation, although use of RMAN is highly recommended because it is more robust and greatly simplifies administration.

Whether RMAN is used, or user-managed methods, physical backups can be supplemented with logical backups of schema objects made using the Oracle Export utility. The utility writes data from an Oracle database to binary operating system files. You can later use Oracle Import to restore this data into a database.

21.5 Database Tuning

To maintain acceptable database performance, it is necessary to carry out periodic performance tuning. As the database is being used, it will exhibit a natural tendency to degraded performance. The current database system may not be performing acceptably, based on user-defined criteria, due to any of the following:

- Poor database design
- Database growth
- Changing application requirements (possibly including a redefinition of what acceptable performance is)

Note however, that there might be occasions when database tuning efforts are not fully effective. When components that are external to the database, yet vital to the entire client-server application performance, fail to perform acceptably, database tuning might not help without the corresponding tuning of these other application infrastructure pieces. The main components external to the backend database are the backend operating system, the network, and the client operating system. The major examples are the following:

- Very weak clients (PCs)
- Network saturation
- Very weak, saturated, or poorly tuned operating system

21.5.1 Tuning Goals

There are different ways of determining the goals of a performance tuning effort. A DBA should consider them all. Database systems can be sampled on various quantitative measures; the most important of these are the following:

- Throughput: This is the accomplished work per unit time, as measured by transactions per second (tps); higher is better.
- Response time: This is the time it takes for an application to respond, as measured in milliseconds or seconds, lower is better.
- Wait time: This is the elapsed time a program takes to run; lower is better.

In any system, throughput and response time usually run counter to one another as tuning goals. If response time is high (bad), throughput might be high (good). If throughput is low (bad), response time might be low (good).

Common sense helps when sorting out these two conflicting measures. The more users that are concurrently using a system within a certain amount of time, the more likely it is that each user will experience longer delays than normal, but the number of transactions going through the system will be greater. On the other hand, if you decrease

the number of concurrent users accessing the system within a certain time window, each user will enjoy faster response time at the expense of fewer overall transactions completing in that duration.

Typically, on-line transaction processing (OLTP) systems (also called operational databases) want low response time or high throughput, in terms of transactions per second, depending on the application needs. A decision support system (DSS) also wants low response time. However, a DSS might also want high throughput in terms of data blocks read or written per unit time. This type of throughput is not necessarily counterproductive to high concurrency and low response times. A batch (production) system typically wants lower wait times. For example, everyone likes for the payroll application to complete on time!

It is important to always consider the following two central tuning goals:

- Maximize your return on investment (ROI). Invest your time and effort wisely by working on the problems most likely to yield the optimum improvement.
- Minimize contention. Bottlenecks are characterized by delays and waits; eliminate or reduce these whenever possible.

Finally, the following general-purpose database tuning goals should be considered:

- Minimize the number of data blocks that need to be accessed; review and rewrite database access code as necessary.
- Use caching, buffering, and queuing whenever possible to compensate for the electro-mechanical disadvantage (memory is faster than disk).
- Minimize the data transfer rates (the time it takes to read or write data); fast disks, RAID, and parallel operations help do this.
- Schedule concurrent programs that complement instead of compete with each other.

21.5.2 Tuning Methodology

It is best to approach tuning with a structured methodology. After ensuring that the operating system is performing at its peak and that sufficient operating system resources have been allocated to your database system, you should tune following in this order:

- Database design
- Database application
- Memory management
- I/O management
- Database contention

Database Design Tuning: In database design tuning, you are concerned with the physical and logical structure of the database. During this exercise, the DBA examines and takes decisions about refining the logical and physical structure of the database. Among the issues that may be addressed are the following:

- Determining whether critical database components (tablespaces, datafiles, log-files, etc) need to be redefined and/or relocated (to different directories/folders).
- Determining whether critical database objects (primarily tablespaces and tables) need to be partitioned (i.e. fragmented into different partitions).
- Determining whether critical database tables need to be restructured.
- Determining whether additional database objects (tables, logical views, etc.) are required, and if so, where they should be placed.

Database Application Tuning: In database application tuning, the concern is on end-user access to the database. During this exercise, the DBA determines ways to better facilitate access to the database by the various applications that need to use it. Among the issues that may be addressed are the following:

- Ascertaining whether existing database application objects (procedures, triggers, etc.) are performing according to expectations.
- Determining whether additional database application objects (procedures, triggers, etc.) are required and where to place them.
- Determining whether adequate database access points (including ODBC connections, database service connections, etc.) are in place and are working acceptably.

Memory and I/O Management Tuning: Memory management tuning in closely related to database design tuning. This is critical because poor database design could lead to poor memory performance which in turn leads to poor database performance. Among the issues that may be addressed are the following:

- Storage allocations for database objects (primarily tablespaces, datafiles, and tables).
- Storage allocations for the database itself (these parameters are set at database creation or database alteration).

Oracle provides a number of utilities for managing memory performance of database tables. However, a full discussion of these is beyond the scope of this course. Suffice it to say that the database fault rate on each table can be monitored. If the fault rate is high, the table needs to be reorganized.

Database Contention: Database contention relates to how the database is handling multi-user access as well as concurrent access. Like memory management, there are specific utilities for managing this issue; these utilities are typically provided by the DBMS suite.

21.6 Database Removal

Sometimes, it becomes necessary to remove a database. Often, there is no specific command for this, for obvious reasons: The database exists under the auspices of the host operating system. Database removal is therefore an operating system command.

Depending on the DBMS being used, database removal may be a trivial matter, or one requiring a few steps. For instance, removal of a Delphi or MySQL database involves a single step. On the other hand, removing an Oracle database involves several steps of deleting related folders/directories managed by the DBMS in collaboration with the underlying operating system. These folders/directories were created when the database was created (or altered).

Once a database has been deleted, it is completely gone, and can only be reintroduced via a recovery operation.

21.7 Summary and Concluding Remarks

Here is a summary of what has been discussed in this chapter:

- Depending on the DBMS being used, database creation may be complex or simple. Delphi and Oracle are at the two extreme ends of the spectrum — database creation is very simple in Delphi, and very complex in Oracle.
- Database security must ideally be multi-tiered. It must address access to the system, access to the system resources, and access to data.
- Database management must continue after database creation. It must address issues relating to the performance of the database system in the face of growing data collection and changing user needs. Database tuning is and integral part of this.
- Backup must be carefully planned and methodically implemented, in order to minimize or eliminate data loss due to system failures. The recovery procedures must also be reviewed as required.
- Like database creation, depending on the DBMS used, database removal may be trivial or complex.

21.8 Review Questions

1. What are the main issues to be considered when creating a database?
2. What are the critical issues to be addressed when configuring the security mechanism of a database?

3. What are the main issues to be addressed during the management of a database?

4. Why are backups important? Discuss how backup and recovery are handled in Oracle.

5. Why is performance tuning of a database important? Identify basic tuning guidelines to be observed.

21.9 References and/or Recommended Readings

[Hoffer, 2007] Hoffer, Jeffrey A., Mary B. Prescott and Fred R. McFadden. *Modern Database Management* 8th ed. Upper Saddle River, NJ: Prentice Hall, 2007. See Chapter 12.

[Mullins, 2002] Mullins, Craig S. *Database Administration: The Complete Guide to Practices and Procedures*. Reading, MA: Addison-Wesley, 2002.

[Oracle, 2008] Oracle Corporation. *Documentation: Oracle10G Database Release 2 Documentation*. http://technet.Oracle.com (accessed October 2008).

[Rob, 2007] Rob, Peter and Carlos Coronel. *Database Systems: Design, Implementation & Management* 7th ed. Boston, MA: Course Technology, 2007. See Chapters 11 and 15.

CHAPTER 22

Distributed Database Systems

As wonderful as database systems are, they would not be delivering on their true potential, if they could not be networked in a distributed environment. This chapter discusses distributed database systems under the following subheadings:

- Introduction
- Advantages of Distributed Database Systems
- Twelve Rules for Distributed Database Systems
- Challenges to Distributed Database Systems
- Database Gateways
- The Future of Distributed Database Systems
- Summary and Concluding Remarks

22.1 Introduction

The concept of a distributed system was introduced in Chapter 2. A distributive database system consists of a collection of sites, connected via a communication network in which:

- Each site is a database system in its own right.
- The sites work together (if necessary) so that a user at any given site can access data at any other site as if the data resides of the host (user's) site.

Section 2.7 of Chapter 2 mentioned some connectivity possibilities. Here are a few noteworthy points to remember:

1. From the definition, the user is given the notion of virtual database systems consisting of data that may reside anywhere in the network.
2. The sites may be distributed over a wide geographical area, or in a local area/building. A distributed database system can therefore be a LAN (local area network), a MAN (metropolitan area network), or a WAN (wide area network).
3. Distributed database systems are not to be confused with remote access systems, sometimes called distributive processing systems. The latter has been around for some time. In such systems, the user accesses data at remote sites but the operation is not seamless; the user is aware and the consequences may be obvious. In a distributed database system, access across sites is seamless.

The literature [on electronic communications and computer networks] documents several alternate approaches to setting up a distributed database. Three prevalent ones are as follows:

- Using client-server technology to set up a federated database
- Setting up a virtual private network (VPN)
- Setting up a data warehouse

There is no shortage of information on client-server technology and VPNs. Exploration of this is beyond the scope of this course; however, references [Martin, 1995] and [Ozsu, 1999] should provide you with a useful start. Suffice it to say that Oracle as described earlier (Chapters 10-16) qualifies as a distributed DBMS. When you install Oracle Server on a node in a network, that node acts as a database server. If you then install Oracle Client on other nodes in the network, your database server can be accessed from anywhere in the network (as well as from other network systems with Web accessibility) in a seamless manner. Also, Chapters 24 and 25 provide overview of data warehousing and Web-accessible databases respectively.

22.2 Advantages of Distributed Database Systems

Figure 22-1 provides some benefits that distributed database systems provide. The benefits may be summarized in three categories:

- Efficiency and Productivity
- Convenience
- Reliability

Efficiency and Productivity:

1. Improved response time and throughput since processing (front-end and back-end as well as at different sites) can be done in parallel.
2. Data (from foreign sites) can be replicated (locally) to improve access time.
3. Easy and fast communication over relatively long distances.
4. Distributive database systems can ensure that the best resources are brought together and utilized in such systems.

Convenience:

5. The systems used (back-ends and front-ends, and at different sites) may be different according to the needs and circumstances of the (users of) respective sites.
6. The structure of the database can mirror the structure of the enterprise — local data is stored locally; access is available to other data over the network.
7. Due to information sharing, the integrated system provides for more data storage, (possibly) increased functionality and features than a single database system.
8. The system is more easily designed to facilitate multiple users (this could lead to improved productivity).

Reliability:

9. Reduction of dependence on a central system (also, this might not be practical).
10. Improved reliability: if one machine fails, the whole system does not fail (there is no reliance on a central system).

Figure 22-1. *Benefits of Distributed Database Systems*

When weighed against the challenges posed by distributed database (discussed later), they outweigh them by a huge margin; we therefore expect the continued proliferation of distributed databases.

22.3 Twelve Rules for Distributed Database Systems

In his classic text, *Introduction to Database Systems*, Christopher Date discusses twelve rules (objectives) for distributed database systems [Date, 2004]. Let us take a brief look at these rules.

Rule 1: Local Autonomy

The sites should be autonomous to the maximum possible extent. All operations at a site are governed by that site alone. This is not always entirely possible, but is an objective to strive for.

Rule 2: No Reliance on Central Site

This is a consequence of objective 1. The sites must be treated as equals. There is no reliance on a central site. Reliance on a central site would make the system vulnerable to the central site (bottleneck could occur or the central site could go down). This is undesirable.

Rule 3: Continuous Operation

The system must be able to run continuously. There should be no need for a planned shut down in order to carry out any function (for instance backup or tuning).

Rule 4: Location Independence (Transparency)

Users should not need to know where data is physically stored in order to access it; the system should operate as if all the data resided at the local site. The distributed nature of the database must be transparent to the end end users.

Rule 5: Fragmentation Independence

The system should support data fragmentation — it should be possible to partition a given relation into fragments that are stored at different sites. Thus, data can be stored where it is most frequently used. Network trafficking is therefore reduced. Fragmentation should be transparent to the end users.

Example 1: Suppose that a large organization has employee records in an **Employee** relation. The departments are at different localities. Records for each department are stored at different sites in those respective departments. This can be easily facilitated in Oracle or DB2 by *partitioning* the **Employee** table. A full discussion of table partitioning is beyond the scope of this course. However, suffice it to say that that this is the technique used by several leading DBMS suites to facilitate fragmentation independence.

Rule 6: Replication Independence

A given relation (or fragment of a relation) can be replicated at different sites. Replication may improve access time and hence performance. The drawback however, is that at update, all copies have to be updated simultaneously. Replication should be transparent to the end users.

Rule 7: Distributed Query Processing

Distributed query processing must be facilitated among different sites. Records are transmitted set (relation) at a time instead of record at a time.

Example 2: Suppose we have an international company, IBM, say, where employees are stored in the relation **Employee**, fragmented in various countries, where there are IBM offices. IBM Canada, issues the request: "Find all Jamaican male employees." Then:

a. Suppose there are n records that satisfy this request. If the system is relational, the query will involve tw o messages one from Canada to Jamaica and one from Jamaica to Canada. If the system is not relational, but record-at-a-time, the query will involve $2n$ messages — n from Canada to Jamaica and n from Jamaica to Canada.

b. Query optimization of the request occurs before execution (record-at-a -time requests cannot be optimized).

Due to these two points, distributive database systems are usually relational.

Rule 8: Distributed Transaction Management

Each transaction must be atomic — fully committed or fully rolled back. This objective must be met irrespective of the agents (constituent processes) of the transaction. Concurrency control must be ensured (usually by data locking).

Rule 9: Hardware Independence

It should be possible to integrate the system across different hardware platforms. It should therefore be possible to run the DBMS on different hardware systems.

Rule 10: Operating System Independence

It should be possible to integrate the system across different operating system platforms. It should therefore be possible to run the DBMS on different operating systems.

Rule 11: Network Independence

The system should be able to support different sites with different hardware and different operating systems and networking protocols.

Rule 12: DBMS Independence

The DBMS suites used may be different. For instance DB2 and Oracle both support SQL and open database connectivity (ODBC); it should therefore be possible to link databases running on the two DBMS suites. The same argument should apply for other DBMS suites.

22.4 Challenges to Distributed Database Systems

Distributed databases did not come easily; neither were they easy to maintain. Fortunately, the software engineering industry has figured out how to address these challenges. However, improving the algorithms used, and finding new ones are always topical research issues. There are five well known challenges to distributed database systems. These are:

- Query Optimization
- Catalog Management
- Update Propagation
- Concurrency Control
- Transaction Management

Query Optimization

Query optimizing processing must be distributed in order to minimize network trafficking. To illustrate, consider a query Qa of site A, accessing to relations in a natural join: Rb of site B and Rc of site C. The optimizer must decide on one of the following strategies:

a. Move copies of Rb and Rc to site A

b. Move copy of Rb to site C and process the join there

c. Move copy of Rc to site B and process the join there

The optimizer must be able to calculate what would be most economical alternative (given the structure and configuration of the underlying network) and choose that alternative. For example, Oracle implements two query optimization strategies — a rule-based optimization and a cost-based optimization. Before executing a query, the query optimizer optimizes the query by converting it to an internal format (based on an Oracle algorithm) that will ensure the most efficient execution.

Catalog Management

Catalog management is one of the most complex issues that a distributed database must resolve. This is so since additional information must be stored for the database objects (e.g. fragmentation, replication, location etc.). Where and how the catalog should be stored is a complicated issue. Below are some alternatives:

- **Centralized:** The catalog is stored at a centralized location, and is accessible to the other participating sites.
- **Fully Replicated:** The catalog is replicated at each participating site.
- **Partitioned:** Each site maintains its own catalog. The total catalog is the union of each site catalog.
- **Hybrid:** Each site maintains its own catalog; additionally, a central site maintains the global catalog.

Each of these approaches has its related advantages and challenges. Resolving this issue is often done with the use of simulation software, and much research into the matter.

Update Propagation

In the case where data is replicated at different sites, it may not be possible to effect update to all replicas at the desired time. How is this resolved? The *primary copy* approach is a common method of resolution:

- One replica is deemed the primary copy. As soon as that copy is updated, the update process is deemed completed.
- The site with the primary copy is responsible for updating the other sites as son as possible.

This somewhat contradicts the objectives of transaction independence and redundancy control. As was emphasized in chapter 4, once data replication is introduced in a database, with it come various other data integrity problems. Resolution is therefore a matter of tradeoff.

Concurrency

Concurrency is another issue that must be resolved. To illustrate, consider what might happen if user X tries to retrieve a particular data set for update purposes, but that data set is being updated by another user Y. Consider that in a distributed system, there might be thousands of users, so that this kind of contention could easily develop among several users. Typically, the DBMS handles this problem by *record locking*: a record or data set that is retrieved for update is *locked* to that transaction until the update is completed; it is then released (unlocked) for other users. Requests to release (*unlock*) objects in a distributed database system must be managed. This is a serious overhead.

In application development where distributed databases are accessed, or there is multi-user access of a single database, the application programmer must check for record lock on a data set before attempting to lock that data set. The application programmer must program a graceful recovery from a record lock situation (normally done by issuing an appropriate message to the end user and allowing them to defer that particular request until some subsequent time).

Transaction Management

Issues such as when to lock records, and when to commit or rollback transactions are critical in a distributed database. The application developer must be familiar with the facilities provided by the DBMS and SQL (COMMIT and ROLLBACK) for managing transactions.

22.5 Database Gateways

Traditionally, a database gateway is a software component that links two different DBMS suites. It could run on either of the two systems running the dissimilar DBMS suites, or on a separate machine for that purpose. The simplest configuration is to run the software on either system as a driver for the other DBMS. Another alternative is to use the software called Open Database Connectivity (ODBC). ODBC is marketed with the Windows operating system, and is readily available for other operating systems.

Suppose for instance, that Oracle and DB2 both support SQL (as in fact, they do). It should therefore be possible to link the two DBMS suites, as illustrated in Figure 22-2. In reality, each product includes an ODBC driver that in effect acts as the gateway. Following are some functions of the gateway:

- Mapping between the two different protocols (formats)
- Mapping between the two different dialects of SQL (e.g. Oracle's dialect versus DB2's dialect)

- Mapping between the two different system catalogs so that users of one DBMS (Oracle) can access all files from the other DBMS (DB2)
- Mapping between the two different sets of data used
- Resolution of semantic mismatches across the two systems (e.g. attribute **Employee.Empno** in a DB2 table may map to **Empl.Emp#** in an Oracle table)
- Resolution of transaction management issues such as data locking, updates, commit and rollback
- Resolution of security and accessibility issues across the two different systems

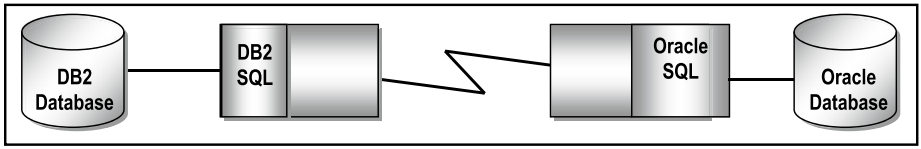

Figure 22-2. *Illustrating the Function of a Database Gateway*

By way of example, Oracle operates a very sophisticated gateway system that allows connectivity to other non-Oracle databases either directly through its gateway, or indirectly through ODBC (see [Oracle, 2008]). Among the many features of the Oracle gateway system are the following:

- Location independence and transparency
- Data-type translations between Oracle and non-Oracle systems
- Data dictionary between Oracle and non-Oracle systems
- Read/write access
- Support for large objects (LOBs)
- Support for non-Oracle stored procedures
- Transmission of pass-through SQL between Oracle and non-Oracle system
- Data encryption services

22.6 The Future of Distributed Database Systems

Since the mid-1990's, two significant technologies have significantly influenced the development and direction of distributed databases. These are:

- Object Technology (OT)
- Electronic Communication Systems (ECS)

22.6.1 Object Technology

In a few short years, OT has come to dominate the contemporary software engineering industry. New products are forced by industry demands to support OT in some form. Much work has been done by the Object Management Group (OMG) in establishing the Common Object Request Broker Architecture (CORBA) standards for distributed database systems. The Microsoft equivalents of CORBA are Component Object Model (COM), Distributed COM (DCOM), and more recently, the .NET framework.

CORBA standards span a wide range of specifications from user interface to object-object communication. They are supported by some of the leading software engineering firms in the industry. With the emergence of Java, the software industry has made significant progress in the area of platform independent software components than ever before. As mentioned in chapter 6, Java (through JDBC) supports both CORBA and ODBC.

22.6.2 Electronic Communication Systems

Complimentary to the advances in OT, the past decade has seen much achievement in the arena of *electronic communication systems* (ECS). Contemporary operating systems are more sophisticated, supporting a wider range of communication protocols. The protocols and their underlying technology have been refined to provide much higher transmission rates. Also, communication protocols provide much more services than previously.

With a refinement of, and emphasis on standards, interoperability is now a much more attainable goal than in the previous decade. These advances, when combined with those in OT, will contribute to the proliferation of heterogeneous information models.

22.7 Summary and Concluding Remarks

Let us summarize what we have covered in this chapter:

- A distributed database systems is a conglomeration of database systems in which each system operates as an autonomous system on its own, or collaborates with other systems as required.
- Distributed databases systems provide a number of benefits in the areas of efficiency and productivity, convenience, and reliability.
- Distributed database systems should strive to conform to the standards outlined in Date's twelve rules for such systems.
- Distributed databases face challenges in the areas of query optimization, catalog management, update propagation, concurrency control and transaction management.
- A database gateway is a software component that links two different DBMS suites.
- Distributed databases have been significantly affected by developments in object technology and electronic communications technology. This is expected to continue in the foreseeable future.

Distributed databases have helped to transform our world in a significant way. To fully appreciate the power of distributed databases, just consider for a moment, a world without them: no World Wide Web; minimized reliability on critical company databases; reduced capabilities on operating systems; limited remote access to databases. The next chapter takes a closer look at object databases.

22.8 Review Questions

1. Define a distributed database system. Discuss the advantages of such systems.
2. Outline and clarify the twelve rules for distributed database systems.
3. Discuss the challenges to distributed database system.
4. As an IT professional (perhaps in training), what are your future expectations for distributed database systems?

22.9 References and/or Recommended Readings

[Chung, 1997] Chung, P. Emerald, Yennun Huang, Shalini Yajnik, Deron Liang, Joanne C. Shih, Chung-Yih Wang, and Yi-Min Wang. *DCOM and CORBA Side by Side, Step by Step and Layer by Layer*. http://research.microsoft.com/~ymwang/papers/HTML/DCOMnCORBA/S.html (accessed October 2008).

[Date, 2004] Date, Christopher J. *Introduction to Database Systems* 8th ed. Menlo Park, CA: Addison-Wesley, 2004. See Chapter 21.

[Elmasri, 2007] Elmasri, Ramez and Shamkant B. Navathe. *Fundamentals of Database Systems* 5th ed. Reading, MA: Addison-Wesley, 2007. See Chapter 25.

[Kifer, 2005] Kifer, Michael, Arthur Bernstein, and Philip M. Lewis. *Database Systems: An Application-Oriented Approach* 2nd ed. New York, NY: Addison-Wesley, 2005. See Chapter 16.

[Martin, 1995] Martin, James, and Joe Leben. *Client/Server Databases: Enterprise Computing*. Upper Saddle River, NJ: Prentice Hall, 1995. See Chapters 7-9.

[Oracle, 2008] Oracle Corporation. *Oracle Database Gateways*. http://www.oracle.com/technology/products/gateways/index.html (accessed October 2008).

[Ozsu, 1999] Ozsu, M. Tamer, and Patrick Velduriez. *Principles of Distributed Database Systems*. 2nd ed. Eaglewood, NJ: Prentice Hall, 1999.

[Rob, 2007] Rob, Peter and Carlos Coronel. *Database Systems: Design, Implementation & Management* 7th ed. Boston, MA: Course Technology, 2007. See Chapter 12.

[Silberschatz, 2006] Silberschatz, Abraham, Henry Korth, and S. Sudarshan. *Database System Concepts* 5th ed. Boston, MA: McGraw-Hill, 2006. See Chapter 22.

CHAPTER 23

Object Databases

The previous decade has witnessed the advancement of several so-called object database management systems (ODBMS) and universal database management systems (UDBMS). This Chapter discusses such systems under the following subheadings:

- Introduction
- Overview of Object Oriented Database Management Systems
- Challenges to Object Oriented Database Management Systems
- Hybrid Approaches
- Summary and Concluding Remarks

23.1 Introduction

Object Technology (OT) has been dominating the software engineering industry in recent times. For better or worse, there has been a heightened interest and indulgence in object-oriented methodologies (OOM). Full treatment of OT and OOM is better handled in another course on the subject. However, for completeness, a brief introduction is made here.

OT provides obvious advantages to application programming, with benefits of encapsulation, polymorphism and complexity (information) hiding, code reusability, etc (Figure 23-1 summarizes the commonly mentioned advantages). By contrast, an OO approach to database design may or may not bring significant benefits, depending on the situation. Data structure encapsulation may or may not be desirable; besides, the principle of encapsulation often contradicts the principle of data independence in database design.

1. **Reusability of Code:** A tested system component can be reused in the design of another component.
2. **Stability and Reliability:** Software can be constructed from tested components. Organizations can be assured of guaranteed performance of software.
3. More complex systems can be constructed.
4. **Understandability:** Designer and user think in terms of object and behavior rather than low-level functional details. This results in more realistic modeling that is easier to learn and communicate.
5. **Faster Design:** Most RAD tools and contemporary CASE tools are object oriented to some degree. Also code reusability enhances faster development.
6. **Higher Quality Design:** New software can be constructed by using tested and proven components.
7. **Easier Maintenance:** Since systems are broken down into manageable component objects, isolation of system faults is easy.
8. **Dynamic Lifecycle:** I-OO-CASE tools integrate all stages of the software development life cycle (SDLC).
9. **Interoperability:** Generic classes may be designed for multiple systems.
10. **Design Independence:** Classes may be designed to operate and/or communicate across different platforms.
11. **Clarity:** OT promotes better communication between IS professionals and business people.
12. **Better CASE Tools:** OT leads to the development of more sophisticated and flexible CASE and RAD tools.
13. **Better Machine Performance:** OT leads to more efficient use of machine resources.

Figure 23-1. *Benefits of Object Technology*

It is widely believed that an OO approach to database design and implementation is preferable for complex applications such as:

- CAD/CAM systems
- CASE tools
- Geographic information systems (GIS)
- Document Storage and retrieval systems
- Artificial intelligence (AI) systems and expert systems

Which approach is the better? This is a difficult question to answer. It depends on the situation at hand, but the following is a summary of the alternatives:

- Truly function oriented systems with relational database and procedural application development
- Truly OO systems involving (encapsulation of) both data structure and operation
- Hybrid approach A with relational database and OO user interface
- Hybrid approach B with object/relational database and OO user interface

The first approach is a well-known traditional approach, and will not be discussed further. The latter three approaches are more aligned to current practices in the software engineering industry. They will be discussed in the upcoming sections.

23.2 Overview of Object-Oriented Database Management Systems

In a truly object-oriented DBMS (OO DBMS), the following concepts hold (for more details, see the references):

- An object type is a concept or thing about which data is to be stored, and a set of operations is to be defined.
- An object is an instance of an object type.
- An operation is a set of actions to be performed on an object.
- A method specifies the way an operation is to be performed.
- Encapsulation is the packaging of data structure and operations, typically into a class. The internal structure of the class is hidden from the outside, and only members (member functions) of the class have access to it.
- A subclass may inherit properties (structure or operations from a subclass. Also a class may be comprised of several component classes).
- Polymorphism is the phenomenon where a given object or operation may take on a different form, depending on the context of usage.
- Objects communicate by sending messages to each other. These messages are managed via events; an object therefore responds to events.

23.3 Challenges for Object-Oriented Database Management Systems

A number of challenges stand in the way of achieving purely object-oriented DBMSs, some of which have been articulated in [Date, 2004]. Figure 23-2 provides a brief summary of these challenges.

1. OT as proposed by its ardent proponents, encourages generalization and the building of inheritance hierarchies where objects consist of other objects (as in subtype, component and aggregation relationships. If pursued to their logical conclusion without pragmatic deviations (as discussed in chapter 3), the system could break down into a convoluted mess. Another related problem is that hierarchies, as we know, do not lend themselves to representation of M:M relationships. Consider, for instance, an M:M relationship between **Academic Program** and **Course**. How do we model and implement this in the OO paradigm? Do we assume that academic programs contain courses or vice versa?. History has shown that hierarchical based systems (such as CODASYL) are unsuited for complex distributed databases. For this reason, purely OO databases have been criticized as "CODASYL warmed over."

2. The OO model does not encourage the introduction of keys to uniquely identify objects, since object identifiers (OIDs) are generated (as internal address) at the point of instantiation. Note however, that OIDs do not obliterate the need for user keys, since end users still need to have a way of identifying objects separate and apart from internal addresses. The problem is further compounded when we have intermediate transient objects, or what in relational terminology is referred to as logical (virtual) objects such as a natural join record. In the OO model, there is no way to negotiate these situations without duplicating data into separate storage variables which must be maintained by the running program, thus increasing system overheads.

   The relational model gracefully handles these situations by allowing the user to define keys and surrogates (note that surrogates are not identical to OIDs). Then thanks to data independence, you can define multiple logical views on physical data. It has been argued that encapsulation is to the OO model what data independence is to the relational model. However, when analyzed, encapsulation is not a perfect replacement for data independence.

3. The OO model does not promote data sharing, a fundamental tenet of the relational model. At best, we could have encapsulated objects sharing their data contents with other objects, but in a distributed environment database (or mere network with one database server and several users), this would significantly increase the system overheads.

4. The matters of class, instance and collection are particularly difficult to negotiate. In the OO paradigm as we know, a class is essentially a complex data type; an instance is a specific object which belongs to at least one class. Without intense programmatic intervention, it would be difficult to determine a collection of objects belonging to a given class.

   The relational model has no such problem. A relational table defines the type and contains the collection of related data (objects) all in the same place. And as you are aware, this data can be shared and used to construct any number of logical perspectives as required by external end users.

5. The perfect encapsulation of data structure and operation in a data model is still for the most part, an ideal. We have seen complete achievement of this objective in the programming domain, but seldom in the database domain.

Figure 23-2. *Challenges to Object-Oriented Database Management Systems*

23.4 Hybrid Approaches

Due to the above-mentioned challenges, it is unlikely that we will see a proliferation of purely object- oriented DBMS suites involving only encapsulated database objects in the immediate future. The benefits of OT are therefore likely to continue to be more significant in the area of user interface than database (of course with a few exceptions as mentioned earlier). The relational model on the other hand, has long proven its worth. The best we can therefore expect is a peaceful coexistence of both OO systems and relational databases — a kind of hybrid. The next two subsections describe two hybrid approaches.

23.4.1 Hybrid Approach A

In the Hybrid Approach A, a relational database is accessed by an OO user interface. This means that the application development is done via an OO Programming Language (OOPL), CASE tool, or RAD tool.

There is no shortage of OOPL (some of them pure OOPLs, others are hybrid OOPLs). The more popular ones include C++, Java, C#, and Object Pascal. There are many object-oriented CASE (OO-CASE) tools and RAD tools that support this approach. They include (but are not confined to) products such as Team Developer, Delphi, Oracle JDeveloper, WebSphere, NetBeans etc.

Three advantages of this approach are as follows:

1. It can facilitate *legacy systems* (software systems based on old technology), which are prevalent in large organizations.
2. It facilitates peaceful coexistence of traditional and more contemporary system approaches.
3. It reaps the benefits of a relational database and OO application development.

The main disadvantage of the approach is that it does not address the earlier mentioned situations that warrant OO database design. However, the skillful database designer can use techniques mentioned in Chapter 3 for implementing relationships such as subtype, component and aggregation.

23.4.2 Hybrid Approach B

In this approach, a relational/object database (often referred to as a universal database) is accessed via an OO user interface. Like the Hybrid Approach A, application development is done via an OOPL, OO-CASE tool, or RAD tool.

The universal database supports both the relational database as well as the OO database. The designer can therefore make critical decisions as to which approach is preferred, given the scenario. The skillful database designer also has the flexibility of employing techniques mentioned in Chapter 3 for implementing relationships such as sub-type, component and aggregation.

Several of the leading software engineering companies have in recent times, introduced products that support object databases. Examples include:

- IBM's DB2 Universal
- Oracle Universal Database Server
- Informix's Universal Data Option for its Dynamic Server

Advantages of the approach are as follows:

1. The strengths of the relational model and the OO model can be emphasized, and the respective weak points avoided.
2. Legacy systems can be facilitated.
3. It facilitates peaceful coexistence of traditional and more contemporary system approaches.

One possible disadvantage is that the database designer could be sometimes forced to make difficult decisions on database design.

23.5 Summary and Concluding Remarks

It is time to summarize what we have discussed in this Chapter:

- OT provides several huge benefits to the software engineering arena.
- An OO approach to database design may or may not bring significant benefits, depending on the situation. In some situations the OO approach is ideal; in others it is not.
- There are a number of challenges to object databases that do not exist in a relational database.
- The most pragmatic approach for merging the benefits of the relational model and the OO model is to superimpose an OO user interface on a relational or universal database.

The strength of the relational model lies in its firm mathematical foundations, its huge benefits, and the immense financial outlay that have been placed in relational systems. This third fact is significant: Several large corporations have invested millions of dollars hugely in relational databases. They are not likely to discard these investments. The strength of the OO model lies in its huge benefits in certain situations, and its intuitiveness. Given the strength of both models, we can expect them to continue to peacefully coexist in the foreseeable future, complementing instead of rivaling each other.

23.6 Review Questions

1. Identify the main benefits that object technology brings to the arena of database systems.
2. Describe the main features of the OO DBMS model.
3. Discuss the main challenges to the object-oriented database management systems.
4. Discuss the hybrid approach to database systems.

23.7 References and/or Recommended Readings

[Date, 2004] Date, Christopher J. *Introduction to Database Systems* 8th ed. Menlo Park, CA: Addison-Wesley, 2004. See Chapters 25–26.

[Elmasri, 2007] Elmasri, Ramez and Shamkant B. Navathe. *Fundamentals of Database Systems* 5th ed. Reading, MA: Addison-Wesley, 2007. See Chapters 20–22.

[Kifer, 2005] Kifer, Michael, Arthur Bernstein and Philip M. Lewis. Database Systems: An Application-Oriented Approach 2nd ed. New York, NY: Addison-Wesley, 2005. See Chapter 14.

[Lee, 2002] Lee, Richard C. and William M. Tepfenhart. *Practical Object-Oriented Development With UML and Java*. Upper Saddle River, NJ: Prentice Hall, 2002.

[Martin, 1993] Martin, James and James Odell. *Principles of Object Oriented Analysis and Design*. Englewood Cliffs, NJ: Prentice Hall, 1993.

[Rumbaugh, 1991] Rumbaugh, James, et. al. *Object Oriented Modeling And Design*. Eaglewood Cliffs, NJ: Pretence Hall, 1991.

[Silberschatz, 2006] Silberschatz, Abraham, Henry Korth, and S. Sudarshan. *Database System Concepts* 5th ed. Boston, MA: McGraw-Hill, 2006. See Chapter 9.

CHAPTER 24

Data Warehousing

Since the mid1990s, database technology has expanded into a new area of interest — the development and management of data warehouses. It is a fascinating field of study that deserves some attention. This chapter provides an overview of data warehousing and information extraction. The chapter proceeds under the following subheadings:

- Introduction
- Rationale For Data Warehousing
- Characteristics of a Data Warehouse
- Data Warehouse Architectures
- Extraction, Transformation and Loading
- Summary and Concluding Remarks

24.1 Introduction

The concept of a data warehouse springs from the combination of two sets of needs:

- The business requirement for a global view of information, independent of and despite its source or underlying structure
- The need of information systems (IS) professionals to manage large volumes of company data in a more effective manner

The data warehouse has been approached many times and from many directions in the last decade; many implementations exist today. In order to proceed, we must make a distinction between data and information:

- Data is the computerized representation of business information.
- Information is the assimilation of data to convey meaning as understood and used by end users.

A data warehouse is an *integrated, subject-oriented, time-variant, nonvolatile,* consistent database, constructed from multiple sources, and made available (in the form of read-only access) to support decision making in a business context. As you will soon see (in section 24.3), the highlighted terms in this definition are deliberate because of their significance.

Here is an alternate definition: A data warehouse is a relational database that is typically constructed from multiple transactional databases (called *source databases*), and designed for query and analysis rather than transaction processing. The data warehouse usually contains historical data that is derived from transaction data from multiple sources. It separates analysis workload from transaction workload, and enables a business to consolidate data from several sources.

In addition to a relational database, a data warehouse environment often consists of an ETL (extract, transformation, and load) solution, an OLAP (on-line analytical processing) engine, client analysis tools, and other applications that manage the process of gathering data and delivering it to business users. End users typically require some kind of catalog that describes the data in its business context, and acts as a guide to the location and use of this information. Finally, end users require a set of tools to analyze and manipulate the information thus made available.

As you are no doubt aware by now, achieving comprehensiveness and consistency of data in today's business environment is often a complex and challenging undertaking. This is also true for a data warehouse. The following steps are necessary for the construction of a data warehouse:

1. Conduct an information infrastructure analysis to determine the required structure of the data warehouse.
2. Identify the source databases that will feed the data warehouse.
3. Design the integrated logical data model and determine the architecture of the data warehouse.
4. Develop and implement a comprehensive meta-data methodology.
5. Determine, and then implement the physical structure of the data warehouse.
6. Design and implement an integrated staging area for the data warehouse.
7. Extract, transform and load the data (from various sources) into the data warehouse. This involves first cleansing the source data of various structural and content errors.
8. Conduct comprehensive post-implementation review(s) to ensure that the data warehouse is performing acceptably.
9. Maintain the data warehouse.

Data mining is the act of extracting data/information from assorted sources and presenting information in a manner that is consistent with user requirement. Data mining often implies the existence of data warehouses, so the two terms are closely related. Another related term is *information extraction* (IE) —the extraction of structured information from unstructured text. IE sometimes involve access of data warehouse(s) either as the source or destination of information.

24.2 Rationale for Data Warehousing

Data warehousing is a technology that is fast enhancing more traditional decision support systems (DSS), because of the added flexibility and benefits that the new technology brings. Let us briefly examine the problems that DSS end users tend to have from two perspectives — user constraints and information system (IS) constraints.

User Constraints: In the absence of data warehousing, users of traditional decision support systems developed using the traditional application-driven approach, commonly complained of the following difficulties:

- Difficulty in finding and accessing information needed
- Difficulty in understanding information found
- Information obtained is not as useful as expected

IS Constraints: In the absence of data warehousing, IS personnel also complained of a variety of problems:

- Developing copy programs is often very challenging
- Maintaining copy programs and copy databases presents serious integrity and work scheduling problems
- Data storage volumes tend to grow rapidly
- Database administration also tends to become quite complex

The solution to the above-mentioned problems is the implementation of a data warehouse. A data warehouse provides the decision support benefits that a traditional DSS provides, while providing more flexibility for expansion beyond the confines of the company. This is so for two reasons:

- The data warehouse has the capacity to attract interest in the salient facts about the organization, without providing unnecessary details.
- While companies may be hesitant about putting their transaction database(s) into the public domain (due to security and confidentiality concerns), they are more likely to be willing to put their data warehouse into the public domain (via the World Wide Web).

24.3 Characteristics of a Data Warehouse

In the definition of a data warehouse, a number of terms were deliberately highlighted. These terms convey important characteristics about a data warehouse. These will be clarified in the next subsection. Next, we will examine what kind of data that is typically stored in a data warehouse. We also examine the processing requirements of a data warehouse. Finally, we will review twelve rules that govern data warehouses.

24.3.1 Definitive Features

In the introduction, it was established that a data warehouse is an integrated, subject-oriented, time-variant, nonvolatile database. Let us briefly examine what these adjectives mean.

Subject-Oriented: Data warehouses are designed to aid the analysis of data in order to make decisions. For example, to learn more about your company's sales data, you can build a warehouse that concentrates on sales. Using this warehouse, you can answer questions like "Who was our best customer for this item last year?" This ability to define a data warehouse by subject matter, sales in this case, makes the data warehouse subject-oriented.

Integrated: Integration is closely related to subject orientation. Data warehouses typically contain data from disparate sources into a consistent format. They must resolve such problems as naming conflicts and inconsistencies among units of measure. When a data warehouse achieves this, it is said to be integrated.

Nonvolatile: Nonvolatile means that, once entered into the warehouse, data should not change. This is logical because the purpose of a warehouse is to enable you to analyze historical data.

Time-Variant: A data warehouse's focus may change over time; also, it could grow (in terms of data volume, data structure and complexity).

24.3.2 Nature of Data Stored

Of importance also, is the nature of data stored in a data warehouse. A data warehouse differs from an operational database in the nature of data stored. An operational database consists of a set of normalized relational tables that store atomic data. A data warehouse on the other hand, stores decision support data, often in non-normalized, aggregated formats. Three distinctions can be made between operational data and decision support data:

Time Span: Operational data represent atomic transactions at specific points in time. Decision support data represent (aggregated) data over a period of time.

Granularity: Operational data is atomic; decision support data is often aggregated. Data warehouses contain decision support data that have been aggregated from various sources and transformed to its intended format.

Dimension: Operational data is instamatic; decision support data is multi-dimensional, typically involving the dimension of time as well as other factors of concern.

24.3.3 Processing Requirements

Data warehouses have very different processing requirements from OLTP systems and operational databases, as explained below.

Workload: Data warehouses are designed to accommodate ad hoc queries. The workload of the data warehouse might not be known in advance, so a data warehouse should be optimized to perform well for a wide variety of possible query operations. OLTP systems and operational databases support only predefined operations. Your applications might be specifically tuned or designed to support only these operations.

Data Modifications: A data warehouse is updated on a regular basis by the ETL process (run nightly or weekly) using bulk data modification techniques. The end users of a data warehouse do not directly update the data warehouse. In OLTP systems and operational databases, end users routinely issue individual data modification statements to the database. The OLTP database is always up to date, and reflects the current state of each business transaction.

Schema Design: Data warehouses often use non-normalized or partially normalized schemas (such as a *star schema*) to optimize query performance. OLTP systems and operational databases often use fully normalized schemas to optimize update/insert/delete performance, and to guarantee data consistency.

The star schema was introduced in Chapter 5 (section 5.6); it is so widely mentioned in database literature, it deserves a bit of attention: A star schema describes a mechanism where there is a "central" table referred to as a *fact table*, and other so-called *dimensional tables* that relate to the fact table via 1:M relationships. The dimensional tables contain dimensional data about details stored in the fact table. While the star schema is widely used in data warehouses, it is also applicable in operational databases as well. Figure 24-1 provides an illustration of a star schema of five relational tables for tracking the graduation statistics from a regional university that operates multiple schools and programs in multiple locations. Tables **Location**, **AcademicProgram**, **TimePeriod**, and **School** qualify as dimensional tables, while table **GraduationSummary** qualifies as the fact table.

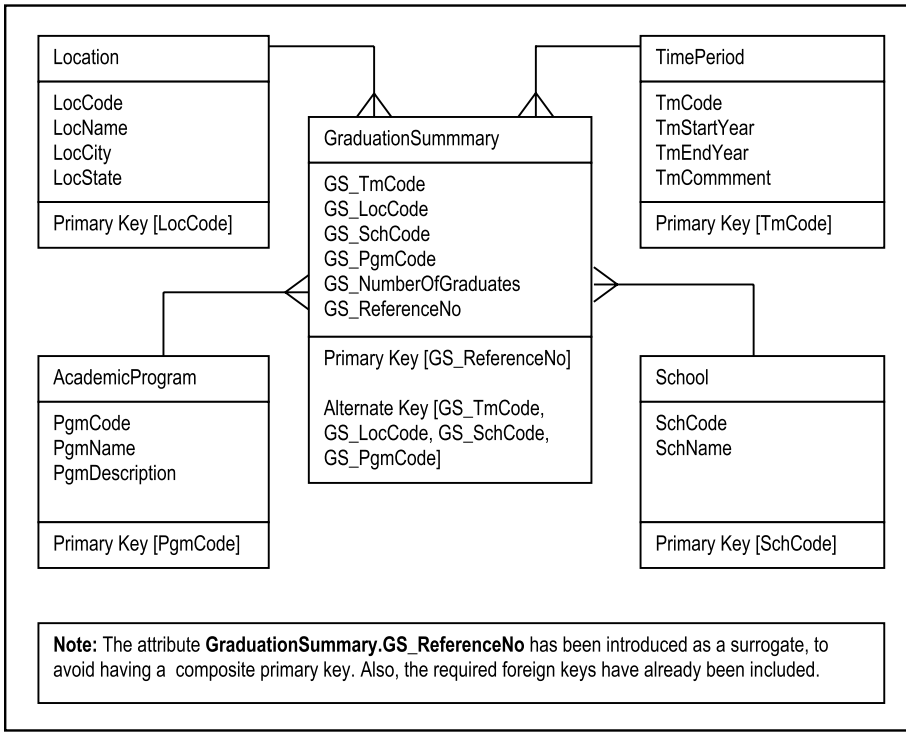

Figure 24-1. *Illustration of a Star Schema for Graduations from a Regional University*

Typical Operations: A typical data warehouse query may scan thousands or millions of rows of data. For example, a data warehouse query may involve determining the total sales for all customers in a specific time period. Except for sophisticated queries, a typical OLTP or operational database operation accesses only a small percentage of records, relative to the amount stored. For example, an operational query may involve retrieving the current purchase order for a particular customer, from a table storing hundreds of thousands of purchase orders.

Figure 24-2 provides a summary of what we have established so far: that a data warehouse is typically constructed for various operational databases (sources), and possesses certain characteristics.

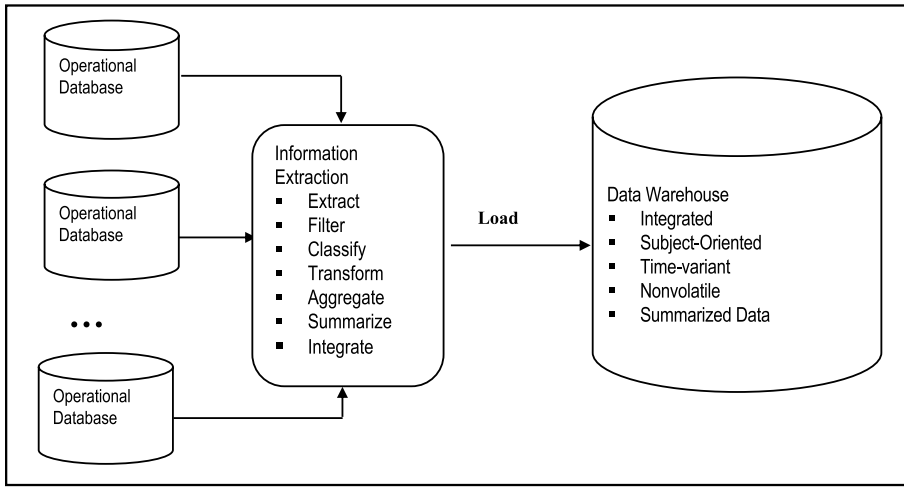

Figure 24-2. *Constructing a Data Warehouse*

24.3.4 Twelve Rules That Govern a Data Warehousing

Data warehouse was first introduced by William Inmon in the 1990s; he still operates as an expert in the field (see [Inmon, 2002] and [Inmon, 2007]). Like E.F. Codd and C. J. Date of the relational model, Inmon introduced twelve rules for governing data warehouses. These rules aptly summarize the previously mentioned characteristics of the data warehouse. Many of these rules have been subsumed in the foregoing discussions. Nonetheless, for emphasis, they are paraphrased below:

1. **Separation:** The data warehouse and operational database environments should be separated.
2. **Integration:** The data warehouse data are integrated from various operational sources.
3. **Time Horizon:** The data warehouse typically contains historical data with an extended time horizon. This is in contrast to a significantly shorter time horizon for the operational databases that may be used as sources for the data warehouse.
4. **Nature of Data:** The data in a data warehouse represent snapshot captures (from operational data sources) at specific points in time.
5. **Orientation:** The data contained in the warehouse are subject-oriented.

6. **Accessibility:** The data warehouse is a predominantly read-only database, with periodic batch updates from the operational databases that are connected to it. It does not support on-line interactive updates.
7. **Life Cycle:** The data warehouse development life cycle differs from classical approach to database development in that whereas the data warehouse development is data-driven, the classical database approach tends to be process-driven.
8. **Levels of Detail:** In a typical data warehouse, there may be several levels of detail. These include current detail, old detail, lightly summarized data, and highly summarized data.
9. **Data Set:** The data warehouse is characterized by read-only transactions on very large data sets. This is in contrast to the operational database, which is characterized by numerous update transactions on a more narrowly defined data set.
10. **Relevance:** The data warehouse environment has a system that keeps track of all data sources, transformations and storage. This is essential if the data warehouse is to maintain its relevance.
11. **Metadata:** The data warehouse's metadata forms a critical component of its environment. The metadata provides the following functions: definitions of data elements in the warehouse; identification of data source, transformation, integration, storage, usage, relationships, and history of each data element.
12. **Resource Usage:** The data warehouse typically enforces optimal usage of the data by enforcing some form of chargeback mechanism for resource usage.

24.4 Data Warehouse Architecture

The architecture of a data warehouse varies depending upon the specifics of an organization's situation. Three common architectures have been identified:

- Basic Data Warehouse
- Data Warehouse With a Staging Area
- Data Warehouse With Staging Area and Data Marts

24.4.1 Basic Data Warehouse Architecture

Figure 24-3 shows a simple architecture for a data warehouse. End users directly access data derived from several source systems through the data warehouse.

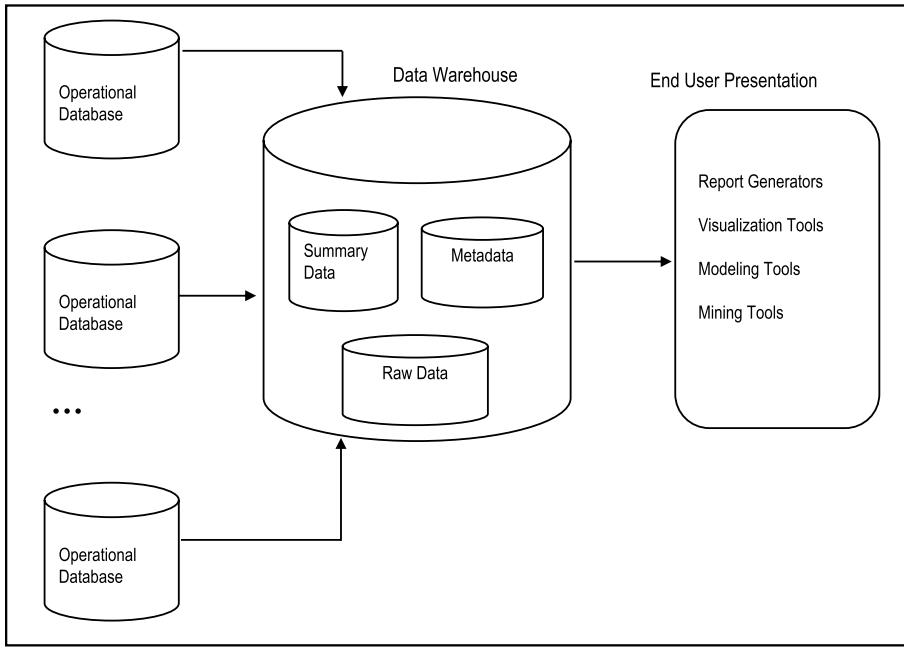

Figure 24-3. *Basic Data Warehouse*

The data warehouse consists of raw data (from operational databases), summary data and metadata. Summaries are very valuable in data warehouses because they pre-compute long operations in advance. For example, a typical data warehouse query may be to retrieve something like aggregate sales for a specific period. In Oracle, this may be implemented as a materialized view or snapshot relation set up for that purpose.

24.4.2 Data Warehouse Architecture with a Staging Area

In the basic data warehouse, you need to clean and process your operational data before putting it into the warehouse. You can do this programmatically, although most data warehouses use a staging area instead. A staging area simplifies building summaries and general warehouse management. Figure 24-4 illustrates this typical architecture.

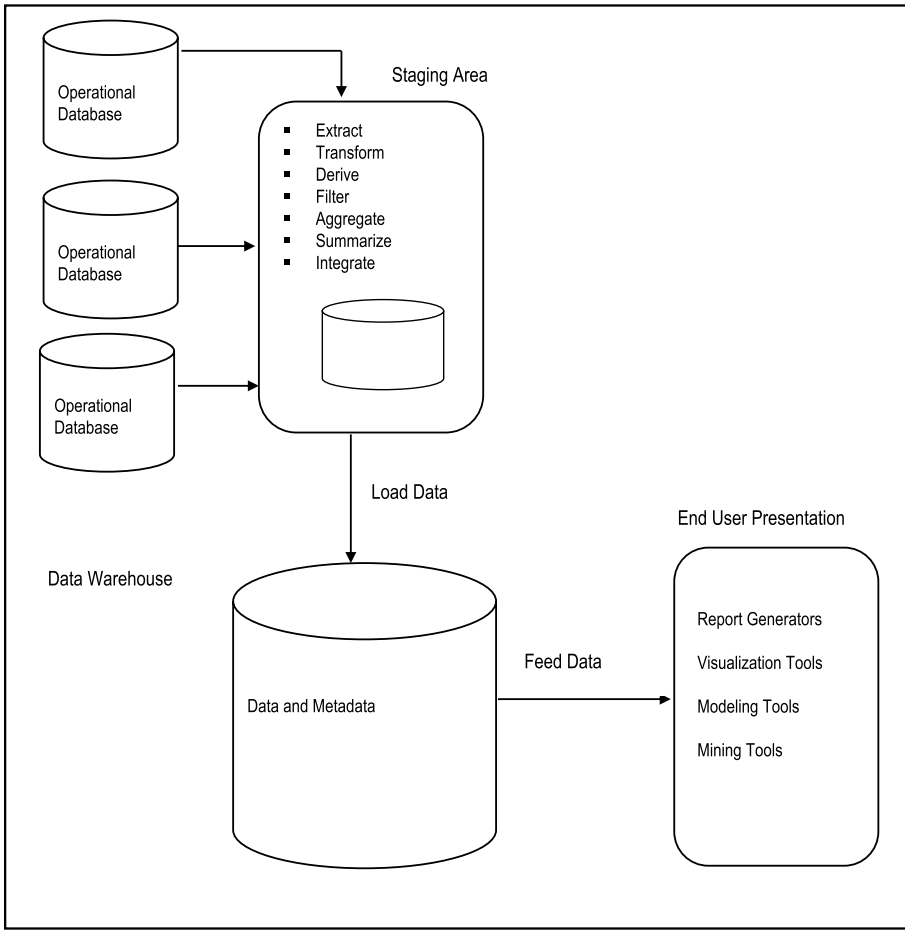

Figure 24-4. *Data Warehouse with Staging Area*

24.4.3 Data Warehouse Architecture with a Staging Area and Data Marts

Although the data warehouse with staging area is quite common, you may want to customize your warehouse's architecture for different groups within your organization, or across different related organizations. You can do this by adding *data marts*. A data mart is a small, single-subject data warehouse that provides decision support for a particular aspect (line) of the business. Figure 24-5 summarizes the approach for three data marts. For example, the data marts may respectively represent information relating to purchasing, sales, and inventory for an organization. A financial analyst would then be able to conduct separate analysis on purchasing, sales and inventory, and then a global analysis on the three aspects combined.

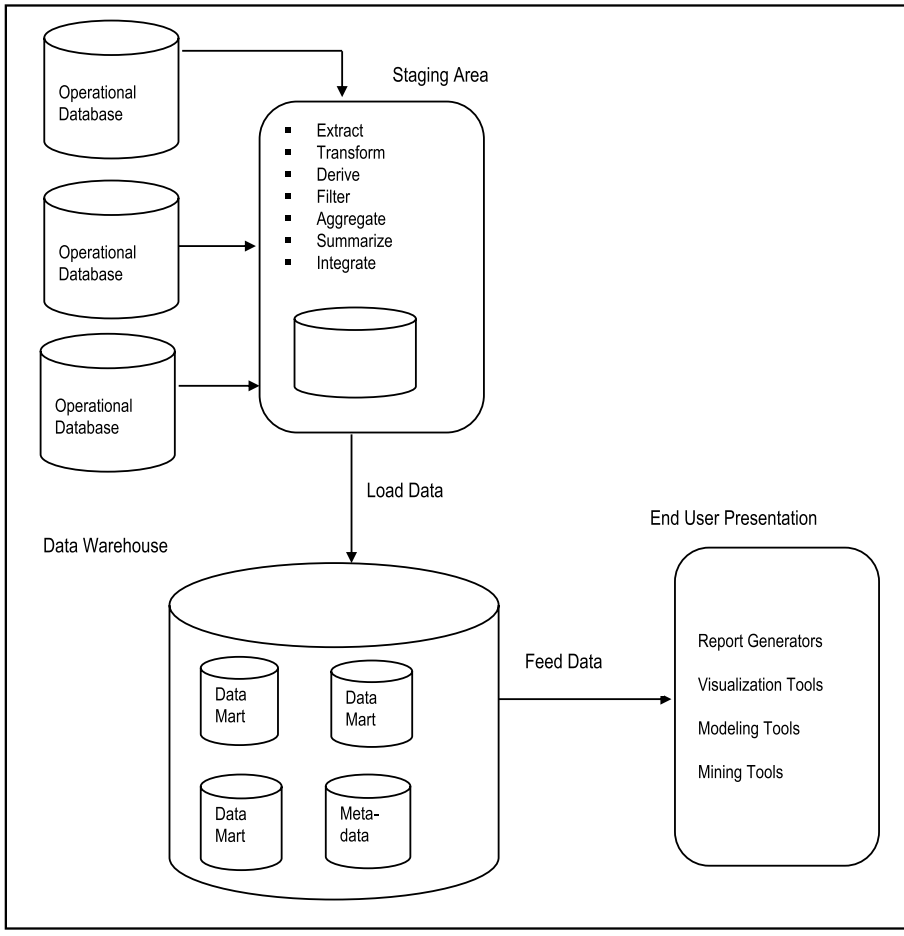

Figure 24-5. *Data Warehouse with Staging Area and Data Marts*

24.5 Extraction, Transformation, and Loading

The data warehouse must be loaded regularly so that it can serve its purpose of facilitating business analysis. To do this, data from one or more operational databases needs to be extracted, transformed (where necessary), and copied into the warehouse. The acronym ETL is often used to refer to this extraction, transformation and loading of data. The acronym is perhaps too simplistic, since it omits the transportation phase and implies that each of the other phases of the process is distinct. We refer to the entire process, including data loading, as ETL. However, you should understand that ETL refers to a broad process, and not three well-defined steps.

The methodologies and tasks of ETL have been well known for many years, and are not necessarily unique to data warehouse environments. A wide variety of proprietary applications and database systems are used as the IT backbone business enterprises.

Data are shared between applications or systems, trying to integrate them, giving at least two applications the same picture of the world. This data sharing was mostly addressed by mechanisms similar to what we now call ETL. Data warehouse environments face the same challenge with the additional burden that they not only have to exchange, but to integrate, rearrange and consolidate data over many operational systems, thereby providing a new unified information base for business intelligence. Additionally, the data volume in data warehouse environments tends to be very large.

24.5.1 What Happens During the ETL Process

During extraction, the desired data is identified and extracted from many different sources, including database systems and applications. Very often, it is not possible to identify the specific subset of interest, therefore more data than necessary has to be extracted, so the identification of the relevant data will be done at a later point in time.

Depending on the source system's capabilities (for example, operating system resources), some transformations may take place during this extraction process. The size of the extracted data varies from hundreds of kilobytes up to gigabytes, depending on the source system and the business situation. The same is true for the time difference between two (logically) identical extractions: the time span may vary between days/hours and minutes to near real-time. Web server log files for example can easily become hundreds of megabytes in a very short period of time. After extracting data, it has to be physically transported to the target system or an intermediate system for further processing. Depending on the chosen mode of transportation, some transformations can be done during this process, too. For example, an SQL statement that directly accesses a remote target through a gateway can concatenate two or more columns as part of the **Select** statement.

After transportation to the target system, the data may undergo transformation into the desired formats for the target system. Once this process is completed, the data is loaded into the data warehouse.

24.5.2 ETL Tools

Designing and maintaining the ETL process is often considered one of the most difficult and resource-intensive portions of a data warehouse project. Many data warehousing projects use ETL tools to manage this process.

Oracle Tools

Oracle Warehouse Builder (OWB) provides ETL capabilities and takes advantage of inherent database abilities. Other data warehouse builders create their own ETL tools and processes, either inside or outside the database.

Besides the support of extraction, transformation, and loading, there are some other tasks that are important for a successful ETL implementation as part of the daily operations of the data warehouse and its support for further enhancements. The OWB is quite a sophisticated component that facilitates the construction simple as well as complex data warehouses, the population of them via the ETL process, and the management of them.

DB2 Tools

As mentioned in Chapter 17, DB2 provides four main products for managing data warehouses:

- InfoSphere Warehouse Departmental Edition
- InfoSphere Warehouse Departmental Base Edition
- InfoSphere Warehouse Enterprise Edition
- InfoSphere Warehouse Enterprise Base Edition

Each of these products (the latter being an upgrade of the former) consists of related components that allow for the creation, population (via ETL transactions), and management of data warehouses, according to organizational requirements.

24.5.3 Daily Operations and Expansion of the Data Warehouse

Successive ETL transactions to the data warehouse must be scheduled and processed in a specific order. Depending on the success or failure of the operation or parts of it, the result must be tracked and subsequent, alternative processes might be started. The control of the progress and the definition of a business workflow of the operations are typically addressed by special ETL tools provided by the DBMS suite.

As the data warehouse is an active information system, data sources and targets are not beyond the prospect of change. These changes must be maintained and tracked through the lifespan of the system without overwriting or deleting the old ETL process flow information. To build and keep a level of trust about the information in the warehouse, the process flow of each individual record in the warehouse can be reconstructed at any point in time in the future. With time, the data warehouse could therefore expand into something larger and different.

24.6 Summary and Concluding Remarks

Here is a summary of what we have discussed in this chapter:

- A data warehouse is an integrated, subject-oriented, time-variant, nonvolatile, consistent database, obtained from a variety of sources and made available to support decision making in a business context.
- A data warehouse is a relational database that is designed for query and analysis rather than transaction processing.
- The data warehouse is updated via the ETL process.
- A data warehouse provides the decision support benefits that a traditional DSS provides, while providing more flexibility for expansion beyond the confines of the company.

- Data warehouses often use non-normalized or partially normalized schemas to optimize query performance.
- Data warehouses should conform to Inmon's twelve rules for data warehouses.
- Three common data warehouse architectures are the basic data warehouse, data warehouse with a staging area, and data warehouse with a staging area and data marts.

The field of data warehousing is a fascinating breakthrough and is the subject of many contemporary researches. Since it is relatively new, vast opportunities for data warehouse architecture and ETL transaction optimization still abound. Data warehousing is studied as an advanced course in several undergraduate degree programs, as well as graduate programs. The supporting technologies are provided via products from the three leading software engineering companies — IBM's DB2, Oracle (from Oracle Corporation), and Microsoft's SQL Server. Additionally, these companies provide readily available documentation on the topic.

With the advancement of the WWW and Web-accessible databases, it is anticipated that the fascination in data warehousing will continue into the foreseeable future. And speaking of Web-accessible databases, the next chapter discusses this topic.

24.7 Review Questions

1. Give the definition of a data warehouse. Clearly outline what data warehousing entails.
2. Provide a rationale for data warehousing.
3. Describe a scenario that would warrant the use of a data warehouse.
4. Discuss the main characteristics of a data warehouse in terms of:
   - Definitive features
   - Nature of data stored
   - Processing requirements
   - Rules that govern the data warehouse
5. State the three architectural approaches to data warehouse design. For each approach, describe the basic architecture, and provide a scenario that would warrant such an approach.
6. Clearly explain the ETL process. Give examples of ETL tools.

24.8 References and/or Recommended Readings

[Adelman, 2000] Adelman, Sid and Larissa Terpeluk Moss. *Data Warehouse Project Management*. Boston, MA: Addison-Wesley, 2000.

[Connolly, 2002] Connolly, Thomas and Carolyn Begg. *Database Systems: A Practical Approach to Design, Implementation and Management* 3rd ed. New York, NY: Addison-Wesley, 2002. See Chapters 30-32.

[Date, 2004] Date, Christopher J. *Introduction to Database Systems* 8h ed. Menlo Park, CA: Addison-Wesley, 2004. See Chapter 22.

[Elmasri, 2007] Elmasri, Ramez and Shamkant B. Navathe. *Fundamentals of Database Systems* 5th ed. Reading, MA: Addison-Wesley, 2007. See Chapters 28 and 29.

[Hoffer, 2007] Hoffer, Jeffrey A., Mary B. Prescott and Fred R. McFadden. *Modern Database Management* 8th ed. Upper Saddle River, NJ: Prentice Hall, 2007. See chapter 11.

[IBM, 2008] IBM Corporation. *Data Warehouse*. http://www-01.ibm.com/software/data/infosphere/warehouse/ (accessed October 2008).

[Inmon, 2002] Inmon, William. *Building the Data Warehouse* 3 rd ed. New York, NY: John Wiley, 2002.

[Inmon, 2007] Inmon Associates Inc. http://www.billinmon.com/ (accessed October 2008).

[Oracle, 2008] Oracle Corporation. *Oracle10g Database Release 2*. http://technetOracle.com (accessed October 2008).

[Rob, 2007] Rob, Peter and Carlos Coronel. *Database Systems: Design, Implementation & Management* 7th ed. Boston, MA: Course Technology, 2007. See Chapter 13.

[Silberschatz, 2006] Silberschatz, Abraham, Henry Korth, and S. Sudarshan. *Database System Concepts* 5th ed. Boston, MA: McGraw-Hill, 2006. See Chapters 18 – 19.

CHAPTER 25

Web-Accessible Databases

Another new area of database systems that has become widespread since the 1990s is Web-accessible databases. The proliferation of these databases is strongly correlated to the growth of the World Wide Web (WWW or W3); both phenomena have become part and parcel of twenty-first century lifestyle, and both promise to be an integral part of life in the foreseeable future. This Chapter provides an overview of Web-accessible databases. The Chapter proceeds under the following subheadings:

- Introduction
- Web-Accessible Database Architecture
- Supporting Technologies
- Implementation with Oracle
- Implementation with DB2
- Generic Implementation via Front-end and Back-end Tools
- Summary and Concluding Remarks

25.1 Introduction

A Web-accessible database is simply a database that is accessible via the WWW. The rationale for web-accessible databases can be easily appreciated when one considers the huge benefits that they bring to the business community. Figure 25-1 provides a brief discussion of some of the significant benefits.

Electronic Commerce: Through electronic commerce (E-commerce), companies are able to market their products and services via on-line stores in a manner that was impossible prior to the WWW.

Broadening of Company Scope: The on-line market is not constrained by space, time, or geographic region. Companies that trade in this market have virtually joined a global village in which resources and services are only seconds away.

Convenience: Companies that use Web-accessible databases afford themselves easy access to critical company information (preferably without sacrificing security) in very cost-effective way. These conveniences would have much more expensive (if at all possible) if pursued via more traditional methods.

Improved Productivity: By using Web-accessible databases, companies often improve their productivity by making use of resources that otherwise would have been more expensive (financially and in terms of time).

Improved Competitiveness: By using Web-accessible databases, companies often improve their competitive advantage in the marketplace.

Other Benefits: In many cases, Web-accessible databases involve the use of distributed databases. In such cases, the benefits of distributed databases (review section 22.2) also apply.

Figure 25-1. *Significant Benefits of Web-Accessible Databases*

25.2 Web-Accessible Database Architecture

In sections 2.6 and 2.7, we discussed the idea of separating a database system into front-end and back-end. This principle is commonly used in implementing Web-accessible databases. Two approaches are common: the two-tiered approach as represented in Figure 25-2, and the three-tiered approach as represented in Figure 25-3. In the two-tiered approach, client applications send requests to the DBMS, which is running on a database server. These requests are processed according to some scheduling algorithm. In the three-tiered approach, additional sophistication is provided by an intersecting application server, which services client requests from various (heterogeneous) applications and filters them to the DBMS for processing. This provides additional flexibility and functionality to the system.

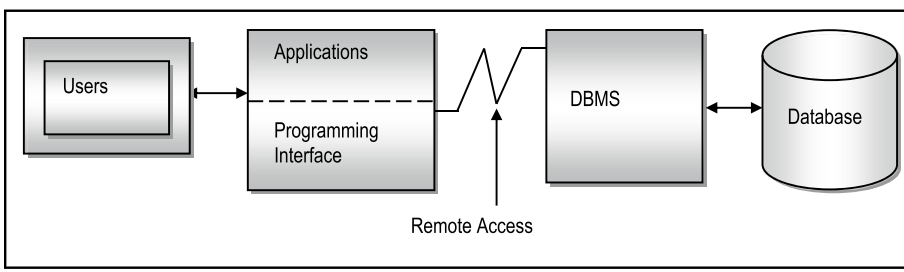

Figure 25-2. *Two-Tiered Approach to Web-Accessible Database*

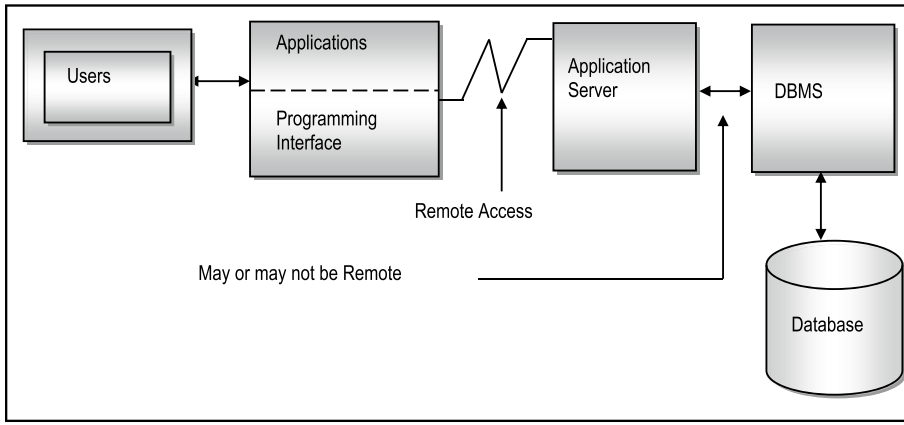

Figure 25-3. *Three-Tiered Approach to Web-Accessible Database*

As you view these two figures, bear in mind that users (in the figures) include end-users as well as businesses making electronic requests from other Web-accessible databases. This is so because there are two types of Web-accessible database systems that are prevalent:

- Consumer-to-business (C2B) systems facilitate individual users accessing a company database.
- Business-to-business (B2B) systems facilitate businesses accessing other Web-accessible databases of other businesses in a manner that is often transparent to the end-user. This relatively new market provides huge opportunities for companies to improve their efficiency and productivity by concentrating on what they do best, and outsourcing non-essential functions that other businesses provide more efficiently. This gives them the opportunity to also forge powerful alliances for more effective operation.

25.3 Supporting Technologies

The usefulness of separating front-end and back-end subsystems was illustrated in the previous section. This strategy allows us to design and secure the database as a separate activity from developing applications that access that database. Web applications could be one of the many different applications that access the database. Following are some supporting technologies for Web-accessible databases.

Web Servers: A Web server is a sophisticated software system that allows a computer to provide various services to multiple client requests made via the WWW. The server runs on an operating system (which runs on a computer). However, loosely speaking, we usually refer to the entire package of machine and software as the Web server.

Popular Web server software products include Apache, Microsoft's Internet Information Service (IIS), CERN server, NCSA (National Center for Supercomputing Applications) server, Spinner server, Plexus server, Perl server, Tomcat server, etc.

Server-side Extensions: A server-side extension is a software that communicates with a Web server to handle assorted client requests. Often, the server-side extension program acts as an intermediary between the Web server and the database, farming out all SQL requests to the DBMS. Both the DBMS and the server-side extension program must be ODBC-compliant. Products such as ColdFusion, Delphi, Java Studio Enterprise, etc. qualify as server-side extension programs.

Web Server Interfaces: Web server interfaces facilitate the display of information on dynamic Web pages. There are two popular categories:

- Common Gateway Interface (CGI) is a set of rules that specify how parameters are passed between client programs and Web servers. A client program that can run on a Web server is called a script, hence the term CGI script. Common scripting languages include JavaScript, Active Server Pages (ASP), and PHP. However, high-level languages such as C++, Java, Perl, etc. also qualify.
- Application Program Interface (API) is a set of routines, protocols and tools that facilitate easy software construction. Since APIs are typically shared code that is resident in memory (in the case of Web technology, they reside on the Web server), they tend to be more efficient than CGI scripts.

Extensible Markup Language (XML): XML is a meta-language that was designed specifically to facilitate the representation, manipulation and transmission of structured data over the WWW. It was first published by the WWW Consortium (W3C) in 1998, and to no surprise, has become the de facto data exchange standard for e-commerce, thus circumventing the pre-existing problem of interoperability among different Web servers.

XML was developed from an earlier standard called the Standard Generalized Markup Language (SGML). And as expected, other XML-based languages are emerging. Three examples are Extensible Business Reporting Language (XBRL), Structured Product Labeling (SPL), and Extensible Style Language (XSL).

Simple Object Access Protocol (SOAP): The original emphasis of SOAP was to support remote procedure calls (RPC). However, the norm is for SOAP messages to be sent by web servers as XML documents (synchronous as well as asynchronous). SOAP messages are frequently transmitted as the data portion of HTTP (hyper-text transport protocol) messages.

Hypertext Transport Protocol (HTTP): HTTP is the protocol used to transfer of information over the WWW. It is characterized by a simply request-response structure that represents interactions between a client (Web browser) and a Web server. It is assumed that you are familiar with the Internet, basic Web page construction, Uniform Resource Locators (URLs), domain names and other related issues.

Web Services Description Language (WSDL): The original authors of WSDL define it as "an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information" [Ariba, 2001] WSDL can be construed as an interface definition language (IDL) with bindings that clearly specify how the various components of a message are mapped. WSDL messages are frequently transmitted as SOAP messages over HTTP.

Client-side Extensions: These add functionality to the Web browsers. Client-side extensions are available in various forms. Following is a summary of the most commonly encountered ones:

- **Plug-ins:** A plug-in is an external application program that is automatically invoked by the Web browser when required. For example, a Web browser, upon receiving a PDF (portable document format) document will invoke the available PDF-reader on the host operating system.
- **Java:** As you are no doubt aware, Java is a platform independent programming language. Most operating systems have a *Java Virtual Machine* (JVM), which allows Java code to execute on the local machine (if there isn't a JVM in the operating system, this is readily available from the Sun Microsystems Web site). Calls to Java routines are often embedded in HTML pages. When a Web browser encounters this, it invokes the local JVM to execute the code.
- **JavaScript:** JavaScript is a Java-like scripting language (developed by Netscape), but is much simpler. JavaScript code is often embedded in Web pages. It is downloaded whenever the Web page is activated, as well as on certain specific events (such as loading of a specific page from the server, or a mouse-click, etc.) Whenever the browser encounters this, it invokes the JavaScript plug-in to execute this code.
- **ActiveX:** ActiveX is Microsoft's alternative to Java. It works perfectly in a Windows environment. Although possible in other languages, C++ and Visual Basic are well known for their facilitation of ActiveX controls.

- **VBScript:** VBScript is another Microsoft product that is often used to add functionality to the Web browser. Like JavaScript, VBScript code is often embedded in Web pages. Invocation and execution are also similar to JavaScript.
- **Cookies:** Cookies are used to expedite user requests when the user visits a site more than once. When the site is first visited, the Web server creates a cookie with basic user information (such as e-mail) and sends it to the browser. When the browser is subsequently used to visit that site, the cookie, upon recognition by the Web server, is used to expedite the user request.

25.4 Implementation with Oracle

Oracle implements a database that is by definition, Web-accessible (this applies since Oracle 10G). This is achieved through the products Oracle Enterprise Manager, (OEM), iSQL*Plus, Oracle JDeveloper, and Oracle SQL Developer in the following ways:

- When you install the Oracle Server suite (including OEM) on a node in your company or home network, the machine is automatically configures as a database server.
- Oracle automatically installs and configures a database listener on database server to respond to incoming requests from client nodes (running Oracle 10G or 11G Client) in the network, on the internet, or an extranet.
- Typically, your database server should have at least one database. This can be created during installation, or subsequently via the Oracle Database Configuration Assistant (DBCA) component.
- You can access your database server from any machine that has an internet connection, through OEM via the URL `http://<Machine.Domain>:5500/em` (you supply the machine name and domain name for your network). Of course, you will need a valid user account, password, and appropriate privileges.
- You can also access your database server from any machine that has an internet connection, through Oracle's iSQL*Plus via the URL `http://<Machine.Domain>:5560/isqlplus` (again, you supply the machine name and domain name for your network). Again, you will need a valid user account, password, and appropriate privileges.
- Through Oracle JDeveloper (OJD), Oracle allows Web-accessible Java-based applications to be constructed. These applications may access local or remote Oracle (or heterogeneous) databases.

- Through Oracle Forms Developer (OFD), Oracle also allows Web-accessible PL/SQL-based applications to be constructed. These applications may access local or remote Oracle (or heterogeneous) databases.

This development represents a huge step forward for Oracle, and has no doubt fuelled its invigorated claim of being the leading software engineering firm for Web-accessible databases.

25.5 Implementation with DB2

Like Oracle, DB2 implements a Web-accessible database (this applies to DB2 8 and subsequent versions). This is achieved through the products DB2 Connect, DB2 Everyplace and WebSphere in the following ways:

- If you configure a mainframe, or mini computer to be a DB2 server, and create your DB2 database on it, you can access the database over an intranet, the Internet, or an extranet by running DB2 Connect from your client machine. As in the case of the Oracle database, you will need to know the machine name, the domain name and port number that the database server listens on, in order to connect to it.
- DB2 Everyplace is a miniscule version of DB2 that runs on mobile devices such as *personal digital assistants* (PDAs), cell phones, etc. DB2 Everyplace can be used as a local independent DBMS, or to query information residing at a remote database server, via the WWW.
- Through WebSphere, IBM allows Web-accessible Java-based applications to be constructed. These applications may access local or remote DB2 (or heterogeneous) databases. There is also a WebSphere Everyplace version for PDAs, cell phones and other pervasive devices.

Like Oracle, IBM also claims to be the leading software engineering firm for Web-accessible databases. The truth is, both companies are archrivals at the top of the database systems market.

25.6 Generic Implementation via a Front-end and a Back-end Tool

This section describes a generic approach to constructing a Web-accessible database system, based on front-end and back-end tools. The operation can be summarized in two steps:

1. Create the database using an appropriate back-end tool. The tool used must support ODBC, JDBC, or both. Of course, it is assumed that appropriate planning and design as discussed in earlier Chapters, have taken place.

2. Create the Web-application using an appropriate front-end tool that incorporates the requisite Web-supporting technologies as discussed in section 25.3. The tool must support ODBC and/or JDBC, and must facilitate code in at least one of the accepted scripting languages (JavaScript, PHP, ASP, etc.). It must also support XML. Again, the basic assumption is that sound principles of software engineering will be used to design the user interface as a pre-requisite to this activity.

Figure 25-4 provides a list of commonly used tools. They have been listed in three categories: front-end tools, back-end tools, and programming languages. In each, it is recommended that you use the most current version of the stated software product (for the RAD tools and DBMS suites, the versions stated are simply safe starting points).

Product	Parent Company
Front-end RAD Tools that Support Web-Accessible Databases	
Delphi 2005	Embarcadero Technologies
WebSphere 6.0	IBM
NetBeans 6.0	Sun Microsystems
ColdFusion MX7	Macromedia
Oracle JDeveloper 10G	Oracle
Visual Studio 2005	Microsoft
Relational and/or Universal DBMS Suites	
DB2 8.2	IBM
Oracle 10G	Oracle
SQL Server 2005	Microsoft
Informix 10	IBM
MySQL 5.0	MySQL AB
Programming/Scripting Languages	
Java, C++, Object Pascal, JavaScript, PHP, XML, ASP, VBScript, Perl	

Figure 25-4. *Commonly Used Tools for Web-accessible Databases*

25.7 Summary and Concluding Remarks

Let us summarize what we have covered in this Chapter:

- A Web-accessible database is simply a database that can be accessed via the WWW.
- Web-accessible databases provide a number of benefits to companies and individuals that implement and/or use them. Among the benefits are facilitation of e-commerce, broadening of the company's scope of operation and market reach, a wide range of conveniences, improved productivity, and improved competitive advantage.
- A Web-accessible database may be implemented as a two-tiered system or a three-tiered system. Additionally, they may be C2B or B2B.
- The supporting technologies for Wed-accessible databases include Web servers, server-side extensions, server interfaces, XML, SOAP, WSDL, and client-extensions.
- Both Oracle and DB2 implement Web-accessible databases as a matter of policy.
- Implementation of a Web-accessible database can be summarized into two simple but profound steps: creating the database and creating the Web-accessible user interface. The tools used must meet minimum industry standards.

Web-accessible databases are among one of the technology-related phenomena that have transformed life in the twenty-first century. They are expected to continue to be an integral part of life in the foreseeable future.

This takes us to the end of the chapters for the course. If you understand most of the issues discussed, and now have a desire to delve more deeply into some aspects of the field, then the course has succeeded in its intent. If you find that you now have a strong desire to make database systems one of your areas of expertise, then welcome to the community! You will find it a wonderfully rewarding and progressive field.

25.8 Review Questions

1. Give the definition of a web-accessible database. Provide justification for their existence.
2. Discuss two example of the usefulness of Web-accessible databases.
3. Discuss the two-tiered approach to implementing Web-accessible database. When should you use such an approach?

4. Discuss the three-tiered approach to implementing Web-accessible database. When should you use such an approach?
5. Briefly describe the main supporting technologies for Web-accessible databases.
6. Describe how Oracle implements Web-accessible databases.
7. Describe how DB2 implements Web-accessible databases.
8. Describe a generic approach for implementing a Web-accessible database. What precautions must be taken?

25.9 References and/or Recommended Readings

[Ariba, 2001] Ariba, IBM Corporation, and Microsoft. *Web Services Description Language (WSDL) 1.1*. http://www.w3.org/TR/wsdl (accessed August 2009).

[Date, 2004] Date, Christopher J. *Introduction to Database Systems* 8th ed. Menlo Park, CA: Addison-Wesley, 2004. See Chapter 27.

[Elmasri, 2007] Elmasri, Ramez and Shamkant B. Navathe. *Fundamentals of Database Systems* 5th ed. Reading, MA: Addison-Wesley, 2007. See Chapters 26 and 27.

[Hoffer, 2007] Hoffer, Jeffrey A., Mary B. Prescott and Fred R. McFadden. *Modern Database Management* 8th ed. Upper Saddle River, NJ: Prentice Hall, 2007. See Chapter 10.

[Kifer, 2005] Kifer, Michael, Arthur Bernstein and Philip M. Lewis. Database Systems: An Application-Oriented Approach 2nd ed. New York, NY: Addison-Wesley, 2005. See Chapter 17.

[Riccardi, 2003] Riccardi, Greg. *Database Management With Web Site Development Applications*. Boston, MA: Addison-Wesley, 2003. See Chapters 12-15.

[Rob, 2007] Rob, Peter and Carlos Coronel. *Database Systems: Design, Implementation & Management* 7th ed. Boston, MA: Course Technology, 2007. See Chapter 14.

[Silberschatz, 2006] Silberschatz, Abraham, Henry Korth, and S. Sudarshan. *Database System Concepts* 5th ed. Boston, MA: McGraw-Hill, 2006. See Chapter 10.

[W3C, 2003] World Wide Web Consortium. *Extensible Markup Language (XML)*. http://www.w3.org/XML/ (accessed August 2009).