INF 3703 - DATABASES II

Summary 2013

Database Principles: Fundamentals of Design, Implementation, and Management
10th Edition - Coronel C, Morris S, Rob P.

v1.00 September 2013
Ron Barnard

Chapter 10 - Distributed Databases

Summary

- A distributed database stores logically related data in two or more physically independent sites connected via a computer network. The database is divided into fragments, which can be a horizontal set of rows or a vertical set of attributes. Each fragment can be allocated to a different network node.
- Distributed processing is the division of logical database processing among two or more network nodes. Distributed databases require distributed processing. A distributed database management system (DDBMS) governs the processing and storage of logically related data through interconnected computer systems.
- The main components of a DDBMS are the transaction processor (TP) and the data processor (DP). The transaction processor component is the resident software on each computer node that requests data. The data processor component is the resident software on each computer node that stores and retrieves data.
- Current database systems can be classified by the extent to which they support processing and data distribution. Three major categories are used to classify distributed database systems: single-site processing, single-site data (SPSD); multiple-site processing, single-site data (MPSD); and multiple-site processing, multiple-site data (MPMD).
- A homogeneous distributed database system integrates only one particular type of DBMS over a computer network. A heterogeneous distributed database system integrates several different types of DBMSs over a computer network.
- DDBMS characteristics are best described as a set of transparencies: distribution, transaction, performance, failure and heterogeneity. All transparencies share the common objective of making the distributed database behave as though it were a centralized database system; that is, the end user sees the data as part of a single, logical centralized database and is unaware of the systems' complexities.
- A transaction is formed by one or more database requests. An undistributed transaction updates or requests data from a single site. A distributed transaction can update or request data from multiple sites.
- Distributed concurrency control is required in a network of distributed databases. A two-phase COMMIT protocol is used to ensure that all parts of a transaction are completed.
- A distributed DBMS evaluates every data request to find the optimum access path in a distributed database. The DDBMS must optimize the query to reduce associated access costs, communication costs, and CPU costs.

- The design of a distributed database must consider the fragmentation and replication of data. The designer must also decide how to allocate each fragment or replica to obtain better overall response time and to ensure data availability to the end user. Ideally, a distributed database should evenly distribute data to maximize performance, availability, and location awareness.

- A database can be replicated over several different sites on a computer network. The replication of the database fragments has the objective of improving data availability, thus decreasing access time. A database can be partially, fully, or not replicated. Data allocation strategies are designed to determine the location of the database fragments or replicas.

- The CAP theorem states that a highly distributed data system has some desirable properties of consistency, availability, and partition tolerance. However, a system can only provide two of these properties at a time.

Content

10.1 The Evolution of Distributed Database Management Systems

A distributed database management system (DDBMS) governs the storage and processing of logically related data over interconnected computer systems, in which both data and processing are distributed among several sites.

10.2 DDBMS Advantages and Disadvantages

Advantages	**Disadvantages**
Data are located near the site of greatest demand - Data dispersed to match business requirements.	**Complexity of management and control** - Working with data at various locations.
Faster data access - Work with nearest stored data subset.	**Technological difficulty** - More technical issues to deal with.
Faster data processing - Data processed at several sites.	**Security** - Probability of security lases increases when data are stored at multiple sites.
Growth facilitation - New sites can be added without affecting the operation of other sites.	**Lack of standards** - No standard communication protocols.
Improved communications - Local sites are smaller and located closer to customers.	**Increased storage and infrastructure requirements** - Multiple copies of data at multiple sites.
Reduced operating costs - More cost-effective to add nodes to a network than upgrade a mainframe.	**Increased training cost** - Higher than compared to a centralized model.
User-friendly interface - PC's equipped with easy-to-use GUI.	**Costs** - Require duplicated infrastructure.
Less danger of single-point failure - If one computer fails, workload is picked up by other computers.	
Processor independence - User can access any available copy of the data, and request is processed by any processor at the data location.	

10.3 Distributed Processing and Distributed Databases

Distributed processing - a databases' logical processing is shared among two or more physically independent sites that are connected through a network. The database is located on one computer, but several sites can access the data and update the database.

Distributed database - stores a logically related database over two or more physically independent sites, connected via a network.

Database fragments - Distributed database is composed of several parts known as database fragments, located at different sites. Can be replicated among various sites. Each fragment is managed by its local database process.

10.4 Characteristics of Distributed Database Management Systems

A DDBMS governs the storage and processing of logically related data over interconnected computer systems in which both data and processing functions are distributed among several sites. DBMS must have the following functions to be classified as distributed -

- Application interface - to interact with users, application programs, and other DBMSs.
- Validation - to analyze data requests for syntax correctness.
- Transformation - t decompose complex requests into atomic data request components.
- Query optimization - to find the best access strategy.
- Mapping - to determine the data location of local and remote fragments.
- I/O interface - to read / write data t / from permanent storage.
- Formatting - to prepare data for presentation.
- Security - to provide data privacy.
- Backup and recovery - to ensure availability of the database in case of failure.
- DB administration features - for database administrator.
- Concurrency control - to manage simultaneous data access and to ensure data consistency.
- Transaction management - to ensure that data moves from one consistent state to another.

A fully distributed database management system must perform all of the functions of a centralized DBMS system -

- Receive the request of an application or end user.
- Validate, analyze, and decompose the request.
- Map the requests' logical-to-physical data components.
- Decompose the request into several disk I/O operations.
- Search for, locate, read, and validate the data.
- Ensure database consistency, security, and integrity.
- Validate the data for the conditions, if any, specified by the request.
- Present the data in the required format.

10.5 DDBMS Components

DDBMS must include at least the following components -

- Computer workstations or remote devices.

- Network hardware and software

- Communications media, that carry the data from one node to another.

- **Transaction Processor** (TP) - which is the software component found in each computer or device that requests data. Receives and processes the applications' remote and local data requests. (Also known as the **Application Processor** (AP) or **Transaction Manager** (TM)).

- **Data Processor** (DP) - which is the software component residing on each computer or device that stores and retrieves data located at that site. (Also known as **Data manager** (DM)). May even be a centralized DBMS.

A TP and a DP can reside on the same computer, allowing an end user to access both local and remote data transparently.

10.6 Levels of Data and Process Distribution

	Single-site data	**Multiple-site-data**
Single-site process	Host DBMS	n/a
Multiple-site process	File server Client / server DBMS (LAN DBMS)	Fully distributed Client / server DDBMS

10.6.1 Single-Site Processing, Single-Site Data (SPSD)

All processing is done on a single host computer and all data are stored on the host computers' local disk system. The DBMS is on the host computer, which is accessed by terminals. Mainframe / midrange Unix / Linux server DBMSs.

10.6.2 Multiple-Site Processing, Single-Site Data (MPSD)

Multiple processes run on different computers that share a single data repository. Typically a network file server running conventional applications that are accessed through a network. The DP is on the server, TP on each computer.

10.6.3 Multiple-Site Processing, Multiple-Site Data (MPMD)

Fully distributed DBMS with support for multiple data processors (DP) and transaction processors (TP) at multiple sites.

Homogeneous DDBMS - integrate multiple instances of the same DBMS over a network.

Heterogeneous DDBMS - integrate different types of DBMSs over a network, but all support the same data model.

Fully Heterogeneous DDBMS - supports different DBMSs, each one supporting a different data model, running under different computer systems.

10.7 Distributed Database Transparency Features

A distributed database system should provide some desirable transparency features that hide the systems' complexity to to the end user. Should appear similar to working on a centralized DBMS.

Minimum desirable DDBMS transparency features are -

- **Distribution transparency** - allows a distributed database to be treated as a single logical database.

- **Transaction transparency** - allows a transaction to update data at more than one network site. Ensures that the transaction will be either entirely completed or aborted, thus maintaining database integrity.

- **Failure transparency** - ensures that the system will continue to operate in the event of a node or network failure. Important feature.

- **Performance transparency** - allows the system to perform as if it were a centralized DBMS.
- **Heterogeneity transparency** - allows the integration of several different local DBMSs (relational, network, and hierarchical) under a common, or global, schema.

10.8 Distribution Transparency

Allows a physically dispersed database to be managed as though it were a centralized database.
Three levels of distribution transparency -

- **Fragmentation transparency** - highest level of transparency. End user does not need to know that a database is partitioned. Neither fragment names nor locations are specified prior to data access.
- **Location transparency** - exists when the end user must specify the database fragment names, but does not need to specify the fragment locations.
- **Local mapping transparency** - exists when the end user must specify both the fragment names and locations.

Distribution transparency is supported by a **distributed data dictionary** (DDD) or **distributed data catalog** (DDC), which contains the description of the entire database as seen by the database administrator.

10.9 Transaction Transparency

Transaction transparency is a DDBMS property that ensures database transactions will maintain the distributed databases' integrity and consistency. A DDBMS database transaction can update data stored in many different computers connected in a network. Transaction transparency ensures that the transaction will be completed only when all database sites involved in the transaction complete their part of the transaction.

10.9.1 Distributed Requests and Distributed Transactions

Remote request - lets a single SQL statement access the data that are to be processed by a single remote database processor.

Remote transaction - composed of several requests, accesses data at a single remote site.

Distributed request - lets a single SQL statement reference data located at several different local or remote DP sites.

Distributed transaction - can reference several different local or remote DP sites. Each single request can reference only one local or remote DP site, the transaction as a whole can reference multiple DP sites.

Remote - single location.
Distributed - multiple locations.

Request - single SQL statement.
Transaction - multiple SQL statements.

10.9.2 Distributed Concurrency Control

Concurrency control is especially important in distributed databases because multisite, multiple process operations are more likely to create data inconsistencies and deadlocked transactions. The TP component of a DDBMS must ensure that all parts of a transaction are completed at all sites before a final COMMIT is issued.

If a transaction updates data at three DP sites, and the first two complete and COMMIT the data, but the third cannot commit, there would be an inconsistent database with integrity problems - cannot uncommit committed data. The solution is a two-phase commit protocol.

10.9.3 Two-Phase COMMIT Protocol

Centralized databases require only one DP. Distributed databases make it possible for a transaction to access data at several sites. A final COMMIT cannot be issued until all sites have committed their parts of a transaction. The **two-phase commit protocol** (2PC) guarantees that if a portion of a transaction operation cannot be committed, all changes made at the other sites participating in the transaction will be undone to maintain a consistent database.

Each DP maintains its own transaction log. The two-phase commit protocol requires that the transaction log entry for each DP be written before the database fragment is actually updated. Therefore it requires a DO-UNDO-REDO protocol and a write-ahead protocol.

The **DO-UNDO-REDO protocol** is used by the DP to roll transactions back and forward with the help of the systems' transaction log entries. Defines three types of operations -

- DO performs the operation and records the "before" and "after" values in the transaction log.
- UNDO reverses an operation, using the log entries written by the DO portion of the sequence.
- REDO redoes an operation, using the log entries written by the DO portion of the sequence.

Write-ahead protocol - Ensures that the DO-UNDO-REDO operations can survive a system crash. It forces the log entry to be written to permanent storage before the actual operation takes place. If one of the nodes fails to commit, the information necessary to recover the database is in the transaction log.

10.10 Performance and Failure Transparency

Performance transparency allows a DDBMS to perform as if it were a centralized database. No performance degradation should be incurred due to data distribution.

Failure transparency ensures that the system will continue to operate in the case of a node or network failure.

Carefully planning how to partition a database and where to locate the database fragments can help ensure the performance and consistency of a distributed database.

10.11 Distributed Database Design

Design of a distributed database introduces three new issues -

- How to partition the database into fragments;
- Which fragments to replicate;
- Where to locate those fragments and replicas.

Data fragmentation and data replication deal with the first two issues, and data allocation deals with the third issue. Ideally, data in a distributed database should be evenly distributed to maximize performance, increase availability (reduce bottlenecks), and provide location awareness, an ever-increasing requirement for mobile applications.

10.11.1 Data Fragmentation

Data fragmentation - allows one to break a single object into two or more segments, or fragments. The object might be a users' database, a system database, or a table. Each fragment can be stored at any site over a computer network. Information about data fragments is stored in the distributed data catalog (DDC), from which it is accessed by the TP to process requests.

Consider table level fragmentation:

Horizontal fragmentation - division of a relation into subsets (fragments) of tuples (rows). Each fragment is stored at a different node, and each fragment has unique rows. The unique rows all have the same attributes (columns). Each fragment represents the equivalent of a SELECT statement, with the WHERE clause on a single attribute.

- **Round-robin partitioning** - Rows are assigned to a given fragment in round-robin fashion (F1, F2, F3 etc) to ensure an even distribution. Not a good strategy if location awareness is required.

- **Range partitioning** - based on a partition key, one or more attributes which determine the fragment in which a row will be stored. Allows location awareness. Most common and useful partitioning strategy.

Vertical fragmentation - division of a relation into attribute (column) subsets. Each subset (fragment is stored at a different node, and each fragment has unique columns - with the exception of the key column, which is common to all fragments. Equivalent of the PROJECT statement.

Mixed fragmentation - combination of horizontal and vertical strategies. A table may be divided into several horizontal subsets (rows), each one having a subset of the attributes (columns).

10.11.2 Data Replication

Data replication - refers to the storage of data copies at multiple sites served by a computer network. Fragment copies can be stored at at several sites to serve specific information requirements. The existence of fragment copies can enhance data availability and response time, so data copies can help to reduce communication and total query costs.

Mutual consistency rule - requires that all copies of data fragments must be identical. The DDBMS must ensure that a database update is performed at all sites where replicas exist.

- **Push replication** - After a data update, the originating DP node sends the changes to the replica nodes. Focuses on maintaining data consistency. Decreases data availability due to latency involved in ensuring data consistency at all nodes.

- **Pull replication** - After a data update, the originating DP node sends "messages" to the replica nodes to notify them of the update. The replica nodes decide when to apply the update. Updates propagate more slowly to the replicas, with the focus on maintaining data availability. Allows for temporary data inconsistencies.

Benefits of replication - improved data availability, better load distribution, improved data failure tolerance, and reduced query costs.

Disadvantages of replication - additional DDBMS processing overhead because each data copy must be maintained by the system. There are associated storage costs and increased transaction times.

There are three replication scenarios -

- **Fully replicated database** - stores multiple copies of each database fragment at multiple sites. Can be impractical due to the amount of overhead it imposes on the system.

- **Partially replicated database** - stores multiple copies of some database fragments at multiple sites. Most DDBMSs are able to handle partially replicated database well.

- **Unreplicated database** - stores each database fragment at a single site, there are no duplicate database fragments.

10.11.3 Data Allocation

Data allocation - describes the process of deciding where to locate data. Data allocation is closely related to the way a database is divided or fragmented. Three strategies -

- **Centralized data allocation** - entire database is stored at one site.

- **Partitioned data allocation** - database is divided into two or more disjointed parts (fragments) and stored at two or more sites.

- **Replicated data allocation** - copies of one or more database fragments are stored at several sites.

10.12 The CAP Theorem

There are three commonly desirable properties in a highly distributed data system -

Consistency - All nodes should see the same data at the same time, all replicas should be updated immediately. Involves dealing with latency and network partitioning delays.

Availability - A request is always fulfilled by the system, no request is ever lost.

Partition tolerance - System continues to operate even in the event of node failure. The system will fail only if all nodes fail.

CAP Theorem - It is impossible for a system to provide all three properties at the same time.

ACID properties - Four database transaction properties - Atomicity, Consistency, Isolation, and Durability, which ensure that all successful transactions result in a consistent database state,

When dealing with highly distributed systems, some companies tend to forfeit the consistency and isolation components of the ACID properties to achieve higher availability. Generated a new type of distributed data systems -

BASE Basically available, soft state, eventually consistent - A data consistency model in which data changes are not immediate, but propagate slowly through the system until all replicas are eventually consistent. Eg: NoSQL databases provide highly distributed database with eventual consistency.

DBMS Type	Consistency	Availability	Partition Tolerance	Transaction Model	Trade-Off
Centralized DBMS	High	High	n/a	ACID	No distributed data processing
Relational DDBMS	High	Relaxed	High	ACID (2PC)	Sacrifices availability to ensure consistency and isolation
NoSQL DDBMS	Relaxed	High	High	BASE	Sacrifices consistency to ensure availability

10.13 CJ Dates' 12 Commandments for Distributed Databases

Most vendors have implemented their own versions of distributed databases. CJ Dates 12 Commandments represent a useful target, or ideal. No current DDBMS conforms to all of them.

-ooOoo-

Key Terms

- **application processor** (AP) - See *transaction processor* (TP).
- **basically available, soft state, eventually consistent** (BASE) - A data consistency model in which data changes are not immediate, but propagate slowly through the system until all replicas are eventually consistent.
- **centralized data allocation** - A data allocation strategy in which the entire database is stored at one site. Also known as a *centralized database*.
- **client / server architecture** - The arrangement of hardware and software components to form a system composed of clients, servers, and middleware. The client / server architecture features a user of resources, or a client, and a provider of resources, or a server.
- **coordinator** - The transaction processor (TP) node that coordinates the execution of a two-phase COMMIT in a DDBMS. See also *data processor* (DP), *transaction processor* (TP), and *two-phase commit protocol*.
- **data allocation** - In a distributed DBMS, the process of deciding where to locate data fragments.
- **database fragments** - See below.
- **data fragmentation** - A characteristic of a DDBMS that allows a single object to be broken into two or more segments or fragments. The object might be a users database, a system database, or a table. Each fragment can be stored at any site on a computer network.
- **data manager** (DM) - A DP specialist who evolved into a department supervisor. Roles include managing technical and human resources, supervising senior programmers, and troubleshooting the program. Also known as data processing (DP) manager.
- **data processor** (DP) - The resident software component that stores and retrieves data through a DDBMS. The DP is responsible for managing the local data in the computer and coordinating access to that data. See also *transaction processor* (TP).
- **data replication** - The storage of duplicated database fragments at multiple sites on a DDBMS. Duplication of the fragments is transparent to the end-user. Data replication provides fault tolerance and performance enhancements.
- **distributed database** - A logically related database that is stored in two or more physically independent sites.
- **distributed database management system** (DDBMS) - A DBMS that supports a database distributed across several different sites; a DDBMS governs the storage and processing of logically related data over interconnected computer systems in which both data and processing functions are distributed among several sites.
- **distributed data catalog** (DDC) - A data dictionary that contains the description (fragment names, locations) of a distributed database. Also known as a *distributed data dictionary* (DDD).
- **distributed data dictionary** (DDD) - See *distributed data catalog*.
- **distributed global schema** - The database schema description of a distributed database as seen by the database administrator.
- **distributed processing** - Sharing the logical processing of a database over two or more sites connected by a network.

- **distributed request** - A database request that allows a single SQL statement to access data in several remote data processors (DPs) in a distributed database.

- **distributed transaction** - A database transaction that accesses data in several remote data processors (DPs) in a distributed database.

- **distributed transparency** - A DDBMS feature that allows a distributed database to look like a single logical database to an end user.

- **DO-UNDO-REDO protocol** - A protocol used by a data processor (DP) to roll back, or roll forward transactions with the help of a systems' transaction log entries.

- **failure transparency** - A feature that allows continuous operation of a DDBMS, even if a network node fails.

- **fragmentation transparency** - A DDBMS feature that allows a system to treat a distributed database as a single database even though it is divided into two or more fragments.

- **fully heterogeneous DDBMS** - A system that integrates different types of database management systems (hierarchical, network, and relational) over a network. It supports different database management systems that may even support different data models running under different computer systems, such as mainframes, minicomputers, and microcomputers. See also *heterogeneous DDBMS* and *homogeneous DDBMS*.

- **fully replicated database** - In a DDBMS, the distributed database that stores multiple copies of each database fragment at multiple sites.

- **heterogeneity transparency** - A feature that allows a system to integrate several centralized DBMSs into one logical DDBMS.

- **heterogeneous DDBMS** - A system that integrates different types of centralized database management systems over a network. See also *fully heterogeneous distributed database system* (*fully heterogeneous DDBMS*) and *homogeneous DDBMS*.

- **homogeneous DDBMS** - A system that integrates only one type of centralized database management system over a network. See also *heterogeneous DDBMS* and *fully heterogeneous distributed database system* (*fully heterogeneous DDBMS*).

- **horizontal fragmentation** - The distributed database design process that breaks a table into subsets of unique rows. See also *database fragment* and *vertical fragmentation*.

- **local mapping transparency** - A property of a DDBMS in which database access requires the end user to know both the name and location of the fragments. See also *location transparency*.

- **location transparency** - A property of a DDBMS in which database access requires the user to know only the name of the database fragments. (Fragment locations need not be known). See also local *mapping transparency*.

- **mixed fragmentation** - A combination of horizontal and vertical strategies for data fragmentation, in which a table may be divided into several rows and each row has a subset of the attributes (columns).

- **multiple-site processing, multiple-site data** (MPMD) - A scenario describing a fully distributed database management system with support for multiple data processors and transaction processors at multiple sites.

- **multiple-site processing, single-site data** (MPSD) - A scenario in which multiple processes run on different computers sharing a single data repository.

- **mutual consistency rule** - A data replication rule that requires all copies of data fragments to be identical.
- **network latency** - The delay imposed by the amount of time required for a data packet to make a round trip from point A to point B.
- **network partitioning** -
- **partially replicated database** - A distributed database in which copies of only some database fragments are stored at multiple sites. See also *fully replicated database*.
- **partitioned data allocation** - A data allocation strategy of dividing a database into two or more fragments that are stored at two or more sites.
- **partition key** - In partitioned databases, one or more attributes in a table that determine the fragment in which a row will be stored.
- **performance transparency** - A DDBMS feature that allows a system to perform as though it were a centralized DBMS.
- **remote request** - A DDBMS feature that allows a single SQL statement to access data in a single remote DP. See also *remote transaction*.
- **remote transaction** - A DDBMS feature that allows a transaction (formed by several requests) to access data in a single remote DP. See also *remote request*.
- **replica transparency** - The DDBMSs ability to hide the existence of multiple copies of data from the user.
- **replicated data allocation** - A data allocation strategy in which copies of one or more database fragments are stored at several sites.
- **single-site processing, single-site data** (SPSD) - A scenario in which all processing is done on a single CPU or host computer, and all data are stored on the host computers' local disk.
- **subordinates** - In a DDBMS, a data processor (DP) node that participates in a distributed transaction using the two-phase COMMIT protocol.
- **transaction manager** (TM) - See *transaction processor* (TP).
- **transaction processor** (TP) - In a DDBMS, the software component on each computer that requests data. The TP is responsible for the execution and coordination of all databases issued by a local application that accesses data on any DP. Also called *transaction manager* (TM). See also *data processor* (DP).
- **transaction transparency** - A DDBMS property that ensures database transactions will maintain the distributed databases' integrity and consistency, and that a transaction will be completed only when all database sites involved complete their part of the transaction.
- **two-phase commit protocol** - In a DDBMS, an algorithm used to ensure atomicity of transactions and database consistency, as well as integrity in distributed transactions.
- **unique fragment** - In a DDBMS, a condition in which each row is unique, regardless of which fragment it is located in.
- **unreplicated database** - A distributed database in which each database fragment is stored at a single

site.

- **vertical fragmentation** - In distributed database design, the process that breaks a table into a subset of columns from the original table. Fragments must share a common primary key. See also *database fragment* and *horizontal fragmentation*.

- **write-ahead protocol** - In concurrency control, a process that ensures transaction logs are written to permanent storage before any database data are actually updated. Also called *write-ahead-log protocol*.

--ooOoo--

Chap 11 - Interacting with Databases through the Web

Summary

- Database connectivity refers to the mechanisms through which application programs connect and communicate with data repositories. Database connectivity software is also known as *database middleware.*

- Microsoft database connectivity interfaces are dominant players in the market and enjoy the support of most database vendors. OBDC, OLE-DB, and ADO.NET form the backbone of Microsofts' Universal Data Access (UDA) architecture.

- Native database connectivity refers tot he connection interface that is provided by the database vendor and is unique to that vendor. OBDC is probably the most widely supported database connectivity interface. OBDC allows any Windows application to access relational data sources using standard SQL. Data Access Objects (DAO) is an older, object-oriented application interface. Remote Data Objects (RDO) is a higher-level, object-oriented application interface used to access remote database servers. RDO was optimized to deal with server-based databases such as MS SQL Server and Oracle.

- Object Linking and Embedding for Database (OLE-DB) is database middleware developed with the goal of adding object-oriented functionality for access to relational and nonrelational data. ActiveX Data Objects (ADO) provides a high-level, application-oriented interface to interact with OLE-DB, DAO, and RDO. Based on ADO, ADO.NET is the data access component of Microsofts' .NET application development framework. Java Database Connectivity (JDBC) is the standard way to interface Java applications with data sources.

- Database access through the Web is achieved through middleware. To improve the capabilities on the client side of the Web browser, you must use plug-ins and other client-side extensions such as Java and JavaScript, or ActiveX and VBScript. On the server side, Web application servers are middleware that expand the functionality of Web servers by linking them to a wide range of services, such as databases, directory systems, and search engines.

- Extensible Markup Language (XML) facilitates the exchange of B2B and other data over the Internet. XML provides the semantics that facilitate the exchange, sharing, and manipulation of structured documents across organizational boundaries. XML produces the description and representation of data, thus setting the stage for data manipulation in ways that were not possible before. XML documents can be validated through the use of document type definition (DTD) documents, and XML schema definition (XSD) documents.

- Cloud computing is a computing model that provides ubiquitous, on-demand access to a shared pool of configurable resources that can be rapidly provisioned. SQL data services (SDS) refers to a cloud computing-based data management service that provides relational data storage, ubiquitous access, and local management to companies of all sizes. This service enables rapid application development for businesses with limited information technology resources. SDS allows rapid deployment of business solutions using standard protocols and common programming interfaces.

Content

11.1 Database Connectivity

Database connectivity - refers to the mechanisms through which application programs connect and communicate with data repositories.

Database middleware - Database connectivity software is also known as database middleware, because it provides an interface between the application program and the database.

Data repository (data source) - represents the data management application, such as Oracle RDBMS, SQL Server, or IBM DBMS, that will be used to store the data.

A standard database connectivity interface is necessary for enabling applications to connect to data repositories.

Universal Data Access (UDA) - a collection of technologies used to access any type of data source and manage the data through a common interface. OBDC, OLE-DB, ADO.NET. (Microsoft).

11.1.1 Native SQL Connectivity

Most DBMS vendors provide their own methods for connecting to their databases. Native SQL connectivity refers to the connection interface that is provided by the database vendor, and is unique to that vendor. Best example is Oracles' SQL*Net interface for the Oracle RDBMS.

Most current DBMS products support other database connectivity standards, ODBC is the most common.

11.1.2 ODBC, DAO, and RDO

Open Database Connectivity - (ODBC) is a superset of the SQL Access Group **Call Level Interface** (CLI) standard for database access. ODBC is probably the most widely supported database connectivity interface. It allows any Windows application to access relational data sources, using SQL via a standard API. Does not provide much functionality beyond its ability to execute SQL to manipulate relational data, need better ways.

Data Access Objects (DAO) - is an object-oriented API used to access MS Access, MS FoxPro, and DBase databases (using the Jet data engine) from Visual Basic programs. It provides an optimized interface that exposes the functionality of the Jet data engine. Can also be used to access other relational-style data sources.

Remote Data Objects (RDO) - is a higher level, object-oriented application interface used to access remote database servers. RDO uses the lower-level DAO and ODBC for direct access to databases. RDO is optimized to deal with server-based databases such as MS SQL Server, Oracle, and DB2.

11.1.3 OLE-DB

ODBC, DAO, and RDO do not provide support for nonrelational data.

Object Linking and Embedding for Database (OLE-DB) - is database middleware that adds object-oriented functionality for access to relational and non-relational data. It is based on Microsofts' Component Object Model (COM).

OLE-DB does not provide support for scripting languages used for Web development, such as Active Server pages (ASP) and ActiveX.

ActiveX Data Objects (ADO) - provides a high-level, application-oriented interface to interact with OLE-DB, DAO, and RDO. ADO provides a unified interface to access data from any programming language that uses the underlying OLE-DB objects. Although ADO is an improvement over the OLE-DB model, Microsoft is encouraging the use of its newer data access framework, ADO.NET.

11.1.4 ADO.NET

Based on ADO, ADO.NET is the data access component of Microsofts .NET application development framework. The .NET framework extends and enhances the functionality provided by the ADO / OLE-DB duo. ADO.NET introduced two new features that are critical for the development of distributed applications: DataSets and XML support.

DataSet - is a disconnected, memory-resident representation of the database. The DataSet contains tables, columns, rows, relationships, and constraints. Once the data are read from the data provider, they are placed in a memory-resident DataSet, which is then disconnected form the data provider. The data consumer application interacts with the data in the DataSet object to make inserts, updates, and deletes in the DataSet. Once the processing is done, the DataSet is synchronized with the data source and the changes are made permanent.

The DataSet is stored internally in XML format, and the data in the DataSet can be made persistent as XML documents. This is critical in todays' distributed environments.

11.1.5 Java Database Connectivity (JDBC)

Java is an object-oriented programming language that runs on top of web browser software. When Java applications need to access data outside the Java run-time environment, they use predefined application programming interfaces.

Java Database Connectivity (JDBC) - an application programming interface that allows a Java program to interact with a wide range of data sources, including relational databases, tabular data sources, spreadsheets, and text files. JBDC allows a Java program to establish a connection with a data source, prepare and sent the SQL code to the database server, and process the result. JDBC also provides a way to connect to databases through an ODBC driver.

11.2 Database Internet Connectivity

Characteristics of Internet technologies -

- Hardware and software independence;
- Common and simple user interface;
- Location independence;
- Rapid development at manageable costs.

The simplicity of the Webs' interface and its cross-platform functionality are at the core of its success as a data access platform.

11.2.1 Web-to-Database Middleware: Server-side Extensions

In general the, the Web server is the main hub through which all Internet services are accessed. Dynamic Web pages are at the heart of current Websites, but neither Web browsers nor Web servers can handle dynamic requests, eg: database queries. Web servers capability must be extended so it can understand and process database requests.

Server-side extension (Web-to-database middleware) - is a program that interacts directly with the Web server to handle specific types of requests. It provides its services to the Web server in a way that is totally transparent to the client browser. Server-side extension adds significant functionality to the Web server, and therefore to the Internet.

11.2.2 Web Server Interfaces

Extending Web server functionality implies that the Web server and the Web-to-database middleware will properly communicate with each other. A Web server interface defines a standard way to exchange messages with external programs. There are two well defined Web server interfaces -

- Common Gateway Interface (CGI);
- Application programming interface (API),

Common Gateway Interface (CGI) - uses script files that perform specific functions based on the clients' parameters that are passed to the Web server. Main disadvantage is that the script file is an external program that executes separately for each user request, and therefore causes a resource bottleneck. Performance can also be degraded by using an interpreted language, or an inefficient script.

Application programming interface (API) - is a newer Web server interface standard that is more efficient and faster than a CGI script. They are implemented as shared code or dynamic-link libraries (DLLs), which means that the API is treated as part of the web server program and dynamically invoked when needed.

APIs are faster than CGI scripts because the code resides in memory, and there is no need to run an external program for each request. They can also use a shared connection to the database instead of creating a new one each time.

Disadvantages - APIs share the same memory space as the Web server, so an API error can bring down the Web server. APIs are specific to the Web server and the operating system.

11.2.3 The Web Browser

Web Bowser is software such as Internet Explorer, Chrome, or Firefox that lets end users navigate the Web from their client computer. The Web Browsers job is to interpret the HTML code it receives from the Web server and to present the various page components in a standard formatted way.

11.2.4 Client-side Extensions

Client-side extensions - add functionality to the Web browser. Most common are -

Plug-in - an external application that is automatically invoked by the browser when needed. The plug-in is associated with a data object - generally using the file extension - to allow the Web server to handle data that are not originally supported.

JavaScript - is a scripting language that allows Web authors to design interactive sites. It is embedded in the Web page, and executed after a specific event.

ActiveX - specification for writing programs that run in Internet Explorer. Extends the Web browser by adding controls to Web pages. Has low portability, Windows specific.

VBScript - derived from Visual Basic. Scripting language similar to JavaScript.

11.2.5 Web Application Servers

Web application server - is a middleware application that expands the functionality of Web servers by linking them to a wide range of services such as databases, directory systems, and search engines.

Examples are: ColdFusion (Adobe), WebLogic Server (Oracle), Visual Studio .NET (Microsoft).

11.2.6 Web Database Development

Web database development deals with the process of interfacing databases with the Web browser - how to create Web pages that access data in a database.

Two examples - ColdFusion, and PHP.

11.3 Extensible markup language (XML)

Extensible Markup Language (XML) - is a metalanguage used to represent and manipulate data elements. XML is designed to facilitate the exchange of structured documents, such as orders and invoices, over the internet. It is the data exchange standard for e-commerce applications.

Characteristics of XML -

- allows the definition of new tags to describe data elements;
- is case sensitive;
- must be well formed, tags must be properly formatted;
- must be properly nested;
- comments marked by <-- and --> symbols;
- XML prefix reserved for XML only.

XML is not a replacement for HTML. XML is concerned with the description and representation of the data, rather than the way the data is displayed.

11.3.1 Document Type Definitions (DTD) and XML Schemas

Document type definition (DTD) - is a file with a .dtd extension that describes XML elements. In effect a DTD file provides the composition of the databases' logical model and defines the syntax rules or valid elements for each type of XML document.

Companies that intend to engage in e-commerce transactions must develop and share DTDs. A DTD only provides descriptive information for understanding how the elements - root, parent, child etc - relate to one another, and limited additional semantic value such as data type support or data validation rules.

XML Schema - is an advanced data definition language that is used to describe the structure of XML data documents. This structure includes elements, data types, relationship types, ranges, and default values. One of the main advantages of an XML schema is that it more closely maps to database terminology and features.

11.3.2 XML Presentation

One of the main benefits of XML is that it separates data structure from its presentation and processing. By separating the two you can present the same data in different ways.

Extensible Style Language (XSL) - specification provides the mechanism to display XML data. Divided into two parts -

- Extensible Style Language Transformations (XSLT) - describes the general mechanism that is used to extract and process data from one XML document and enable its transformation within another document;

- XSL style sheets - define the presentation rules applied to XML elements.

11.3.3 XML Applications

Uses of XML -

- B2B exchanges - enables the exchange of B2B data;
- Legacy systems integration - integrate legacy systems data with modern e-commerce Web systems;
- Web page development - XML is a good fit for some types of Web development scenarios. Eg: Web portals with large amounts of data can use XML to pull data from multiple external sources;
- Database support - DBMS that supports XML can integrate with external systems such as the Web, mobile data, and legacy systems;
- Database metadictionaries - create vocabularies for entire industries;
- XML databases - most databases support XML to manage data in some shape or form;
- XML services - XML provides the infrastructure that helps heterogeneous systems work together.

11.4 Cloud Computing Services

Cloud computing - is "a computing model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computer resources (networks, servers, storage, applications etc) that can be rapidly provisioned and released with minimal management or service provider interaction".

Cloud services - refers to the services provided by cloud computing.

Cloud computing is important for database technologies because it has the potential to become a "game changer". Cloud computing eliminates financial and technological barriers so organizations can leverage database technologies in their business process with minimal effort and cost. Instead of spending large amounts of cash buying hardware and software, organizations can employ a pay-per-use model for their IT services.

11.4.1 Cloud Implementation Types

Different types of implementations -

- **Public cloud** - built by third-party organization to sell cloud services to the general public. Managed exclusively by the third-party provider;
- **Private cloud** - built by an organization for the sole purpose of servicing its own needs. Used by large geographically dispersed organizations to add agility and flexibility to internal IT services. Managed by internal IT staff, or an external third-party;
- **Community cloud** - built by and for a specific group of organizations that share a common trade. Managed by internal IT staff, or by an external third-party.

11.4.2 Characteristics of Cloud Services

- Ubiquitous access via Internet technologies - Basic requirement that device has access to the Internet;
- Shared infrastructure - Cloud service infrastructure is shared by multiple users;
- Lower costs and variable pricing - Initial costs of using cloud services tend to be significantly lower than building on-premises IT infrastructure;
- Flexible and scalable services - Cloud services are built on an infrastructure that is highly scalable , fault tolerant and very reliable. Services can scale up and down on demand;
- Dynamic provisioning - User can quickly provision any needed resources, by accessing the Web management dashboard and then adding or removing services on demand;
- Service orientation - Cloud computing focuses on providing consumers with specific, well-defined services that use well-known interfaces. Hide the complexity from the end user, and can be delivered anytime anywhere;
- Managed operations - The system infrastructure is managed by the cloud provider, minimizes the need for extensive and expensive in-house IT staff.

11.4.3 Types of Cloud Services

- **Software as a Service** (SaaS) - The cloud service provider offers turnkey applications that run in the cloud. Application shared among users from multiple organizations. Eg: MS Office Live, Google Docs;
- **Platform as a Service** (PaaS) - The cloud service provider offers the capability to build and deploy consumer-created applications using the providers' cloud infrastructure. Eg: Microsoft Azure platform, Google Application Engine;
- **Infrastructure as a Service** (IaaS) - The cloud service provider offers consumers the ability to provision their own resources on demand. These include storage, servers, databases etc. The consumer can add or remove the resources as needed.

11.4.4 Cloud Services: Advantages and Disadvantages

ADVANTAGE	DISADVANTAGE
Low initial cost of entry - compared to alternative of building in-house.	**Issues of security, privacy and compliance** - Trusting sensitive company data to external entities.
Scalability / elasticity - Easy to add and remove resources on demand.	**Hidden costs of implementation and operation** - Hard to estimate bandwidth & data migration costs.
Support for mobile computing - Support multiple types of mobile computing devices.	**Data migration difficult and lengthy process** - Migrating large amounts of data can be difficult and time consuming.
Ubiquitous access - Can access the cloud resources from anywhere at any time.	**Complex licensing schemes** - Complicated licensing and service-level agreements.
High reliability and performance - Solid infrastructures.	**Loss of ownership and control** - No longer in complete control of data.

Fast provisioning - Resources can be provisioned on demand in a matter of minutes.	**Organization culture** - End users tend to be resistant to change.
Managed infrastructure - Managed by dedicated staff, allows organizations staff to focus on other areas.	**Difficult integration with internal IT system** - Configuring the cloud services to integrate transparently with internal services could be difficult.

11.4.5 SQL Data Services

Databases remain at the centre of all system development. Cloud computing brings a new dimension to data management, which is within reach of any type of organization.

SQL data services (SDS) - refers to a cloud computing-based data management service that provides relational data storage, access, and management to companies of all sizes without the typically high costs of in-house hardware, software, infrastructure, and personnel.

Advantages of SQL data services -

- Highly reliable and scalable relational database for a fraction of the cost;
- High level of failure tolerance;
- Dynamic and automatic load balancing;
- Automated data backup and disaster recovery included with the service;
- Dynamic creation and allocation of database processes and storage.

Key Terms

- **ActiveX** - Microsofts' alternative to Java. A specification for writing programs that will run inside the Microsoft client browser, Internet Explorer. Oriented mainly to Windows applications, it is not portable. It adds controls such as dropdown windows and calendars to Web pages.

- **ActiveX Data Objects** (ADO) - A Microsoft object framework that provides a high-level, application-oriented interface to OLE-DB, DAO, and RDO. ADO provides a unified interface to access data from any programming language that uses the underlying OLE-DB objects.

- **application programming interface** (API) - Software through which programmers interact with middleware. An API allows the use of generic SQL code, thereby allowing client processes to be database server-independent.

- **Call Level Interface** (CLI) - A standard developed by the SQL Access Group for database access.

- **client-side extensions** - Extensions that add functionality to a Web browser. The most common extensions are plug-ins, Java, JavaScript, ActiveX, and VBScript.

- **cloud computing** - A computing model that provides ubiquitous, on-demand access to a shared pool of configurable resources that can be rapidly provisioned.

- **cloud services** - The services provided by cloud computing. Cloud services allow any organization to quickly and economically add information technology services such as applications, storage, servers, processing power, databases, and infrastructure.

- **Common Gateway Interface** (CGI) - A Web server interface standard that uses script files to perform specific functions based on a clients' parameters.

- **community cloud** - A type of cloud built by and for a specific group of organizations that share a common trade, such as agencies of the federal government, the military, or higher education.

- **Data Access Objects** (DAO) - An object-oriented application programming interface used to access MS Access, MS FoxPro, and dBase databases from Visual Basic programs. DAO provides an optimized programming interface that exposes the functionality of the Jet data engine, on which MS Access is based. The DAO interface can be used to access other relational-style data sources.

- **data source name** (DSN) - A name that identifies and defines an OBDC data source.

- **database middleware** - Database connectivity software through which application programs connect and communicate with data repositories.

- **DataSet** - In ADO.NET, a disconnected, memory-resident representation of the database. The DataSet contains tables, columns, rows, relationships, and constraints.

- **document type definition** (DTD) - A file with a .dtd extension that describes XML elements; in effect, a DTD file describes a documents' composition and defines the syntax rules or valid tags for each type of XML document.

- **dynamic-link libraries** (DLL's) - Shared code modules that are treated as part of the operating system or server process so they can be dynamically invoked at run time.

- **Extensible Markup Language** (XML) - A metalanguage used to represent and manipulate data elements. Unlike other markup languages, XML permits the manipulation of a documents' data elements. XML facilitates the exchange of structured documents such as orders and invoices over the internet.

- **Infrastructure as a Service** (IaaS) - A model in which the cloud service provider offers consumers the ability to provision their own resources on demand; these resources include storage, servers, databases, processing units, and even a complete virtualized desktop.

- **Java** - An object-oriented programming language developed by Sun Microsystems that runs on top of the Web browser software. Java applications are compiled and stored on the Web server. Javas' main advantage is its ability to let application developers create their applications once and then run them in many environments.

- **Java Database Connectivity** (JDBC) - An application programming interface that allows a Java program to interact with a wide range of data sources, including relational databases, tabular data sources, spreadsheets, and text files.

- **JavaScript** - A scripting language developed by NetScape that allows Web authors to design interactive Websites. JavaScript code is embedded in Web pages, and then downloaded with the page and activated when a specific event takes place, such as a mouse click on an object.

- **Microsoft .NET framework** - A component-based platform for the development of distributed, heterogeneous, interoperable applications aimed at manipulating any type of data over any network regardless of operating system and programming language.

- **Object Linking and Embedding for Database** (OLE-DB) - Based on MicroSofts' Component Object Model (COM), OLE-DB is database middleware that adds object-oriented functionality for accessing relational and non-relational data. OLE-DB was the first part of MicroSofts' strategy to provide a unified object-oriented framework for the development of next-generation applications.

- **Open Database Connectivity** (ODBC) - Database middleware developed by MicroSoft to provide a database access API to Windows applications.

- **Platform as a Service** (PaaS) - A model in which the cloud service provider can build and deploy consumer-created applications using the providers' cloud infrastructure.

- **plug-in** - In the World Wide Web (WWW), a client-side external application that is automatically invoked by the browser when needed to manage specific type of data.

- **private cloud** - A form of cloud computing in which an internal cloud is built by an organization to serve its own needs.

- **public cloud** - A form of computing in which the cloud infrastructure is built by a third-party organization to sell cloud services to the general public.

- **Remote Data Objects** (RDO) - A high-level, object-oriented application interface used to access remote database servers. RDO uses the lower-level DAO and ODBC for direct access to databases. RDO was optimized to deal with server-based databases such as MS SQL Server, Oracle, and DB2.

- **script** - A programming language that is not compiled, but is interpreted and executed at run time.

- **server-side extension** - A program that interacts directly with the server process to handle specific types of requests. Server-side extensions add significant functionality to Web servers and intranets.

- **Software as a Service** (SaaS) - A model in which the cloud service provider offers turnkey applications that run in the cloud.

- **SQL data services** (SDS) - Data management services that provide relational data storage, access, and management over the internet.

- **stateless system** - A system in which a Web server does not know the status of the clients communicating with it. The Web does not reserve memory to maintain an open communication state between the client and the server.

- **tags** - In markup languages such as HTML and XML, a command inserted in a document to specify how the document should be formatted. Tags are used in server-side markup languages and interpreted by a Web browser for presenting data.

- **Universal Data Access** (UDA) - Within the Microsoft application framework, a collection of technologies used to access any type of data source and manage the data through a common interface.

- **VBScript** - A client-side extension in the form of a Microsoft product that extends a browsers' functionality; VBScript is derived from Visual Basic.

- **Web application server** - A middleware application that expands the functionality of web servers by linking them to a wide range of services, such as databases, directory systems, and search engines.

- **Web-to-database middleware** - A database server-side extension that retrieves data from databases and passes them to the Web server, which in turn sends the data to the clients' browser for display.

- **XML schema** - An advanced data definition language used to describe the elements, data types, relationship types, ranges, and default values of XML data documents. One of the main advantages of an XML schema is that it more closely maps to database terminology and features.

- **XML schema definition** (XSD) - A file that contains the description of an XML document.

--ooOoo--

Chapter 12 - Database Administration and Security

Summary

- Data management is a critical activity for any organization, so data must be treated as a corporate asset. The value of a data set is measured by the utility of the information derived from it. Good data management is likely to produce good information, which is the basis for better decision making.
- Data quality is a comprehensive approach to ensure the accuracy, validity, and timeliness of data. Data quality focuses on correcting dirty data, preventing future inaccuracies in the data, and building user confidence in the data.
- The DBMS is the most commonly used tool for corporate data management. The DBMS supports strategic, tactical, and operational decision making at all levels of the organization. The introduction of a DBMS into an organization is a delicate job; the impact of the DBMS on the organizations' managerial and cultural framework must be carefully examined.
- The database administrator (DBA) is responsible for managing the corporate database. The internal organization of database administration varies from company to company. Although no standard exists, it is common practice to divide DBA operations according to phases of the Database Life Cycle. Some companies have created a position with a broader mandate to manage computerized data and other data; this activity is handled by the data administrator (DA).
- The DA and DBA functions tend to overlap. Generally speaking, the DA has more managerial tasks than the more technically oriented DBA. Compared to the DBA function, the DA function is DBMS-independent, with a broader and longer-term focus. However, when the organization does not include a DA position, the DBA executes all of the DAs' functions. In this combined role, the DBA must have a diverse mix of technical and managerial skills.
- A DBAs' managerial services include supporting end users; defining and enforcing policies, procedures and standards for the database; ensuring data security, privacy, and integrity; providing data backup and recovery services; and monitoring distribution and use of the data in the database.
- The DBAs' technical role includes involvement in at least the following activities: evaluating, selecting, and installing the DBMS; designing and implementing databases and applications; testing and evaluating databases and applications; operating and maintaining the DBMS, utilities, and applications; and training and supporting users.
- Security refers to activities and measures that ensure the confidentiality, integrity, and availability of an information system and its main asset, data. A security policy is a collection of standards, policies, and practices that guarantee the security of a system and ensure auditing and compliance.

- A security vulnerability is a weakness in a system component that could be exploited to allow unauthorized access or service disruption. A security threat is an imminent security violation caused by an unchecked vulnerability. Security vulnerabilities exist in all components of an information system: people, hardware, software, network, procedures, and data. Therefore, it is critical to have robust database security. Database security refers to DBMS features and related measures that comply with the organizations' security requirements.

- The development of a data administration strategy is closely related to the companys' mission and objectives. Therefore, the strategic plan requires a detailed analysis of company goals, its situation, and its business needs. To guide the development of this data administration plan, an integrating methodology is required. The most commonly used integrating methodology is known as information engineering (IE).

- To help translate strategic plans into operational plans, the DBA has access to an arsenal of database administration tools, including a data dictionary and computer-aided systems engineering (CASE) tools.

Content

12.1 Data as a Corporate Asset

Data are the raw material from which information is produced, and a valuable resource. If the information is accurate and timely, it can enhance the companys' competitive position and generate wealth. Data form the basis for decision making, strategic planning, control, and operations monitoring.

Efficient asset management is critical to the success of an organization. To manage data as a corporate asset, managers must understand the value of information. As organizations become more dependent on information, its accuracy becomes more critical.

Dirty data - data that suffers from inaccuracies and inconsistencies.

Data can become dirty for several reasons -

- Lack of enforcement of integrity constraints, such as not null, uniqueness, and referential integrity;
- Data-entry errors and typographical errors;
- Use of synonyms and homonyms across systems;
- Non-standard use of abbreviations in character data;
- Different decompositions of composite attributes into simple attributes across systems.

Data quality - is a comprehensive approach to ensuring the accuracy, validity, and timeliness of data. This comprehensive approach is important, because data quality involves more than just cleaning dirty data; it also focuses on preventing future inaccuracies and building user confidence in the data.

Data quality efforts involve the following -

- A data governance structure that is responsible for data quality;
- Measurements of current data quality;
- Definition of data quality standards in alignment with business goals;
- Implementation of tools and processes to ensure future data quality.

A number of tools can assist in data quality initiatives - Eg: Data-profiling software, and Master data management software. While these tools provide an important part of data quality, the overall solution to high-quality data within an organization still relies heavily on data administration and management.

12.2 The Need for a Database and its Role in an Organization

Regardless of the organization, the predominant role of the database is to support managerial decision making at all levels in the organization, while preserving data privacy and security.

The DBMS must give each level of management a useful view of the data and support the required level of decision making. Typical activities of each management level -

- Top management level (Strategic decisions) -
    - Provide the information necessary for strategic decision making, strategic planning, policy formulation, and goals definition.
    - Provide access to external and internal data to identify growth opportunities and to chart the direction of such growth.
    - Provide a framework for defining and enforcing organizational policies that are translated into business rules at lower levels in the organization.

  - Improve the likelihood of a positive return on investment by searching for new ways to reduce costs and boost productivity in the company.
  - Provide feedback to monitor whether the company is achieving its goals.
- Middle management level (Tactical decisions) -
  - Deliver the data necessary for tactical decisions and planning.
  - Monitor and control the allocation and use of company resources and evaluate the performance of various departments.
  - Provide a framework for enforcing and ensuring the security and privacy of the data in the database.
- Operational management level (Daily working decisions) -
  - Represent and support company operations as closely as possible.
  - Produce query results within specified performance levels.
  - Enhance the companys' short-term operations by providing timely information for customer support and for application development and computer operations.

12.3 Introduction of a Database: Special Considerations

The introduction of a DBMS is a process that includes three important aspects -

- **Technological** (DBMS software and hardware) - includes selecting, installing configuring, and monitoring the DBMS. Database administration staffing is a key technological consideration, personnel must have appropriate technical and managerial skills.
- **Managerial** (Administrative functions) - must create an appropriate organizational structure to accommodate the personnel responsible for administering the system.
- **Cultural** (Corporate resistance to change) - The DBMS is likely to have an effect on people, functions, and interactions.

12.4 The Evolution of Database Administration

Database Administrator (DBA) - Person responsible for control of the centralized and shared database.

DBA operations are commonly defined and divided according to the phases of the Database Life Cycle (DBLC). Requires personnel to cover following activities -

- Database planning, including definition of standards, procedures, and enforcement;
- Database requirements gathering and conceptual design;
- Database logical and transaction design;
- Database physical design and implementation;
- Database testing and debugging;
- Database operations and maintenance, including installation, conversion, and migration;
- Database training and support;
- Data quality monitoring and management.

Data administrator (DA) (Information resource manager IRM) - Growing trend toward specialization in data management. Distinction between DBA and DA. The DA reports directly to top management, with a higher degree of responsibility and authority than the DBA. Responsible for controlling the overall corporate data sources, both computerized and manual.

12.5 The Database Environments' Human Component

The DBA must perform two distinct roles -

- Managerial role - focused on personnel management and on interactions with end users.
- Technical role - involves the use of the DBMS - database design, development, and implementation - as well as the production, development, and use of application programs.

12.5.1 The DBAs' Managerial Role

As a manager, the DBA must concentrate on the control and planning of database administration. Therefore the DBA is responsible for -

- **End-User Support** - Support services include the following -
  - Gathering user requirements - Must work with end users to help gather data required to identify and describe their present and future information needs. Varying computer backgrounds and communication styles.
  - Building end-user confidence - Educate end users about the services provided.
  - Resolving conflicts and problems - DBA must have authority and responsibility to resolve conflicts with regard to data requirements.
  - Finding solutions to information needs - Need to develop solutions that will properly fit within the data management framework and address end users information needs.
  - Ensuring quality and integrity of data and applications - Once solution has been found, it must be properly implemented. Must teach application programmers and end users the database standards and procedures required for data quality, access, and manipulation.
  - Managing the training and support of DBMS users - One of the most time consuming activities is teaching end users how to use the database.
- **Policies, Procedures, and Standards** - A successful data administration strategy requires the continuous enforcement of policies, procedures, and standards for correct data creation, usage, and distribution within the database.
  - **Policies** - general statements of direction or action that communicate and support DBA goals.
    - Examples -
      - all users must have passwords;
      - passwords must be changed every six months.
  - **Standards** - describe the minimum requirements of a given DBA activity; they are more detailed and specific than policies. They are rules that evaluate the quality of an activity.

        - Examples -
            - A password must have a minimum of five characters;
            - A password must have a maximum of 12 characters;
            - ID numbers, names, birth dates cannot be used in passwords.

    - **Procedures** - are written instructions that describe a series of steps to be followed during the performance of a given activity. Must define, communicate, and enforce procedures that cover -
        - End-user data requirements gathering;
        - Database design and modelling;
        - Documentation and naming conventions;
        - Design, coding, and testing of database application programs;
        - Database software selection;
        - Database security and integrity;
        - Database backup and recovery;
        - Database maintenance and operation;
        - End-user training.

- **Data Security, Privacy, and Integrity** - The distribution of data across multiple sites has made it more difficult to maintain database control, security, and integrity. The DBA must enforce the database administration policies defined for the organization.

- **Data Backup and Recovery** - Data backup and recovery procedures are critical in all database installations. The DBA must ensure that data can be fully recovered in case of data loss, or loss of database integrity.
    - **Database Security Officer** - sole job is to ensure database security and integrity. Also referred to as Disaster Management.
    - **Disaster Management** - includes all of the DBA activities designed to secure data availability following a physical disaster or a database integrity failure. Includes all planning, organizing, and testing of database contingency plans and recovery procedures. Backup and recovery procedures must include at least the following -
        - Periodic data and application backups - Different types of backup -
            - **Full backup** (Database dump) - complete copy of the entire database;
            - **Incremental backup** - backup of all data since the last backup date;
            - **Concurrent backup** - takes place while the user is working on the database.
        - Proper backup identification - Backups must be clearly identified through detailed descriptions and date information, to ensure that the correct backups are used to recover the database.

    - Convenient and safe backup storage - Multiple backups of the same data are required, and each backup copy must be stored in a different location.

    - Physical protection of both hardware and software - Protection might include the use of closed installations with restricted access, as well as preparation of the computer sites to provide air conditioning, backup power, and fire protection.

    - Personal access control to the software of a database installation - Multilevel passwords and privileges.

    - Insurance coverage for the data in the database - Should have insurance cover to provide financial protection in the event of a database failure.

- **Data Distribution and Use** - Data are only useful when they reach the right users at the right time, and in the right format.

12.5.2 The DBAs' Technical Role

The DBAs technical role requires a broad understanding of DBMS functions, configuration, programming languages, and data-modeling and design methodologies.

- **Evaluating, Selecting, and Installing the DBMS and Utilities** - Most important technical responsibility is selecting the database management system, utility software, and supporting hardware. Checklist of desired DBMS features -

  - DBMS model - relational , object-oriented, object / relational ?
  - DBMS storage capacity - maximum storage and database sizes ?
  - Application development support - Programming languages, application development tools ?
  - Security and integrity - Referential and entity integrity rules, access rights supported ?
  - Backup and recovery - Automated backup and recovery tools ?
  - Concurrency control - Multiple users supported ?
  - Performance - How many transactions per second does the DBMS support ?
  - Database administration tools - Does the DBMS offer DBA management interface ?
  - Inteoperability and data distribution - Can the DBMS work with other DBMS types ?
  - Portability and standards - Can the DBMS run on different operating systems and platforms ?
  - Hardware - What hardware does the DBMS require ?
  - Data dictionary - Does the DBMS have a data dictionary ?
  - Vendor training and support - What support does vendor offer ? Documentation ?
  - Available third party tools - offered by third party vendors ?
  - Costs - What are costs of hardware and software ? Staff costs ?

- **Designing and Implementing Databases and Applications** - DBA also provides data-modeling and design services to end users. Must ensure that standards and procedures are adhered to.

- **Testing and Evaluating Databases and Applications** - DBA must also provide testing and evaluation services for all database and end-user applications. Evaluation process covers the following -

    - Technical aspects of both the applications and the database; backup and recovery, security and integrity, use of SQL, and application performance.

    - Evaluation of written documentation and procedures.

    - Observance of standards for naming, documenting, and coding.

    - Checking for data duplication conflicts with existing data.

    - The enforcement of all data validation rules.

- **Operating the DBMS, Utilities, and Applications** - DBMS operations can be divided into four main areas -

    - <u>System Support</u> - all tasks directly related to the day-to-day operations of the DBMS, including filling out job logs, changing tape, and verifying the status of hardware and emergency power sources.

    - <u>Performance monitoring and tuning</u> - are very time-consuming. Must ensure that the DBMS, utilities, and applications maintain satisfactory performance levels. The DBA must -

        - Establish DBMS performance goals;

        - Monitor the DBMS to evaluate whether the performance objectives are being met;

        - Isolate the problem and find solutions if performance objectives are not met;

        - Implement the selected performance solutions.

    - <u>Backup and recovery</u> - activities are of primary concern because data loss could be devastating to an organization. The DBA must establish a schedule for backing up database and log files at appropriate intervals, and must plan, implement, test and enforce backup recovery procedures.

    - <u>Security auditing and monitoring</u> - involves creating users, assigning access rights, and using SQL commands to grant and revoke access rights to users and database objects. DBA must also monitor actual or attempted security violations, and take remedial action.

- **Training and Supporting Users** - Training users and application programmers to use the DBMS and its tools and utilities are part of the DBAs technical activities. This includes the procedures and standards required for database programming.

- **Maintaining the DBMS, Utilities, and Applications** - Maintenance activities are dedicated to the preservation of the DBMS environment. One of the most common maintenance activities is reorganizing the physical location of data in the database. Also includes upgrading the DBMS and utility software.

12.5.3 The DBAs' Role in the Cloud

The use of cloud-based data services has a significant impact on the role of the DBA, with a reduced role in installing and maintaining the DBMS. The DBAs technical role is still critical to the organization, and the managerial role is largely unchanged.

12.6 Security

Security - refers to activities and measures that ensure the confidentiality, integrity, and availability of an information system, and its main asset, data. Must secure all the processes and systems around the data, including hardware systems, software applications, the network and its devices, internal and external users, procedures, and the data itself.

- **Confidentiality** - deals with ensuring that data are protected against unauthorized access, and if the data is accessed by an authorized user, that the data are used only for an authorized purpose. Data must be evaluated and classified according to the level of confidentiality: highly restricted (very few people have access), confidential (only certain groups have access), and unrestricted (can be accessed by all users).

- **Integrity** - is concerned with keeping data consistent and free of errors or anomalies. Data should be treated as the most valuable asset in an organization and rigorous data validation carried out at all levels.

- **Availability** - refers to the accessibility of data whenever required by an authorized user, and for authorized purposes. To ensure data availability, the entire system must be protected from service degradation or interruption caused by any internal or external source. System availability is an important goal of security.

12.6.1 Security Policies

Security Policy - is a collection of standards, policies, and procedures created to guarantee the security of a system and ensure auditing compliance.

12.6.2 Security Vulnerabilities

Security vulnerability - is a weakness in a system component that could be exploited to allow unauthorized access, or cause service disruptions. Categories -

- Technical - A flaw in the operating system or browser;

- Managerial - An organization may not educate users about security issues;

- Cultural - Users write down passwords, or do not shred confidential documents;

- Procedural - Company procedures may not require secure passwords, or checking of user IDs.

Security threat - is an imminent security violation. If a security vulnerability is left unresolved, it could become a security threat.

Security breach - occurs when a security threat is exploited to endanger the integrity, confidentiality, or availability of the system. Can lead to a database with integrity either *preserved*, or *corrupted* -

- Preserved - Action required to avoid similar recurrence, but data recovery not required.
- Corrupted - Action required to avoid similar recurrence, and the database must be recovered to a consistent state. Include database access by computer viruses, or by hackers destroying or altering data.

12.6.3 Database Security

Database security - refers to DBMS features and other related measures that comply with the organizations' security requirements. Recommended security safeguards -

- Change default system passwords;
- Change default installation paths;
- Apply the latest patches;
- Secure installation folders with proper access rights;
- Make sure that only required services are running;
- Set up auditing logs;
- Set up session logging;
- Require session encryption.

Authorization management - defines procedures to protect and guarantee database security and integrity. These procedures include -

- User access management - designed to limit access to the database -
    - **Define each user to the database** - Create a unique user ID for each user;
    - **Assign passwords to each user** - Can be defined with predetermined expiration dates;
    - **Define user groups** - according to common access needs, can help control and manage access privileges of individual users;
    - **Assign access privileges** - grant access to specific databases, may be limited to read-only, or may include read, write, and delete privileges;
    - **Control physical access** - physical security can prevent unauthorised users from directly accessing the DBMS installation and facilities.
- View definition - define data views to protect and control the scope of the data that are accessible to authorised users.
- DBMS access control - Access can be controlled by placing limits on the use of DBMS query and reporting tools.
- DBMS usage monitoring - DBA must audit the use of data in the database. Create an audit log.

12.7 The Database Administration Tools

12.7.1 The Data Dictionary

Data dictionary - A DBMS component that stores the definition of data characteristics and relationships. (Metadata - data about data). The data dictionaries main function is to store the description of all objects that interact with the database. Two types of data dictionaries -

- **Integrated** - included with the DBMS. All relational DBMSs include a built-in data dictionary.
- **Standalone** - Other, older DBMSs, do not have a built-in data dictionary, may use a standalone third party system.

Can also be classified as active or passive -

- **Active data dictionary** - is automatically updated by the DBMS with every database access to keep its access information up to date.
- **Passive data dictionary** - is not updated automatically, and usually requires running a batch process.

There is no standard, but a data dictionary typically stores descriptions of the following -

- Data elements that are defined in all tables of all databases;
- Tables defined in all databases;
- Indexes defined for each database table;
- Defined databases;
- End users and administrators of the database;
- Programs that access the database;
- Access authorizations for all users of all databases;
- Relationships among data elements.

12.7.2 Case Tools

Computer-aided systems engineering (CASE) - tool that provides an automated framework for the Systems Development Life Cycle (SDLC). Classified according to the extent of support they provide for the SDLC -

- **Front-end CASE tools** - provide support for the planning, analysis, and design phases;
- **Back-end CASE tools** - provide support for the coding and implementation phases.

The benefits of CASE tools include -

- A reduction in development time and costs;

- Automation of the SDLC;
- Standardization of system development methodologies;
- Easier maintenance of application systems developed with CASE tools.

Typical CASE tool provides five components -

- Graphics designed to produce diagrams such as data flow diagrams, ER diagrams, class diagrams;
- Screen painters and report generators to produce the information systems input and output formats;
- An integrated repository for storing and cross-referencing the system design data; includes a comprehensive data dictionary;
- An analysis segment to provide a fully automated check on system consistency, syntax, and completeness;
- A program document generator.

12.8 Developing a Data Administration Strategy

Several methodologies are available to ensure the compatibility of data administration and information system plans, and to guide strategic plan development. Most commonly used is information engineering.

Information engineering (IE) - allows for translation of the companys' strategic goals into the data and applications that will help the company achieve those goals. IE focuses on the description of common corporate data instead of the process. Business data types tend to remain stable, but processes change often, and so require frequent modification of existing systems. By placing the emphasis on data, IE helps decrease the impact on systems when processes change.

Information systems architecture (ISA) - is the output of the IE process. It serves as the basis for planning, development, and the control of future information systems. An ISA provides a framework that includes computerized, automated, and integrated tools such as a DBMS and CASE tools.

12.9 The DBA at Work: Using Oracle for Database Administration

The DBA needs to handle the following technical tasks in a specific DBMS -

- Creating and expanding database storage structures;
- Managing database objects such as tables, indexes, triggers, and procedures;
- Managing the end-user database environment, including the type and extent of database access;
- Customizing database initialization parameters.

12.9.1 Oracle Database Administration Tools

In Oracle most DBA tasks are performed via the Oracle Enterprise Manager interface.

12.9.2 The Default Login

To perform any administrative task, you must connect to the database using a username with administrative privileges.

12.9.3 Ensuring that the RDBMS Starts Automatically

Basic task to ensure that database access starts automatically when computer is turned on.

Service - Windows name for a special program that runs automatically as part of the operating system.

12.9.4 Creating Tablespaces and Datafiles

Tablespace - is a logical storage space. A database is *logically* composed of one or more tablespaces.

Datafile - physically stores the databases' data. Each datafile is associated with only one tablespace, although it can reside in different directories or disks.

12.9.5 Managing the Database Objects: Tables, Views, Triggers, and Procedures

Important aspect of managing a database is monitoring the objects that are created in the database.

Database object - any object created by end-users - tables, views, indexes, stored procedures, and triggers.

12.9.6 Managing Users and Establishing Security

One of the most common database administration activities is creating and managing database users.

12.9.7 Customizing the Database Initialization Parameters

Fine-tuning a database is another important DBA task that usually requires the modification of database configuration parameters. Some can be can be changed in real-time using SQL commands, others require the database to be shutdown and restarted.

One of the important functions of the initialization parameters is to reserve the resources that the database uses at run time. Eg: primary memory used for database caching.

Key Terms

- **access plan** - A set of instructions generated at application compilation time that is created and managed by a DBMS. The access plan predetermines how an applications' query will access the database at run time.
- **active data dictionary** - A data dictionary that is automatically updated by the database management system every time the database is accessed, thereby keeping its information current. See also *data dictionary*.
- **audit log** - A security feature of a database management system that automatically records a brief description of the database operations performed by all users.
- **authorization management** - Procedures that protect and guarantee database security and integrity. Such procedures include user access management, view definition, DBMS access control, and DBMS usage monitoring.
- **availability** - refers to the accessibility of data whenever required by authorized users and for authorized purposes. Important goal of security.
- **back-end CASE tools** - A computer-aided software tool that provides support for the coding and implementation phases of the SDLC. In comparison, front-end CASE tools provide support for the planning, analysis, and design phases.
- **CASE** (computer-aided systems engineering) - Tools used to automate part or all of the Systems Development Life Cycle.
- **compliance** - refers to activities that meet data privacy and security reporting guidelines.
- **concurrent backup** - A backup that takes place while one or more users are working on a database.
- **confidentiality** - deals with ensuring that data are protected against unauthorized access, and if the data are accessed by an authorised user, that the data are used only for an authorised purpose.
- **data administrator** (DA) - The person responsible for managing the entire database resource, whether it is computerized or not. The DA has broader authority and responsibility than the database administrator (DBA). Also known as an *information resource manager* (IRM).
- **data-profiling software** - Programs that analyze data and metadata to determine patterns that can help assess data quality.
- **data quality** - A comprehensive approach to ensuring the accuracy, validity, and timeliness of data.
- **database administrator** (DBA) - The person responsible for planning, organizing, controlling, and monitoring the centralized and shared corporate database. The DBA is the general manager of the database administration department.
- **database dump** - See *full backup*.
- **database instance** (Oracle) - In an Oracle DBMS, the collection of processes and data structures used to manage a specific database.
- **database object** (Oracle) - Any object in a database, such as a table, view, index, stored procedure, or trigger.
- **database security** - The use of DBMS features and other related measures to comply with the security requirements of an organization.

- **database security officer** (DSO) - The person responsible for the security, integrity, backup, and recovery of the database.

- **datafile** (Oracle) - A named physical storage space that stores a databases' data. It can reside in a different directory on a hard disk, or on one or more hard disks. All data in a database are stored in data files. A typical enterprise database is normally composed of several data files. A data file can contain rows from one or more tables.

- **disaster management** - The set of DBA activities dedicated to securing data availability following a physical disaster or a database integrity failure.

- **enterprise database** - The overall company data representation, which provides support for present and expected future needs.

- **front-end CASE tools** - A computer-aided software tool that provides support for the planning, analysis, and design phases of the SDLC. In comparison, back-end CASE tools provide support for the coding and implementation phases.

- **full backup** (database dump) - A complete copy of an entire database saved and periodically updated in a separate memory location. A full backup ensures a full recovery of all data after a physical disaster or database integrity failure.

- **incremental backup** - A process that only backs up data that has changed in the database since the last incremental or full backup.

- **information engineering** (IE) - A methodology that translates a companys' strategic goals into helpful data and applications. IE focuses on the description of corporate data instead of the process.

- **information resource dictionary** - See *data dictionary*.

- **information resource manager** (IRM) - See *data administrator* (DA).

- **information systems architecture** (ISA) - The output of the information engineering (IE) process that serves as the basis for planning, developing, and controlling future information systems.

- **information systems** (IS) **department** - An evolution of the data-processing department in which responsibilities are broadened to include service and production functions.

- **integrity** - In a data security framework, refers to keeping data consistent and free of errors or anomalies. See also *data integrity*.

- **master data management** (MDM) **software** - In business intelligence, a collection of concepts, techniques, and processes for the proper identification, definition, and management of data elements within an organization.

- **passive data dictionary** - A DBMS data dictionary that requires a command initiated by an end user to update its data access statistics. See also *data dictionary.*

- **policies** - General statements of direction that are used to manage company operations through the communication and support of the organizations' objectives.

- **privacy** - The rights of individuals and organizations to determine access to data about themselves.

- **procedures** - Series of steps to be followed during the performance of an activity or process.

- **profile** (Oracle) - In Oracle, a named collection of settings that controls how much of the database resource a given user can use.

- **role** (Oracle) - In Oracle, a named collection of database access privileges that authorize a user to connect to a database and use its system resources.

- **schema** (Oracle) - A logical grouping of database objects, such as tables, indexes, views, and queries, that are related to each other. Usually, a schema belongs to a single user or application.

- **security** - Activities and measures to ensure the confidentiality, integrity, and availability of an information system and its main asset, data.

- **security breach** - An event in which a security threat is exploited to endanger the integrity, confidentiality, or availability of the system.

- **security policy** - A collection of standards, policies, and procedures created to guarantee the security of a system and ensure auditing and compliance.

- **security threat** - An imminent security violation that could occur due to unchecked security vulnerabilities.

- **security vulnerability** - A weakness in a system component that could be exploited to allow unauthorized access or cause service disruption.

- **standards** - A detailed and specific set of instructions that describes the minimum requirements for a given activity. Standards are used to evaluate the quality of the output.

- **systems administrator** - The person responsible for co-ordinating an organizations' data-processing activities.

- **tablespace** (Oracle) - In a DBMS, a logical storage space used to group related data. Also known as a *file group*.

- **user** (Oracle) - In a system, a uniquely identifiable object that allows a given person or process to log on to the database.

--ooOoo--

Chapter 13 - Managing Transactions and Concurrency

Summary

- A transaction is a sequence of database operations that access the database. A transaction represents a real-world event, and it must be a logical unit of work; that is, no portion of the transaction can exist by itself. Either all parts are executed or the transaction is aborted. A transaction takes a database from one consistent state to another. A consistent database state is one in which all data integrity constraints are satisfied.

- Transactions have four main properties: atomicity, consistency, isolation, and durability. Atomicity means that all parts of the transaction must be executed; otherwise, the transaction is aborted. Consistency means that the databases' consistent state is maintained, and isolation means that data used by one transaction cannot be accessed by another transaction until the first one is completed. Durability means that changes made by a transaction cannot be rolled back once the the transaction is committed. In addition, transaction schedules have the property of serializability - the result of the concurrent execution of transactions is the same as that of the transactions being executed in serial order.

- SQL provides support for transactions through the use of two statements: COMMIT, which saves changes to disk, and ROLLBACK, which restores the previous database state.

- SQL transactions are formed by several SQL statements or database requests. Each database request originates several I/O database operations.

- The transaction log keeps track of all transactions that modify the database. The information stored in the transaction log is used for recovery (ROLLBACK) purposes.

- Concurrency control coordinates the simultaneous execution of transactions. The concurrent execution of transactions can result in three main problems: lost updates, uncommitted data, and inconsistent retrievals.

- The scheduler is responsible for establishing the order in which the concurrent transaction operations are executed. The transaction execution order is critical and ensures database integrity in multiuser database systems. The scheduler uses locking, timestamping, and optimistic methods to ensure the serializability of transactions.

- A lock guarantees unique access to to a data item by a transaction. The lock prevents one transaction from using the data item while another transaction is using it. There are several levels of locks: database, table, page, row, and field.

- Two types of locks can be used in database systems: binary locks and shared / exclusive locks. A binary lock can have only two states: locked (1) or unlocked (0). A shared lock is used when a transaction wants to read data from a database and no other transaction is updating the same data. Several shared or "read" locks can exist for a particular item. An exclusive lock is issued when a transaction wants to update (write to) the database and no other locks (shared or exclusive) are held on the data.

- Serializability of schedules is guaranteed through the use of two-phase locking. The two-phase locking schema has a growing phase, in which the transaction acquires all of the locks that it needs without unlocking any data, and a shrinking phase, in which the transaction releases all of the locks without acquiring new locks.

- When two or more transactions wait indefinitely for each other to release a lock, they are in a deadlock, also called a deadly embrace. There are three deadlock control techniques: prevention, detection, and avoidance.

- Concurrency control with timestamping methods assigns a unique timestamp to each transaction and schedules the execution of conflicting transactions in a timestamp order. Two schemes are used to decide which transaction is rolled back and which continues executing: the wait / die scheme, and the wound / wait scheme.

- Concurrency control with optimistic methods assumes that the majority of database transactions do not conflict and that transactions are executed concurrently, using private, temporary copies of the data. At commit time, the private copies are updated to the database.

- Database recovery restores the database from a given state to a previous consistent state. Database recovery is triggered when a critical event occurs, such as a hardware error or application error.

Content

13.1 What is a Transaction ?

Transaction - any action that reads from or writes to a database. It is a *logical* unit of work that must be entirely completed or entirely aborted; no intermediate states are acceptable.

A transaction may consist of the following -

- A simple SELECT statement to generate a list of table contents;
- A series of related UPDATE statements to change the values of attributes in various tables;
- A series of INSERT statements to add rows to one or more tables;
- A combination of SELECT, UPDATE, and INSERT statements.

A multicomponent transaction must not be partially completed, all of the SQL statements in the transaction must be completed successfully. If any of the SQL statements fail, the entire transaction is rolled back to the original database state that existed before the transaction started. A successful transaction changes the database from one consistent state to another.

Consistent state - is one in which all data integrity constraints are satisfied.

Database request - is the equivalent of a single SQL statement in an application program or transaction. Most real-world transactions are formed by two or more database requests.

13.1.1 Evaluating Transaction Results

Although the DBMS is designed to recover a database to a previous consistent state when an interruption prevents the completion of a transaction, the transaction itself is defined by the end user or programmer and must be semantically correct.

The DBMS cannot guarantee that the semantic meaning of the transaction truly represents the real-world event. (Even thought the syntax is correct).

13.1.2 Transaction Properties

Each individual transaction must display *atomicity*, *consistency*, *isolation*, and *durability*. Referred to as the ACID test. When executing multiple transactions, the DBMS must schedule the concurrent execution of the transactions' operations. The schedule must exhibit the property of *serializability*.

- **Atomicity** - requires that all operations (SQL requests) of a transaction be completed; if not, the transaction is aborted. A transaction is treated as a single, indivisible, logical unit of work.
- **Consistency** - indicates the permanence of the databases' consistent state. A transaction takes a database from one consistent state to another. If any part of the transaction violates an integrity constraint, the entire transaction is aborted.
- **Isolation** - means that the data used during the execution of a transaction cannot be used by a second transaction until the first one is completed.
- **Durability** - ensures that once transaction changes are done and committed, they cannot be undone or lost, even in the event of system failure.

- **Serializability** - ensures that the schedule for the concurrent execution of transactions yields consistent results. This property is important in multiuser and distributed databases in which multiple transactions are likely to be executed concurrently. Serializability is not an issue if only a single transaction is executed.

Single-user database system automatically ensures serializability and isolation of the database, because only one transaction can be executed at a time. The atomicity, consistency, and durability of transactions must be guaranteed by a single-user DBMS.

Multiuser database systems are subject to multiple concurrent transactions, and must therefore implement controls to ensure serializability and isolation of transactions, in addition to atomicity, consistency, and durability.

13.1.3 Transaction Management with SQL

Transaction support is provided by two SQL statements: COMMIT, and ROLLBACK. When a transaction sequence is initiated, the sequence must continue though all succeeding SQL statements until one of the following four events occurs -

- A COMMIT statement is reached, in which case all changes are permanently recorded within the database. The COMMIT statement automatically ends the SQL transaction.
- A ROLLBACK statement is reached, in which case all changes are aborted and the database is rolled back to its previous consistent state.
- The end of the program is reached. Equivalent to COMMIT.
- The program is abnormally terminated. Equivalent to ROLLBACK.

13.1.4 The Transaction Log

Transaction log - used by a DBMS to keep track of all transactions that update the database. The DBMS uses the information stored in this log for a recovery requirement triggered by a ROLLBACK statement, abnormal termination, or system failure.

The DBMS automatically updates the transaction log while it executes transactions that modify the database. The transaction log stores the following -

- A record for the beginning of the transaction;
- For each transaction component (SQL statement) -
  - The type of operation being performed (INSERT, UPDATE, DELETE);
  - The names of the objects affected by the transaction;
  - The "before" and "after" values for the fields being updated;
  - Pointers to the previous and next transaction log entries for the same transaction;
- The ending (COMMIT) of the transaction.

The transaction log is a critical part of the database, and is usually implemented as one or more files that are managed separately from the actual database files.

13.2 Concurrency Control

Concurrency control - coordinating the simultaneous execution of transactions in a multiuser database system.

Objective is to ensure the serializability of transactions in a multiuser database environment. Important because the simultaneous execution of transactions can create several data integrity and consistency problems. The three main problems are - *lost updates*, *uncommitted data*, and *inconsistent retrievals*.

13.2.1 Lost Updates

Lost update - problem occurs when two concurrent operations, T1 and T2, are updating the same data element and one of the updates is lost (overwritten by the other transaction).

If T1 has not yet been COMMITed when T2 executes, T2 will use the old value on disk, and then overwrite any value that T1 writes to disk in the interim.

13.2.2 Uncommitted Data

Uncommitted data - occurs when two transactions, T1 and T2, are executed concurrently and the first transaction (T1) is rolled back after the second transaction (T2) has already accessed the uncommitted data. This violates the *isolation* property of transactions.

13.2.3 Inconsistent Retrievals

Inconsistent retrievals - occur when a transaction accesses data before and after one or more other transactions finish working with the data.

Eg: An inconsistent retrieval would occur if T1 calculated some summary (aggregate) function over a set of data while T2 was updating the same data. T1 may read some data before they are updated, and other data after they are updated.

13.2.4 The Scheduler

- Severe problems can arise when two or more concurrent transactions are executed;
- Database transactions involve a series of database I/O operations that take the database from one consistent state to another;
- Database consistency can only be ensured before and after the execution of transactions;
- A database always moves through an unavoidable temporary state of inconsistency during a transactions' execution, if the transaction updates multiple tables and rows;
- The temporary inconsistency exists because a computer executes the operations serially, one after another. During this serial process, the isolation property prevents them from accessing data not yet released by other transactions;
- What would happen if two transactions executed concurrently, and they were accessing the same data ? Conflict is possible among the transaction components, and the selection of one execution order over another might have some undesirable consequences.
- How is the correct order determined ?

Scheduler - is a special DBMS process that establishes the order in which the operations are executed within concurrent transactions. The scheduler interleaves the execution of database operations to ensure serializability and isolation of transactions. The scheduler bases its actions on concurrency control algorithms, such as locking or timestamping methods.

Not all transactions are serializable. The DBMS determines which transactions are serializable, and then interleaves the the execution of the transactions' operations. Transactions that are not serializable are executed on a first-come first-served basis.

The schedulers main job is to create a **serializable schedule** of a transactions' operations, in which the interleaved execution of the transactions yields the same results as if the transactions were executed in serial order (One after another). The scheduler also ensures that the CPU and storage systems are used efficiently. If there were no way to schedule the execution of transactions, all of them would be executed on a first-come first-served basis, which would result in an inefficient use of resources.

13.3 Concurrency Control with Locking Methods

Lock - guarantees exclusive use of a data item to a current transaction. A transaction acquires a lock prior to data access, and the lock is released when the transaction is completed. This series of locking actions assumes that concurrent transactions might attempt to manipulate the same data item at the same time.

Pessimistic locking - the use of locks based on the assumption that conflict between transactions is likely.

Lock manager - handles all lock information, responsible for assigning and policing the locks used by transactions.

13.3.1 Lock Granularity

Lock granularity - indicates the level of lock use. Locking can take place at the following levels -

- **Database Level** - the entire database is locked, preventing the use of any tables in the database by any other transactions. Good for batch processes, but is unsuitable for multiuser DBMSs - data access would be very slow if many transactions had to wait for the previous one to be completed.

- **Table Level** - the entire table is blocked, preventing access to any row by any other transactions. Two transactions can access the same database, as long as they access different tables.

  The table is blocked even though different transactions may require access to different parts of the same table, ie: not interfere with each other. Not suitable for multiuser DBMSs.

- **Page Level** - an entire page is locked. A **diskpage**, or **page**, is the equivalent of a *diskblock* (a directly addressable section of a disk). A page has a fixed size, and the entire page must be read from disk, updated, and written back to disk. A table can span several pages, and a page can contain several rows of one or more tables. Page-level locks are the most commonly used locking method for multiuser DBMSs.

- **Row Level** - is much less restrictive than the other lock types. The DBMS allows concurrent transactions to access different rows of the same table, even when the rows are located on the same page.

  Although the row-level locking approach improves the availability of data, its management requires high overhead because a lock exists for each row in the table.

- **Field Level** - allows concurrent transactions to access the same row as long as they require the use of different fields. Although filed-level locking yields the most flexible multiuser data access, it is rarely implemented because it requires an extremely high level of overhead, and because the row-level lock is much more useful in practice.

13.3.2 Lock Types

Binary Locks - A binary lock has only two states: locked(1), or unlocked(0). If a object such as a database table, page, or row is locked by a transaction, no other transaction can use that object. If an object is unlocked, any transaction can lock the object for its use.

Every database transaction requires that the affected object is locked, and a transaction must unlock the object after its termination. Every transaction requires a lock and unlock operation for each accessed data item, which is automatically managed by the DBMS.

Binary locks are considered too restrictive to yield optimal concurrency conditions. Eg: the DBMS will not allow two transactions to read the same database object even though neither transaction updates the database, and no concurrency problems can occur.

Shared / Exclusive Locks -

Exclusive lock exists when access is reserved specifically for the transaction that locked the object. Must be used when the potential for conflict exists. An exclusive lock is issued when a transaction wants to update (write) a data item, and no locks are held on that data item.

Mutual exclusive rule - Only one transaction at a time can own an exclusive lock on an object.

Shared lock exists when concurrent transactions are granted read access on the basis of a common lock. Produces no conflict as long as all the concurrent transactions are read-only. A shared lock is issued when a transaction wants to read data from the database, and no exclusive lock is held on that data item.

Using the shared locking concept, a lock can have three states: unlocked, shared (read), and exclusive (write).

Although the use of shared locks renders data access more efficient, a shared / exclusive lock schema increases the lock managers overhead -

- The type of lock must be known before a lock can be granted;
- Three lock operations exist: READ_LOCK to check the type of lock, WRITE_LOCK to issue the lock; and UNLOCK to release the lock;
- The schema has been enhanced to allow a lock to upgrade from shared to exclusive, and a lock downgrade from exclusive to shared.

Although locks prevent serious data inconsistencies, they can lead to two major problems -

- The resulting transaction schedule may not be serializable;
- The schedule may create deadlocks.

Deadlock - occurs when two, or more, transactions wait indefinitely for each other to unlock data.

13.3.3 Two-Phase Locking to Ensure Serializability

Two-phase locking - defines how transactions acquire and relinquish locks. Two-phase locking guarantees serializability, but it does not prevent deadlocks. The phases are -

- **Growing phase** - in which a transaction acquires all required locks without unlocking any data. Once all locks have been acquired, the transaction is in its locked state.
- **Shrinking phase** - in which a transaction releases all locks and cannot obtain a new lock.

Governed by the following rules -

- Two transactions cannot have conflicting locks;
- No unlock operation can precede a lock operation in the same transaction;
- No data are affected until all locks are obtained - that is, until a transaction is in its locked state.

Two-phase locking increases the transaction processing cost, and might cause additional undesirable effects, such as deadlocks.

13.3.4 Deadlocks

Deadlock - occurs when two transactions wait indefinitely for each other to unlock data.

Eg: T1 - access X and Y
T2 - access Y and X (known as a deadly embrace).

Deadlock is only possible only when one of the transactions wants to obtain an exclusive lock on a data item, no deadlock condition can exist among shared locks.

Three basic techniques to control deadlocks -

- **Deadlock prevention** - A transaction requesting a new lock is aborted when there is the possibility that a deadlock can occur. If the transaction is aborted, all changes made by the transaction are rolled back and all locks are released. The transaction is rescheduled for execution. Deadlock prevention works because it avoids the conditions that lead to deadlocking.
- **Deadlock detection** - The DBMS periodically tests the database for deadlocks. If a deadlock is found, the "victim" transaction is aborted (rolled back and restarted), and the other transaction continues.
- **Deadlock avoidance** - The transaction must obtain all of the locks it needs before it can be executed. This technique avoids the rolling back of conflicting transactions by requiring that locks be obtained in succession. However, this increases response times.

The choice of which deadlock control method to use depends on the database environment. If probability of deadlocks is low, deadlock detection is recommended. If the probability of deadlocks is high, deadlock prevention is recommended. If response time is not important, deadlock avoidance can be used. Most current DBMSs support deadlock detection.

13.4 Concurrency Control with Timestamping Methods

Timestamping approach - to scheduling concurrent transactions assigns a global, unique timestamp to each transaction. The timestamp value produces an explicit order in which transactions are submitted to the DBMS.

Timestamps must have two properties -

Uniqueness - ensures that no equal timestamp values can exist.

Monotonicity - ensures that timestamp values always increase.

All database operations within the same transaction must have the same timestamp. The DBMS executes conflicting operations in timestamp order, thereby ensuring serializability of the transactions. If two transactions conflict, one is stopped, rolled back, rescheduled, and assigned a new timestamp value.

Disadvantage of the timestamping approach is that each value stored in the database requires two additional timestamp fields: one for the last time the field was read, and one for the last update. It increases memory needs and processing overhead.

Timestamping demands a lot of system resources because many transactions might have to be stopped, rescheduled, and restarted.

13.4.1 Wait / Die and Wound / Wait Schemes

Timestamping methods are used to mange concurrent transaction execution. Two schemes are used to decide which transaction is rolled back, and which continues executing -

Wait / Die - the older transaction waits for the younger one to complete and release its locks

- If the transaction requesting the lock is the older of the two transactions, it will wait until the other transaction is completed and the locks are released;
- If the transaction requesting the lock is the younger of the two transactions, it will die (rollback), and is rescheduled using the same timestamp.

Wound / Wait - the older transaction rolls back the younger transaction and reschedules it.

- If the transaction requesting the lock is the older of the two transactions, it will preempt (wound) the younger transaction by rolling it back. The younger, preempted transaction is rescheduled using the same timestamp;
- If the transaction requesting the lock is the younger of the two transactions, it will wait until the other transaction is completed, and the locks are released.

In both schemes, one of the transactions waits for the other transaction to finish and release the locks. However, in many cases a transaction requests multiple locks. How long should a transaction have to wait ? Some transactions may have to wait indefinitely, causing a deadlock.

To prevent a deadlock, each lock request has an associated time-out value. If the lock is not granted before the time-out expires, the transaction is rolled back.

13.5 Concurrency Control with Optimistic Methods

Optimistic approach - is based on the assumption that the majority of database operations do not conflict. The optimistic approach requires neither locking nor timestamping techniques. Instead, a transaction moves through two or three phases -

- **Read phase** - the transaction reads the database, executes the needed computations, and makes the updates to a private copy of the database values. All update operations of the transaction are recorded in a temporary update file, which is not accessed by the remaining transactions;

- **Validation phase** - the transaction is validated to ensure that the changes made will not affect the integrity and consistency of the database. If the validation test is positive, the transaction goes to the write phase. If the validation test is negative, the transaction is restarted and the changes are discarded.

- **Write phase** - the changes are permanently applied to the database.

The optimistic approach is acceptable for most read or query database systems that require few update transactions.

13.6 Database Recovery Management

Database recovery - restores a database from a given state (usually inconsistent) to a previously consistent state. Recovery techniques are based on the atomic transaction property.

Atomic transaction property - all portions of the transaction must be treated as a single, logical unit of work in which all operations are applied and completed to produce a consistent database.

If a transaction operation cannot be completed for some reason, the transaction must be aborted, and any changes to the database must be rolled back (undone). Transaction recovery reverses all of the changes that the transaction made to the database before the transaction was aborted.

Recovery techniques also apply to the database and to the system after some type of critical error has occurred. Examples are -

- Hardware / software failures - one of the most common sources of database problems;

- Human caused incidents - Can be intentional, or unintentional;

- Natural disasters - Fires, earthquakes, power failures etc.

13.6.1 Transaction Recovery

Database transaction recovery uses data in the transaction log to recover a database from an inconsistent state to a consistent state. Four important concepts that affect the recovery process -

- **Write-ahead-log protocol** - ensures that transaction logs are always written before any database data are actually updated. This protocol ensures that, in the case of failure, the database can later be recovered to a consistent state using the data in the transaction log.

- **Redundant transaction logs** - (multiple copies of the transaction log) ensures that a physical failure will not impair the DBMSs ability to recover data.

- **Buffers** - are temporary storage areas in primary memory used to speed up disk operations. When a transaction updates data, it actually updates the copy of the data in the buffer, which is much faster than accessing the disk. Later, all buffers that contain updated data are written to disk during a single operation, saving significant processing time.

- **Checkpoints** - are operations in which the DBMS writes all of its updated buffers to disk. The DBMS does not execute any other requests while this is happening. A checkpoint operation is also registered in the transaction log. As a result of this operation, the physical database and the transaction log are in sync.

  This synchronization is required because update operations update the copy of the data in the buffers and not in the physical database. Checkpoints are automatically schedule by the DBMS. Checkpoints also play an important role in transaction recovery.

The database recovery process involves bringing the database to a consistent state after a failure. Generally, two techniques are used -

- **Deferred-write technique** (deferred update) - the transaction operations do not immediately update the physical database. Only the transaction log is updated. The database is physically updated only after the transaction reaches its commit point, using information from the transaction log.

  If the transaction aborts before it reaches its commit point, no changes (ROLLBACK) need to be made to the database, because it was never updated. The recovery process for all started and committed transactions follow these steps -

  - Identify the last checkpoint in the transaction log. This is the last time transaction data were physically saved to disk.
  - For a transaction that started and was committed before the last checkpoint, nothing needs to be done because the data are already saved.
  - For a transaction that performed a commit operation after the last checkpoint, the DBMS uses the transaction log records to redo the transaction and update the database, using the "after" values in the transaction log. The changes are made in ascending order, from oldest to newest.
  - For any transaction that has a ROLLBACK operation after the last checkpoint or that was left active (neither a COMMIT nor a ROLLBACK) before the failure occurred, nothing needs to be done because the database was never updated.

- **Write-through technique** (immediate update) - the database is immediately updated by transaction operations during the transactions' execution, even before the transaction reaches its commit point.

  If the transaction aborts before it reaches its commit point, a ROLLBACK or undo operation needs to be done to restore the database to a consistent state. In that case, the ROLLBACK operation will use the transaction log "before" values. The recovery process follows these steps -

  - Identify the last checkpoint in the transaction log. This is the last time transaction data were physically saved to disk.
  - For a transaction that started and was committed before the last checkpoint, nothing needs to be done because the data are already saved.
  - For a transaction that was committed after the last checkpoint, the DBMS redoes the transaction, using the "after" values of the transaction log. Changes are applied in ascending order, from oldest to newest.

- ◦ For any transaction that had a ROLLBACK operation after the last checkpoint, or that was left active (neither a COMMIT nor a ROLLBACK) before the failure occurred, the DBMS uses the transaction log records to ROLLBACK or undo the operations, using the "before" values in the transaction log. Changes are applied in reverse order, from newest to oldest.

Key Terms

- **atomicity** - See *atomic transaction property*.
- **atomic transaction property** - A property that requires all parts of a transaction to be treated as a single, logical unit of work in which all operations must be completed (committed) to produce a consistent database.
- **binary lock** - A lock that has only two states: locked (1), and unlocked (0). If a data item is locked by a transaction, no other transaction can use that data item.
- **buffers (buffer cache)** - A shared, reserved memory area that stores the most recently accessed data blocks in RAM. A buffer cache takes advantage of a computers' fast primary memory compared to the slower secondary memory, minimizing the number of input / output (I/O) operations between primary and secondary memory. (Also called data cache).
- **checkpoints** - In transaction management, an operation in which the database management system writes all of its updated buffers to disk.
- **concurrency control** - A DBMS feature that coordinates the simultaneous execution of transactions in a multiprocessing database system while preserving data integrity.
- **consistency** - A database condition in which all data integrity constraints are satisfied. To ensure consistency of a database, every transaction must begin with the database in a known consistent state. If not, the transaction will yield an inconsistent database that violates its integrity and business rules.
- **consistent database state** - A database state in which all data integrity constraints are satisfied.
- **database-level lock** - A type of lock that restricts database access to the owner of the lock and allows only one user at a time to access the database. This lock works for batch processes but is unsuitable for online multiuser DBMSs.
- **database recovery** - The process of restoring a database to a previous consistent state.
- **database request** - The equivalent of a single SQL statement in an application program or a transaction.
- **deadlock** - A condition in which two or more transactions wait indefinitely for the other to release the lock on a previously locked data item. Also called "*deadly embrace*".
- **deadly embrace** - See *deadlock*.
- **deferred update** - In transaction management, a condition in which transaction operations do not immediately update a physical database. Also called *deferred write technique*.
- **deferred-write technique** - See *deferred update*.
- **diskpage** - In permanent storage, the equivalent of a disk block, which can be described as a directly addressable section of a disk. A diskpage has a fixed size, such as 4K, 8K, or 16K.
- **durability** - The transaction property that indicates the permanence of a databases' consistent state. Transactions that have been completed will not be lost in a system failure if the database has proper

durability.

- **exclusive lock** - A lock that is reserved by a transaction. An exclusive lock is issued when a transaction requests permission to update a data item and no locks are held on that data item by any other transaction. An exclusive lock does not allow other transactions to access the database. See also *shared lock*.

- **field-level lock** - A lock that allows concurrent transactions to access the same row as long as they require the use of different fields (attributes) within that row. This type of lock yields the most flexible multiuser data access, but requires a high level of computer overhead.

- **immediate update** - A database update that is performed immediately during a transactions' execution, even before the transaction reaches its commit point.

- **inconsistent retrievals** - A concurrency control problem that arises when a transaction calculating summary (aggregate) functions over a set of data while other transactions are updating the data, yielding erroneous results.

- **isolation** - A property of a database transaction in which a data item used by one transaction is not available to other transactions until the first one ends.

- **lock** - A device that guarantees unique use of a data item in a particular transaction operation. A transaction requires a lock prior to data access; the lock is released after the operations' execution to enable other transactions to lock the data item for their own use.

- **lock granularity** - The level of lock use. Locking can take place at the following levels: database, table, page, row, and field (attribute).

- **lock manager** - A DBMS component that is responsible for assigning and releasing locks.

- **lost update** - A concurrency control problem in which data updates are lost during the concurrent execution of transactions.

- **monotonicity** - A quality that ensures that timestamp values always increase. The timestamping approach to scheduling concurrent transactions assigns a global, unique timestamp to each transaction. The timestamp value produces an explicit order in which transactions are submitted to the DBMS.

- **mutual exclusive rule** - A condition in which only one transaction at a time can own an exclusive lock on the same object.

- **optimistic approach** - In transaction management, a concurrency control technique, based on the assumption that most database operations do not conflict.

- **page** - See *diskpage*.

- **page-level lock** - In this type of lock, the database management system locks an entire diskpage, or section of a disk. A diskpage can contain data for one or more rows and from one or more tables.

- **pessimistic locking** - The use of locks based on the assumption that conflict between transactions is likely.

- **redundant transaction logs** - Multiple copies of the transaction log kept by database management systems to ensure that the physical failure of a disk will not impair the DBMSs ability to recover data.
- **row-level lock** - A less restrictive database lock in which the DBMS allows concurrent transactions to access different rows of the same table, even when the rows are on the same page.
- **scheduler** - The DBMS component that establishes the order in which concurrent transaction operations are executed. The scheduler interleaves the execution of database operations in a specific sequence to ensure serializabilty.
- **serializerable schedule** - In transaction management, a schedule of operations in which the interleaved execution of the transactions yields the same result as if they were executed in serial order.
- **serializability** - A property in which the selected order of transaction operations creates the same final database state that would have been produced if the transactions had been executed in a serial fashion.
- **shared lock** - A lock that is issued when a transaction requests permission to read data from a database and no exclusive locks are held on the data by another transaction. A shared lock allows other read-only transactions to access the database.
- **table-level lock** - A locking scheme that allows only one transaction at a time to access a table. A table-level lock locks an entire table, preventing access to any row by transaction T2 while transaction T1 is using the table.
- **timestamping** - In transaction management, a technique used in scheduling concurrent transactions that assigns a global unique timestamp to each transaction.
- **transaction** - A sequence of database requests that accesses the database. A transaction is a logical unit of work; that is, it must be entirely completed or aborted - no intermediate ending states are accepted. All transactions must have the properties of atomicity, consistency, isolation, and durability.
- **transaction log** - A feature used by the DBMS to keep track of all transaction operations that update the database. The information stored in this log is used by the DBMS for recovery purposes.
- **two-phase locking** - A set of rules that governs how transactions acquire and relinquish locks. Two-phase locking guarantees serializability, but it does not prevent deadlocks. The two-phase locking protocol is divided into two phases: 1) a growing phase occurs when the transaction acquires the locks it needs without unlocking any existing data locks. Once all locks have been acquired, the transaction is in its locked point. 2) A shrinking phase occurs when the transaction releases all locks and cannot obtain a new lock.
- **uncommitted data** - When you are trying to achieve concurrency control, uncommitted data cause problems with data integrity and consistency. These problems occur when two transactions are executed concurrently and the first transaction is rolled back after the second transaction has already accessed the uncommitted data, thus violating the isolation property of transactions.
- **uniqueness** - In concurrency control, a property of timestamping that ensures no equal timestamp values can exist.
- **wait / die** - A concurrency control scheme in which an older transaction must wait for the younger transaction to complete and release the locks before requesting the locks itself. Otherwise, the newer transaction dies and is rescheduled.

- **wound / wait** - A concurrency control scheme in which an older transaction can request the lock, preempt the younger transaction, and reschedule it. Otherwise, the newer transaction waits until the older transaction finishes.

- **write-ahead-log protocol** - In concurrency control, a process that ensures transaction logs are written to permanent storage before any database data are actually updated. Also called a write-ahead protocol.

- **write-through technique** - In concurrency control, a process that ensures a database is immediately updated by operations during the transactions' execution, even before the transaction reaches its commit point.

--ooOoo--

Chapter 14 - Managing Database and SQL Performance

Summary

- Database performance tuning refers to a set of activities and procedures designed to ensure that an end-user query is processed by the DBMS in the least amount of time.

- SQL performance tuning refers to activities on the client side that are designed to generate SQL code that returns the correct answer in the least amount of time, using the minimum amount of resources at the server end.

- DBMS performance tuning refers to activities on the server side that are oriented so the DBMS is properly configured to respond to clients requests in the fastest way possible while making optimum use of existing resources.

- The DBMS architecture is represented by the many processes and structures (in memory and permanent storage) used to manage a database.

- Database statistics refer to a number of measurements gathered by the DBMS that describe a snapshot of the database objects' characteristics. The DBMS gathers statistics about objects such as tables, indexes, and available resources, which include number of processors used, processor speed, and temporary space available. The DBMS uses the statistics to make critical decisions about improving query processing efficiency.

- DBMSs process queries in three phases. In the parsing phase, the DBMS parses the SQL query and chooses the most efficient access / execution plan. In the execution phase, the DBMS executes the SQL query using the chosen execution plan. In the fetching phase, the DBMS fetches the data and sends the result set back to the client.

- Indexes are crucial in the process that speeds up data access. Indexes facilitate searching, sorting, and using aggregate functions and join operators. The improvement in data access speed occurs because an index is an ordered set of values that contains the index key and pointers. Data sparsity refers to the number of different values a column could have. Indexes are recommended in high-sparsity columns used in search conditions.

- During query optimization, the DBMS must choose what indexes to use, how to perform join operations, which table to use first, and so on. Each DBMS has its own algorithms for determining the most efficient way to access the data. The two most common approaches are rule-based and cost-based optimization.

- A rule-based optimizer uses preset rules and points to determine the best approach to execute a query. The rules assign a "fixed cost" to each SQL operation: the costs are then added to yield the cost of the execution plan.

- A cost-based optimizer uses sophisticated algorithms based on statistics about the objects being accessed to determine the best approach to execute a query. In this case, the optimizer process adds up the processing cost, the I/O costs, and the resource costs (RAM and temporary space) to determine the total cost of a given execution plan.

- Hints are used to change the optimizer mode for the current SQL statement. Hints are special instructions for the optimizer that are embedded inside the SQL command text.

- SQL performance tuning deals with writing queries that make good use of the statistics. In particular, queries should make good use of indexes. Indexes are very useful when you want to select a small subset of rows from a large table based on a condition. When an index exists for the column used in the selection, the DBMS may choose to use it. The objective is to create indexes with selectivity. Index selectivity is a measure of the likelihood that an index will be used in query processing. It is also important to write conditional statements using some common principles.

- Query formulation deals with how to translate business questions into specific SQL code to generate the required results. To do this, you must carefully evaluate which columns, tables, and computations are required to generate the desired output.

- DBMS performance tuning includes tasks such as managing the DBMS processes in primary memory (allocating memory for caching purposes) and managing the structures in physical storage (allocating space for the data files).

Content

14.1 Database Performance-Tuning Concepts

Database performance tuning - refers to a set of activities and procedures designed to reduce the response time of the database system - that is, to ensure that an end-user query is processed by the DBMS in the minimum amount of time.

Good database performance starts with good database design. No amount of fine-tuning will make a poorly designed database perform as well as a well-designed database.

14.1.1 Performance Tuning: Client and Server

In general, database performance-tuning activities can be divided into those on the client side, and those on the server side -

SQL performance tuning - On the client side, the objective is to generate a SQL query that returns the correct answer in the least amount of time, using the minimum amount of resources at the server end.

DBMS performance tuning - On the server side, the DBMS environment must be properly configured to respond to the clients' requests in the fastest way possible, while making optimum use of existing resources.

14.1.2 DBMS Architecture

Data files - All data in a database are stored in data files.

Table space or **file group** - is a logical grouping of several data files that store data with similar characteristics. Eg: system table space, user data table space, index table space, temporary table space.

Data cache or **buffer cache** - is a shared, reserved memory area that stores the most recently accessed data blocks in RAM.

SQL cache or **procedure cache** - is a shared, reserved memory area that stores the most recently executed SQL statements or PL/SQL procedures, including triggers and functions.

Input / output (I/O) request - is a low-level data access operation that reads or writes data to and from computer devices, such as memory, hard disks, video, and printers.

Working with data in the data cache is much faster than working with data in the data files, because the DBMS does not have to wait for the hard disk to retrieve the data.

Most performance-tuning activities focus on minimizing the number of I/O operations because using I/O operations is many times slower than reading data from the disk cache.

Some typical DBMS processes (Names may vary from vendor to vendor) -

- *Listener* - Listens for clients requests and handles the processing of the SQL requests to other DBMS processes;
- *User* - The DBMS creates a user process to manage each client session. This process handles all requests submitted to the server.
- *Scheduler* - Organizes the concurrent execution of SQL requests.
- *Lock manager* - Manages all locks placed on database objects.

- *Optimizer* - Analyzes SQL queries and finds the most efficient way to access the data.

14.1.3 Database Query Optimization Modes

Most of the algorithms proposed for query optimization are based on two principles -

- The selection of the optimum execution order;
- The selection of sites to be accessed to minimize communication costs.

Within these two principles, a query optimization algorithm can be evaluated on the basis of its operation mode, or the timing of its optimization -

- **Operation modes** -
  - **Automatic query optimization** - means that the DBMS finds the most cost-effective access path without user intervention. More desirable from users point of view, but imposes increased overhead on the DBMS.
  - **Manual query optimization** - requires that the optimization be selected and scheduled by the end user or programmer.
- **Timing** -
  - **Static query optimization** - takes place at compilation time, the best optimization strategy is selected when the query is compiled by the DBMS.
  - **Dynamic query optimization** - takes place at execution time, database access strategy is defined when the program is executed. Although dynamic query optimization is efficient, it has high processing overhead. The best strategy is determined every time the query is executed.

Query optimization techniques can be classified according to the type of information that is used to optimize the query -

- **Statistically based query optimization algorithm** - uses statistical information about the database. These statistics are then used by the DBMS to determine the best access strategy.

  The statistical information is managed by the DBMS and is generated in one of two modes -

  - **Dynamic statistical generation mode** - the DBMS automatically evaluates and updates the statistics after each access.
  - **Manual statistical generation mode** - the statistics must be updated periodically through a user selected utility.

- **Rule-based query optimization algorithm** - is based on a set of user-defined rules to determine the best query access strategy. The rules are entered by the end user or database administrator.

14.1.4 Database Statistics

Database statistics - refers to a number of measurements about database objects, such as number of processors used, processor speed, and temporary space available. Provides a snapshot of database characteristics.

Database statistics are stored in the system catalog in specially designated tables. It is common to periodically regenerate the statistics for database objects. The more current the statistics are, the better the chances that the DBMS will properly select the fastest way to execute a given query.

14.2 Query Processing

The DBMS processes a query in three phases -

- **Parsing** - The DBMS parses the SQL query and chooses the most efficient access/execution plan.
- **Execution** - The DBMS executes the SQL query using the chosen execution plan.
- **Fetching** - The DBMS fetches the data and sends the result set back to the client.

The processing of SQL DDL statements (CRETE TABLE) is different from the processing required by DML statements (SELECT, INSERT, UPDATE, or DELETE). The difference is that a DDL statement actually updates the data dictionary tables or system catalog, while a DML statement mostly manipulates end-user data.

14.2.1 SQL Parsing Phase

Parsing - Breaking down the query into smaller units and transforming the original SQL query into a slightly different version of the original SQL code, but one that is fully equivalent and more efficient.

- *Fully equivalent* - means that the optimized query results are always the same as the original query.
- *More efficient* - means that the optimized query will almost always execute faster than the original query. The DBMS may use database statistics to determine the most efficient way to execute the query.

Query optimizer - analyzes the SQL query and finds the most efficient way to access the data. Parsing a SQL query requires several steps. The SQL query is -

- Validated for syntax compliance;
- Validated against the data dictionary to ensure that the table names and column names are correct;
- Validated against the data dictionary to ensure that the user has proper access rights;
- Analyzed and decomposed into more atomic components;
- Optimized through transformation into a fully equivalent but more efficient SQL query;
- Prepared for execution by determining the most efficient execution or access plan.

Access plan - is the result of parsing a SQL statement. It contains the series of steps a DBMS will use to execute the query and return the result in the most efficient way.

14.2.2 SQL Execution Phase

All I/O operations indicated in the access plan are executed. When the execution plan is run, the proper locks - if needed - are acquired for the data to be accessed, and the data are retrieved from the data files and placed in the DBMSs data cache.

14.2.3 SQL Fetching Phase

After the parsing and execution phases are completed, all rows that match the specified condition/s are retrieved, sorted, group, and aggregated (if required). During the fetching phase, the rows of the resulting query result set are returned to the client.

14.2.4 Query Processing Bottlenecks

Query processing bottleneck - is a delay introduced in the processing of an I/O operation that causes the overall system to slowdown.

Five components of a DBMS typically cause bottlenecks -

- **CPU** - The CPU processing power of the DBMS should match the systems' expected workload.
- **RAM** - If there is not enough RAM available, moving data among components that are competing for scarce RAM can create a bottleneck.
- **Hard disk** - Common causes of bottlenecks are hard disk speed and data transfer rates.
- **Network** - When many network nodes access the network at the same time, bottlenecks are likely.
- **Application code** - Two of the most common causes of bottlenecks are inferior application code, and poorly designed databases. Inferior code can be improved with code optimization techniques, as long as the underlying database design is sound. No amount of coding will make a poorly designed database perform better.

14.3 Indexes and Query Optimization

Indexes are crucial in speeding up data access because they facilitate searching, sorting, using aggregate functions, and even join operations. The improvement in data access speed occurs because an index is an ordered set of values that contains the index key and pointers. The pointers are the row IDs for the actual table rows.

An index scan is more efficient than a full table scan because the index data are preordered and the amount of data is usually much smaller. So, when performing searches, it is almost always better for the DBMS to use the index to access a table, than to scan all rows in a table sequentially.

If indexes are so important, why not index every column in every table ? The simple answer is that it is not practical to do so. Indexing every column in every table overtaxes the DBMS in terms of index-maintenance processing, especially if the table has many attributes and rows, or requires many inserts, updates, and deletes.

Measure that determines the need for an index -

Data sparsity - refers to the number of different values a column can have. If a column can only have, say, two values, the column has low sparsity. In contrast, if the column can have many different values, it has high sparsity.
Knowing the sparsity helps decide whether the use of an index is appropriate. Eg; if a search is performed in

a column with low sparsity, a high percentage of the rows have to read anyway, so index processing may be unnecessary work.

Indexes are implemented using one of the following data structures -

- **Hash index** - is based on an ordered list of hash values. This type of index is good for simple and fast lookup operations based on equality conditions. Eg: LName="Fred".
- **B-tree index** - is an ordered data structure organized as an upside-down tree. B-tree indexes are "self-balanced", which means that it takes approximately the same amount of time to access any given row in the index. Most common type of index used in databases. This type of index is used mainly in tables in which column values repeat a relatively small number of times.
- **Bitmap index** - uses a bit array (0s and 1s) to represent the existence of a value or condition. This type of index is used mostly in data warehouse applications in tables with a large number of rows in which a small number of column values repeat many times. Bitmap indexes use less space than B-tree indexes because they use bits instead of bytes.

14.4 Optimizer Choices

Query optimization is the central activity during the parsing phase in query processing. The query optimizer can operate in one of two modes -

- **Rule-based optimizer** - uses preset rules and points to determine the best approach to execute a query. The rules assign a "fixed cost" to each SQL operation, which are then added to yield the cost of the execution plan.
- **Cost-based optimizer** - uses sophisticated algorithms based on statistics about the objects being accessed to determine the best approach to execute a query. The optimizer process adds up the processing cost, the I/O cost, and resources cost (RAM etc) to determine the total cost of a given execution plan.

The optimizers' objective is to find alternative ways to execute a query, to evaluate the "cost" of each alternative, and then to choose the one with the lowest cost.

14.4.1 Using Hints to Affect Optimizer Choices

In some instances the optimizer might not choose the best optimization plan. Sometimes the end user would like to change the optimizer mode for the current SQL statement.

Optimizer hints - are special instructions for the optimizer that are embedded inside the SQL command text.
Eg: SELECT /*+ ALL_ROWS*/*, or SELECT /*+FIRST_ROWS*/*.

14.5 SQL Performance Tuning

A poorly written SQL query can bring a database system to its knees form a performance point of view. The majority of database performance problems are related to poorly written SQL code.

14.5.1 Index Selectivity

Indexes are the most important technique used in SQL performance optimization. The key is to know when an index is used. Indexes are likely to be used -

- When an indexed column appears by itself in the search criteria of a WHERE or HAVING clause.
- When an indexed column appears by itself in a GROUP BY or ORDER BY clause.
- When a MAX or MIN function is applied to an indexed column.
- When the data sparsity on the indexed column is high.

Index selectivity - is a measure of the likelihood that an index will be used in query processing. The objective is to create indexes with high selectivity.

General guidelines for creating and using indexes -

- Create indexes for each single attribute used in a WHERE, HAVING, ORDER BY, or GROUP BY clause.
- Do not use indexes in small tables or tables with low sparsity.
- Declare primary and foreign keys so the optimizer can use the indexes in join operations.
- Declare indexes in join columns other than PK or FK.

14.5.2 Conditional Expressions

A conditional expression is normally placed within a WHERE or HAVING clause of a SQL statement. It restricts the output of a query to only the rows that match the conditional criteria.

Common practice to write efficient conditional expressions in SQL code -

- Use simple columns or literals as operands in conditional expressions - avoid the use of conditional expressions with functions wherever possible.
- Numeric field comparisons are faster than character, date, and NULL comparisons.
- Equality comparisons are faster than inequality comparisons (ie: = as opposed to >=).
- Wherever possible, transform conditional expressions to use literals.
- When using multiple conditional expressions, write the equality conditions first.
- If you use multiple AND conditions, write the condition most likely to be false first.
- When evaluating multiple OR conditions, put the condition most likely to be true first.
- Whenever possible, try to avoid the use of the NOT logical operator.

14.6 Query Formulation

SELECT is most commonly used for queries in applications.

14.7 DBMS Performance Tuning

DBMS performance tuning at the server end focuses on setting the parameters used for -

- **Data cache** - The data cache must be set large enough to permit as many data requests as possible to be serviced from the cache.
- **SQL cache** - stores the most recently executed SQL statements, after they have been parsed by the optimizer. The same query may be executed many times, without having to go through the parsing phase again.
- **Sort cache** - is used as a temporary storage area for ORDER BY or GROUP BY operations, as well as for index-creation functions.
- **Optimizer mode** - Most DBMSs operate in either cost-based or rule-based optimization mode. Others automatically determine the optimization mode based on whether database statistics are available.

Managing the physical storage details of the data files also plays an important role in DBMS performance tuning. General recommendations for physical storage of databases -

- Use **RAID** (Redundant array of independent disks) to provide both performance improvement and fault tolerance. The most common RAID configurations -

RAID Level	Description
0	The data blocks are spread over separate drives. Also known as striped array. Provides increased performance, but no fault tolerance.
1	The same data blocks are written (duplicated) to separate drives. Also referred to as mirroring or duplexing. Provides increased read performance, and fault tolerance via data redundancy.
3	The data are striped across separate drives, and parity data are computed and stored in a dedicated drive. provides good read performance, and fault tolerance via parity data.
5	The data and parity data are striped across separate drives. Provides good read performance and fault tolerance via parity data.

- Minimize disk contention. Use multiple, independent storage volumes with independent spindles to minimize hard disk cycles.
- Put high usage tables in their own table spaces, so the database minimizes conflict with other tables.
- Assign separate data files in separate storage volumes for the indexes, system, and high-usage tables. This ensures that index operations will not conflict with end-user data or data dictionary operations.
- Take advantage of the various table storage organizations available in the database.
- Partition tables based on usage. Put the table partitions closest to where they are used the most.
- Use denormalized tables where appropriate.

- Store computed and aggregated attributes in tables. (ie: Use derived attributes). This minimizes computations in queries and join operations.

14.8 Query Optimization Example

Creating an index -

```
SQL>   CREATE INDEX QOV_NDX2 ON QOVENDOR(V_NAME);
```

Key Terms

- **access plan** - A set of instructions generated at application compilation time that is created and managed by a DBMS. The access plan predetermines how an applications' query will access the database at runtime.
- **automatic query optimization** - A method by which a DBMS finds the most efficient access path for the execution of a query.
- **bitmap index** - An index that uses a bit array (0s and 1s) to represent the existence of a value or condition.
- **b-tree index** - An ordered data structure organized as an upside-down tree.
- **buffer cache** - A shared, reserved memory area that stores the most recently accessed data blocks in RAM. A buffer cache takes advantage of a computers' fast primary memory compared to the slower secondary memory, minimizing the number of input / output (I/O) operations between primary and secondary memory. Also called *data cache*.
- **clustered index table** - See *index organized table*.
- **cost-based optimizer** - A query optimizer technique that uses an algorithm based on statistics about the objects being accessed, including number of rows, indexes available, index sparsity, and so on.
- **database performance tuning** - A set of activities and procedures designed to reduce the response time of a database system - that is, to ensure that an end-user query is processed by the DBMS in the minimum amount of time.
- **database statistics** - In query optimization, measurements about database objects, such as the number of rows in a table, number of disk blocks used, maximum and average row length, number of columns in each row, and number of distinct values in each column. Such statistics provide a snapshot of database characteristics.
- **data cache** - A shared, reserved memory area that stores the most recently accessed data blocks in RAM. Also called *buffer cache*.
- **data files** - A named physical storage space that stores a databases' data. It can reside in a different directory on a hard disk, or on one or more hard disks. All data in a database are stored in data files. A typical enterprise database is normally composed of several data files. A data file can contain rows from one or more tables.
- **data sparsity** - A column distribution of values, or the number of different values a column can have.
- **DBMS performance tuning** - Activities to ensure that clients' requests are addressed as quickly as possible while making optimum use of existing resources.
- **dynamic query optimization** - The process of determining the SQL access strategy at run time, using the most up-to-date information about the database. Contrast with *static query optimization*.
- **dynamic statistical generation mode** - In a DBMS, the capability to automatically evaluate and update the database access statistics after each data access.
- **extends** - In a DBMS environment, refers to the ability of data files to expand in size automatically using predefined increments.
- **file group** - See *table space*.

- **function-based index** - A type of index based on a specific SQL function or expression.
- **hash index** - An index based on an ordered list of hash values.
- **index-organized table** - In a DBMS, a type of table storage organization that stores end-user data and index data in consecutive locations in permanent storage. Also known as *cluster-indexed table*.
- **index selectivity** - A measure of how likely an index is to be used in query processing.
- **input / output (I/O) request** - A low-level operation that reads or writes data to and from computer devices such as memory, hard disks, video, and printers.
- **manual query optimization** - An operation mode that requires the end user or programmer to define the access path for the execution of a query.
- **manual statistical generation mode** - A mode of generating statistical data access information for query optimization. In this mode, the DBA must periodically run a routine to generate the data access statistics - for example, running the RUNSTAT command in an IBM DB2 database.
- **optimizer hints** - Special instructions for the query optimizer that are embedded inside the SQL command text.
- **procedure cache** - A shared, reserved memory area that stores the most recently executed SQL statements or PL/SQL procedures, including triggers and functions. Also called *SQL cache*.
- **query optimizer** - A DBMS process that analyses SQL queries and finds the most efficient way to access the data. The query optimizer generates the access or execution plan for the query.
- **query processing bottleneck** - In query optimization, a delay introduced in the processing of an I/O operation that causes the overall system to slow down.
- **RAID** - An acronym for Redundant Array of Independent Disks. RAID systems use multiple disks to create virtual disks (storage volumes) from several individual disks. RAID systems provide performance improvement, fault tolerance and a balance between the two.
- **rule-based optimizer** - A query optimization mode based on the rule-based query optimization algorithm.
- **rule-base query optimization algorithm** - A query optimization technique that uses preset rules and points to determine the best approach to executing a query.
- **static query optimization** - A query optimization mode in which the access path to a database is predetermined at compilation time. Contrast with *dynamic query optimization*.
- **statistically based query optimization algorithm** - A query optimization technique that uses statistical information about a database. The DBMS then uses these statistics to determine the best access strategy.
- **SQL cache** - A shared, reserved memory area that stores the most recently executed SQL statements or PL/SQL procedures, including triggers and functions. Also called *procedure cache*.
- **SQL performance tuning** - Activities to help generate a SQL query that returns the correct answer in the least amount of time, using the minimum amount of resources at the server end.
- **table space** - In a DBMS, a logical storage space used to group related data. Also known as a *file group.*

--ooOoo--

Chapter 15 - Databases for Decision Support

Summary

- Business intelligence (BI) is a term for a comprehensive, cohesive, and integrated set of applications used to capture, collect, integrate, store, and analyze data with the purpose of generating and presenting information to support business decision making.

- Decision support systems (DSS) refer to an arrangement of computerized tools used to assist managerial decision making within a business. DSSs were the original precursor of current-generation BI systems.

- Operational data are not well suited for decision support. From the end users' point of view, decision support data differ from operational data in three main areas: time span, granularity, and dimensionality.

- The data warehouse is an integrated, subject-oriented, time-variant, non-volatile collection of data that provides support for decision making. The data warehouse is usually a read-only database optimized for data analysis and query processing. A data mart is a small, single-subject data warehouse subset that provides decision support to a small group of people.

- The star schema is a data-modeling technique used to map multidimensional decision support data into a relational database for advanced data analysis. The basic star schema has four components: facts, dimensions, attributes, and attribute hierarchies. Facts are numeric measurements or values that represent a specific business aspect or activity. Dimensions are general qualifying categories that provide additional perspectives to facts. Conceptually, the multidimensional data model is best represented by a three-dimensional cube. Attributes can be ordered in well-defined hierarchies, which provide a top-down organization that is used for two main purposes: to permit aggregation and provide drill-down and roll-up data analysis.

- Data analytics is a subset of BI functionality that provides advanced data analysis tools to extract knowledge from business data. Data analytics can be divided into explanatory and predictive analytics. Explanatory analytics focuses on discovering and explaining data characteristics and relationships. Predictive analytics focuses on creating models to predict future outcomes or events based on the existing data.

- Data mining automates the the analysis of operational data to find previously unknown data characteristics, relationships, dependencies, and trends. The data-mining process has four phases: data preparation, data analysis and classification, knowledge acquisition, and prognosis.

- Predictive analytics uses the information generated in the data-mining phase to create advanced predictive models with high degrees of accuracy.

- Online analytical processing (OLAP) refers to an advanced data analysis environment that supports decision making, business modelling, and operations research.
- SQL has been enhanced with extensions that support OLAP-type processing and data generation.

Content

15.1 The need for Data Analysis

How can companies survive on lower margins and still make a profit ? The key is in having the right data at the right time to support the decision making process.

Changes in the business world, such as globalization, expanding markets, mergers and acquisitions, increased regulation, and new technologies, called for new ways of integrating and managing decision support across levels, sectors, and geographic locations. This more comprehensive and integrated decision support framework within organizations became known as business intelligence.

15.2 Business Intelligence

Business intelligence (BI) - is a term that describes a comprehensive, cohesive, and integrated set of tools and processes used to capture, collect, integrate, store, and analyze data with the purpose of generating and presenting information to support business decision making. This intelligence is based on learning and understanding the facts about the business environment. BI is a framework that allows a business to transform data into information, information into knowledge, and knowledge into wisdom.

BI is not a product by itself, but a framework of concepts, tools, and technologies that help a business better understand its core capabilities, provide snapshots of the company situation, and identify key opportunities to create competitive advantage. BI provides a framework for -

- Collection and storing operational data (Not really part of BI system, operational input to BI);
- Aggregating the operational data into decision support data;
- Presenting such information to the end user to support business decisions;
- Making business decisions, which in turn generate more data that are collected, stored and so on;
- Monitoring results to evaluate outcomes of the business decisions, which again provides more data to be collected, stored, and so on;
- Predicting future behaviour and outcomes with a high degree of accuracy.

15.2.1 Business Intelligence Architecture

Six components provide the functionality for BI systems -

Component	Description
ETL tools	**Extraction, transformation and loading** (ETL) - tools collect, filter, integrate and aggregate internal and external data to be saved into a data store optimized for decision support.
Data store	The data store is optimized for decision support, and is generally represented by a *data warehouse* or a *data mart*.
Query and reporting	Performs data selection and retrieval. Used by the data analyst to create queries that access the database and create the required reports.
Data visualization	Presents data to the end user in a variety of meaningful and innovative ways.
Data monitoring and alerting	Allows real-time monitoring of business activities.

Data analytics	Performs data analysis and data-mining tasks using the data in the data store. Data analysis can be either explanatory or predictive. Explanatory analysis uses the existing data in the data store to discover relationships and their types, and predictive analysis creates statistical models of the data that allow predictions of future values and events.

BI uses an arrangement of best management practices to manage data as a corporate asset.

Master data management (MDM) - is a collection of concepts, techniques, and processes for the proper identification, definition, and management of data elements within an organization. MDMs main goal is to provide a comprehensive and consistent definition of all data within an organization. MDM ensures that all company resources (people, procedures, and IT systems) that work with data have uniform and consistent views of the companys' data.

Governance - is a method or process of government. BI provides a method for controlling and monitoring business health and for consistent decision making. Having such governance creates accountability for business decisions.

Key performance indicators (KPIs) - are quantifiable numeric or scale-based measurements that assess the companys' effectiveness or success in reaching its strategic and operational goals. Some examples of KPIs -

- **General** - Year-to-year measurements of profit by line-of-business, same-store-sales, product turnovers, product recalls, sales by promotion, and sales by employee.
- **Finance** - Earnings per share, profit margin, revenue per employee, percentage of sales to account receivables, and assets to sales.
- **Human resources** - Applicants to job openings, employee turnover, and employee longevity.
- **Education** - Graduation rates, number of incoming freshman, student retention rates, publication rates, and teaching evaluation scores.

The heart of the BI system is its advanced information and decision support capabilities. A modern BI system provides three distinctive reporting styles -

- **Advanced reporting** - presents insightful information about the organization in a variety of presentation formats.
- **Monitoring and alerting** - After a decision has been made, the BI system offers ways to monitor the decisions outcome.
- **Advanced data analytics** - provides tools to help the end user discover relationships, patterns, and trends hidden within the organizations data. Two types of data analysis: explanatory and predictive.

15.2.2 Business Intelligence Benefits

Besides improved decision making, BI provides other benefits -

- *Integrating architecture* - Supports diverse hardware such as mainframes, servers, desktops, and mobile devices.

- *Common user interface for data reporting and analysis* - BI front ends can provide up-to-the-minute consolidated information using a common interface for all company users.
- *Common data repository fosters single version of company data* - BI provides a framework to integrate such data under a common environment and present a single version of the data.
- *Improved organizational performance* - results in added efficiency, reduced waste, increased sales, reduced employee and customer turnover, and an increased bottom line.

15.2.3 Business Intelligence Evolution

Decision support system (DSS) - is an arrangement of computerized tools used to assist managerial decision making. First-generation DSS was the precursor of the modern BI environment. A DSS typically has a much narrower focus and reach than a BI solution.

Evolution of information dissemination used in BI -

- Late 1970s - centralized predefined reports running on mainframes or minicomputers.
- Desktop computers 1980s - Spreadsheet, information downloaded from centralized data stores and manipulated in desktop spreadsheets.
- 1990s - All data integrated into IT umbrella, first-generation DSS. Still used spreadsheet like features.
- Mid 1990s - Once DSSs were established, evolution of BI flourished, with the introduction of the data warehouse, and OLAP (online analytical processing).
- Rapid changes in IT technology and the Internet revolution led to advanced BI systems, such as Web-based dashboards, and mobile BI that runs on mobile smart devices.

15.2.4 Business Intelligence Technology Trends

Some technological trends are -

- *Data storage improvements* - New data storage technologies offer increased performance and larger capacity that make data storage faster and more affordable.
- *Business intelligence appliances* - Vendors offer plug-and-play appliances optimized for data warehouse and BI applications.
- *Business intelligence as a service* - Cloud-based services allow any corporation to rapidly develop a data warehouse store without the need for hardware, software, or extra personnel.
- *Big Data analytics* - The Big Data phenomenon is creating a new market for data analytics. Organizations are turning to social media as the new source for information and knowledge to gain competitive advantages.
- *Personal analytics* - OLAP brought data analytics to the desktop of every user in an organization. Mobile BI is extending business decision making outside the walls of the organization. BI can be deployed to mobile users who are closer to customers.

15.3 Decision Support Data

Although BI is used at strategic and tactical managerial levels, its effectiveness depends on the quality of data gathered at the operational level.

15.3.1 Operational Data vs Decision Support Data

Operational data and decision support data serve different purposes, so their formats and structures differ. Whereas operational data are useful for capturing daily business transactions, decision support data give tactical and strategic business meaning to the operational data.

From the data analysts' point of view, decision support data differ from operational data in three main areas -

- **Time span** - Operational data cover a short time frame. Decision support data tend to cover a longer time frame. Managers focus on sales generated during the last month, or the last year rather than a specific invoice, for example.
- **Granularity** (level of aggregation) - Decision support data must be presented at different levels of aggregation, from highly summarized to nearly atomic.
    - **Drill down** - decompose the data into more atomic components, that is fine-grained data at lower levels of aggregation.
    - **Roll up** - aggregate the data to a higher level.
- **Dimensionality** - Operational data focus on representing individual transactions, rather than the effects of the transactions over time. In contrast, data analysts include many data dimensions and are interested in how the data relate over those dimensions.

From the database designers point of view, the differences between operational and decision support data are -

Characteristic	Operational Data	Decision Support Data
Data currency	Current operations Real-time data	Historic data Snapshot of company data Time component (week / month / year)
Granularity	Atomic-detailed data	Summarized data
Summarization level	Low; some aggregate yields	High; many aggregate yields
Data model	Highly normalized Mostly relational DBMSs	Non-normalized Complex structures Some relational, but mostly multidimensional DBMSs
Transaction type	Mostly updates	Mostly requests
Transaction volumes	High update volumes	Periodic loads and summary calculations
Transaction speed	Updates are critical	Retrievals are critical
Query activity	Low to medium	High
Query scope	Narrow range	Broad range
Query complexity	Simple to medium	Very complex
Data volumes	Hundreds of gigabytes	Terabytes to petabytes

15.3.2 Decision Support Database Requirements

A decision support database is a specialized DBMS tailored to provide fast answers to complex queries. There are three main requirements for a decision support database -

- **Database Schema** - The decision support database schema must support complex (non-normalized) data representations, and must be optimized for query (read-only) retrievals. The decision support database must contain data that are aggregated and summarized.

- **Data Extraction and Filtering** - The decision support database is created largely by extracting data from the operational database, and by importing additional data from external sources. So, the DBMS must support advanced data extraction and data-filtering tools. Data-filtering capabilities must include the ability to check for inconsistent data or, data validation rules.

- **Database Size** - Decision support databases tend to be very large. Must be capable of supporting *very large databases* (VLDBs).

15.4 The Data Warehouse

Data warehouse - an integrated, subject-oriented, time-variant, nonvolatile collection of data that provides support for decision making. More detailed look at the components -

- *Integrated* - The data warehouse is a centralized, consolidated database that integrates data derived from the entire organization and from multiple sources with diverse formats. Data integration implies that all business entities, data elements, data characteristics, and business metrics are described in the same way throughout the enterprise.

- *Subject-oriented* - Data warehouse data are arranged and optimized to provide answers to questions from diverse functional areas within a company. They are organized and summarized by topic, such as sales, marketing, finance, distribution, and transportation.

- *Time-variant* - In contrast to operational data, which focus on current transactions, warehouse data represent the flow of data through time. The data warehouse can also contain projected data generated through statistical and other models.

- *Nonvolatile* - Once data enter the data warehouse, they are never removed. Because data are never deleted and new data are continually added, the data warehouse is always growing.

In summary, the data warehouse is a read-only database optimized for data analysis and query processing.

15.4.1 Data Marts

Data mart - is a small, single-subject data warehouse subset that provides decision support to a small group of people. In addition, a data mart could be created from data extracted from a larger data warehouse for the specific purpose of supporting faster data access to a target group or function. Data marts and data warehouses can coexist.

The only difference between a data mart and a data warehouse is the size and scope of the problem being solved. The problem definitions and data requirements are essentially the same for both.

15.4.2 Twelve Rules that Define a Data Warehouse

Rule No	Description
1	The data warehouse and operational environments are separated.
2	The data warehouse data are integrated.
3	The data warehouse contains historical data over a long period of time.
4	The data warehouse data are snapshot data captured at a given point in time.
5	The data warehouse data are subject oriented.
6	The data warehouse data are mainly read-only with periodic batch updates from operational data. No online updates are allowed.
7	The data warehouse development life cycle differs from classical systems development. Data warehouse development is data-driven; the classical approach is process driven.
8	The data warehouse contains data with several levels of detail: current detail data, old detail data, lightly summarized data, and highly summarized data.
9	The data warehouse environment is characterized by read-only transactions to very large data sets. The operational environment is characterized by numerous update transactions to a few data entities at a time.
10	The data warehouse environment has a system that traces data sources, transformations, and storage.
11	The data warehouses' metadata are a critical component of this environment. The metadata identify and define all data elements. The metadata provide for the source, transformation, integration, storage, usage, relationships, and history of each data element.
12	The data warehouse contains a chargeback mechanism for resource usage that enforces optimal use of the data by end users.

15.5 Star Schemas

Most data warehouse implementations are based on the relational model. Relational data warehouses use the star schema design technique to handle multidimensional data.

Star schema - is a data-modeling technique used to map multidimensional decision support data into a relational database. The star schema creates the near equivalent of a multidimensional database schema from the existing relational database, while preserving the relational structures on which the operational database is built. The star schema has four components -

15.5.1 Facts

Facts - are numeric measurements (values) that represent a specific business aspect or activity. Facts commonly used are units, costs, prices, and revenues.

Fact table - Facts are normally stored in a fact table that is the centre of the star schema. It contains facts that are linked through their *dimensions*.

Metrics - facts computed or derived at run time, so called to differentiate from stored facts.

15.5.2 Dimensions

Dimensions - are qualifying characteristics that provide additional perspectives to a given fact. Eg: Sales may have location, product, and time dimensions. Such dimensions are normally stored in a **dimension table**.

15.5.3 Attributes

Each dimension table contains attributes, which are often used to search, filter, or classify facts. Dimensions provide descriptive characteristics about the facts through their attributes.

Eg: Sales - possible attributes for each dimension:

Dimension Name	Possible Attributes
Location	Region, state, city, store.
Product	Product type, product ID, brand, colour, size.
Time	Year, quarter, month, week, day, time of day.

15.5.4 Attribute Hierarchies

Attribute hierarchy - provides a top-down data organization that is used for two main purposes: aggregation and drill-down / roll-up analysis. The attribute hierarchy provides the capability to perform drill-down and roll-up searches in a data warehouse.

Eg: Region -> State -> City -> Store.

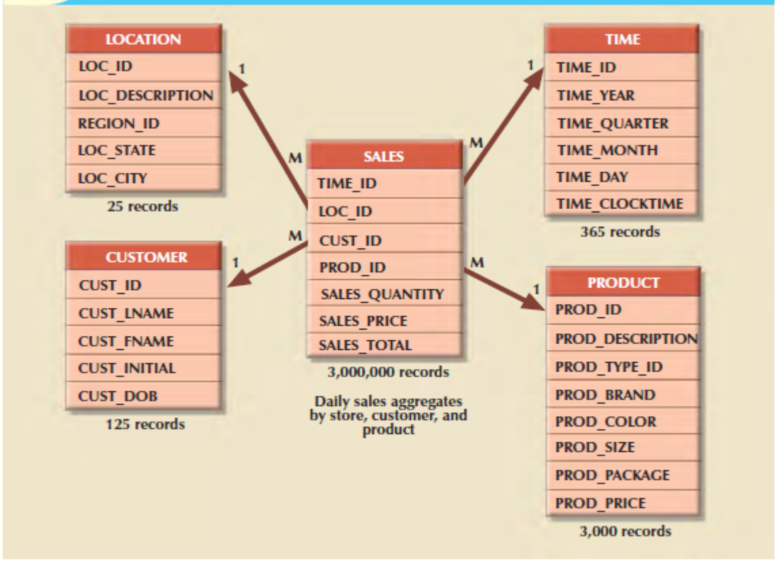
The attribute hierarchy information is stored in the DBMSs data dictionary and is used by the BI tool to access the data warehouse properly.

15.5.5 Star Schema Representation

Facts and Dimensions are normally represented by physical tables in the data warehouse database. The fact table is related to each dimension in a many-to-one relationship. Many Fact rows are related to each Dimension row.

Fact and Dimension tables are related by foreign keys and are subject to the familiar primary key and foreign key constraints. The primary key on the "1" side, the Dimension table, is stored as part of the primary key on the "many" side, the Fact table. Because the Fact table is related to many Dimension tables, the primary key of the Fact table is a composite primary key.

FIGURE 13.17 Star schema for SALES



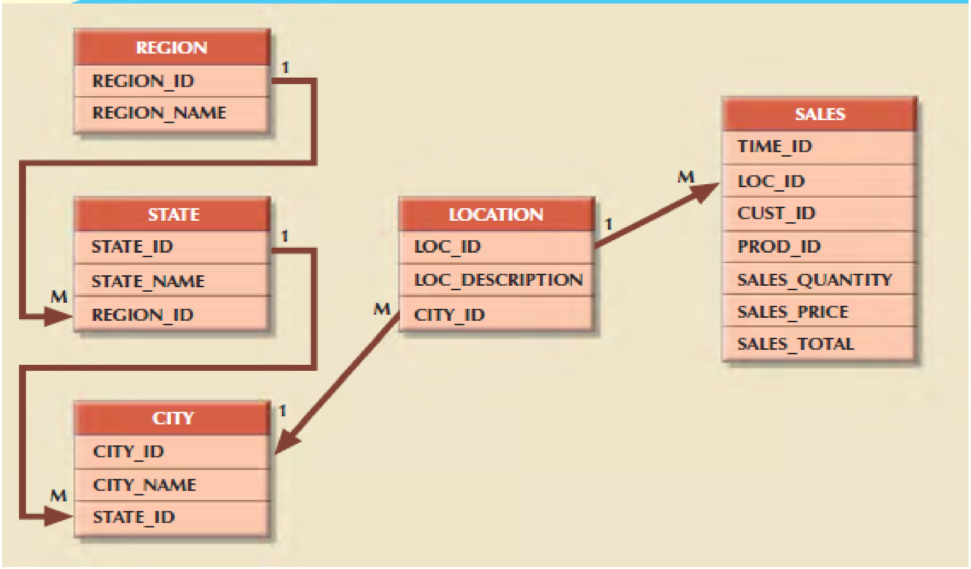
15.5.6 Performance-Improving Techniques for the Star Schema

Creating a database that provides fast and accurate answers to data analysis queries is the prime objective of data warehouse design. The following four techniques are often used to optimize data warehouse design:

Normalizing Dimension Tables

Dimension tables are normalized to achieve semantic simplicity and facilitate end-user navigation through the dimensions. Eg: if the location dimension table contains transitive dependencies among Region, State, and City you can revise those relationships to the 3NF.

FIGURE 13.19 Normalized dimension tables



This is known as a *snowflake schema* - type of star schema in which the Dimension tables can have their own Dimension tables. Usually the result of normalizing Dimension tables.

Maintaining Multiple Fact Tables That Represent Different Aggregation Levels

Query operations can be speeded up by creating and maintaining multiple Fact tables related to each level of aggregation (Region, State, City) in the location dimension. These aggregate tables are precomputed at the data-loading phase rather than at run time. The purpose of this technique is to save processor cycles at run time, thereby speeding up data analysis.

Denormalizing Fact Tables

Denormalizing Fact tables improves data access performance and saves data storage space. (Storage space is becoming less of an issue - data storage costs are decreasing). Denormalizing improves performance by using a single record to store data that normally take many records.

Partitioning and Replicating Tables

Table partitioning and replication are particularly important when a BI system is implemented in dispersed geographic areas.

Partitioning - splits a table into subsets of rows or columns and places the subsets close to the client computer to improve data access time.

Replication - makes a copy of a table or partition and places it in a different location, also to improve access time.

No matter which performance-enhancement scheme is used, time is the most common Dimension used in business data analysis. it is common to have one Fact table for each level of aggregation defined within the time Dimension. These Fact tables must have an implicit or explicit periodicity defined.

Periodicity - provides information about the time span of the data stored in the table. Usually expressed as current year only, previous years, or all years.

15.6 Data Analytics

Data analytics - is a subset of BI functionality that encompasses a wide range of mathematical, statistical, and modeling techniques with the purpose of extracting knowledge from data. Data analytics represents what business managers really want from BI: the ability to extract actionable business insight from current events and foresee future problems or opportunities.

Data analytics discovers characteristics, relationships, dependencies, or trends in the organizatons' data, and then explains the discoveries, and predicts future events based on the discoveries. Data analytics tools can be grouped into two separate, but related and overlapping, areas -

- **Explanatory analytics** - focuses on discovering and explaining data characteristics and relationships based on existing data.

- **Predictive analytics** - focuses on predicting future data outcomes with a high degree of accuracy.

Explanatory analytics explains the past and present, while predictive analytics forecasts the future.

15.6.1 Data Mining

Data mining - refers to analyzing massive amounts of data to uncover hidden trends, patterns, and relationships; to form computer models to simulate and explain the findings; and then to use such models to support business decision making. It focuses on the discovery and explanation stages of knowledge acquisition.

Data mining consists of four general phases -

- Data preparation
  - Identify data set
  - Clean data set
  - Integrate data set

- Data analysis and classification
  - Classification analysis
  - Clustering and sequence analysis
  - Link analysis
  - Trend and deviation analysis

- Knowledge acquisition
  - Select and apply algorithms
  - Neural networks
  - Inductive logic
  - Decision trees
  - Clustering
  - Regression tree

  - Nearest neighbour
  - Visualization

- Prognosis
  - Modeling
  - Forecasting
  - Prediction

Data mining can be run in two modes -

- Guided - The end user decides what techniques to apply to the data.

- Automated - The data mining tool applies multiple techniques to find significant relationships.

15.6.2 Predictive Analytics

Predictive analytics - refers to the use of advanced mathematical, statistical, and modeling tools to predict future business outcomes with high degrees of accuracy.

Predictive analytics can be thought of as the next logical step after data mining - once you understand your data, you can use your data to predict behaviours. Many vendors are replacing the term data mining with the term predictive analytics.

15.7 Online Analytical Processing

Online analytical processing (OLAP) - is a BI style whose systems share three main characteristics -

- Multidimensional data analysis techniques;
- Advanced database support;
- Easy-to-use end-user interfaces.

15.7.1 Multidimensional Data Analysis Techniques

The most distinctive characteristic of OLAP tools is their capacity for multidimensional analysis, in which data are processed and viewed as part of a multidimensional structure.

Eg: Data analyst would be interested in how Sales relate to other business variables, such as customers and time. Customers and time are viewed as different dimensions of Sales. (Pivot table is an example).

Multidimensional data analysis techniques are augmented by the following functions -

- *Advanced data presentation functions* - include 3D graphics, pivot tables, crosstabs, data rotation, and three-dimensional cubes.

- *Advanced data aggregation, consolidation, and classification functions* - allow the data analyst to create multiple data aggregation levels, slice and dice data, and drill-down and rool-up data across different dimensions and aggregation levels.

- *Advanced computational functions* - include business-oriented variables such as market share, period comparisons, sales margins, product margins, and percentage changes; financial and accounting ratios, including profitability, overhead, and cost allocations; and statistical and forecasting functions.

- *Advanced data modeling functions* - provide support for what-if scenarios, variable assessment, contributions to outcome, linear programming, and predictive modeling tools.

15.7.2 Advanced Database Support

OLAP tools must have the following advanced data access features -

- Access to many different kinds of DBMSs, flat files, and internal and external data sources;
- Access to aggregated data warehouse data, as well as to the detail data found in operational databases;
- Advanced data navigation features such as drill-down and roll-up;
- Rapid and consistent query response times;
- The ability to map end-user requests, expressed in either business or model terms, to the appropriate data source and then to the proper data access language (usually SQL);
- Support for very large databases.

15.7.3 Easy-to-Use End-User Interfaces

The end-user analytical interface is one of the most critical OLAP components. Advanced OLAP features become more useful when access to them is kept simple.

Most OLAP vendors have closely integrated their systems with spreadsheets such as MS Excel. Users gain access to advanced data analysis features by using familiar programs and interfaces, and additional training and development costs are minimized.

15.7.4 OLAP Architecture

An OLAP system has three main architectural components -

- Graphical user interface;
- Analytical processing logic;
- Data-processing logic.

These components can exist on the same computer, or be distributed among several computers. Whatever the arrangement of the OLAP components, multidimensional data must be used. Opinion is divided, some favour the use of relational databases, others argue that specialized multidimensional databases are superior.

15.7.5 Relational OLAP

Relational online analytical processing (ROLAP) - provides OLAP functionality by using relational databases and familiar relational query tools to store and analyze multidimensional data. This approach builds on existing relational technologies and represents a natural extension to companies that already use relational database management systems.

ROLAP adds the following extensions to traditional RDBMS technology -

- Multidimensional data schema support within the RDBMS;

- Data access language and query performance optimized for multidimensional data;
- Support for very large databases (VLDBs).

Multidimensional data schema support within the RDBMS

Relational technology uses normalized tables to store data - stumbling block to use in OLAP systems. ROLAP uses a special design technique - star schema - that enables RDBMS technology to support multidimensional data representations.

ROLAP adds support for the star schema when familiar query tools are used, and provides advanced data analysis functions and improves query optimization and data visualization methods.

Data Access Language and Query Performance Optimized for Multidimensional data

Another criticism of relational databases is that SQL is not suited for performing advanced data analysis. Most decision support data requests require the use of multiple-pass SQL queries or multiple nested SQL statements.

ROLAP extends SQL so that it can differentiate between access requirements for data warehouse data (based on the star schema), and operational data (normalized tables). A ROLAP system therefore can generate the SQL code required to access the star schema data.

Support for Very Large Databases

When a relational database is used in a decision support role, it must be able to store very large amounts of data. Both the storage capacity and the process of loading data into the warehouse are crucial. Therefore the RDBMS must have the proper tools to import, integrate, and populate the data warehouse with data.

ROLAP is a logical choice for companies that already use relational databases for their operational data. Most current RDBMS vendors have extended their products to support data warehouses and OLAP capabilities.

15.7.6 Multi-Dimensional OLAP

Multidimensional online analytical processing (MOLAP) - extends OLAP functionality to *multidimensional database management systems* (MDBMSs). An MDBMS uses proprietary techniques to store data in matrix-like n-dimensional arrays. MOLAPs premise is that multidimensional databases are best suited to manage, store, and analyze multidimensional data.

Data cube - MDBMS users visualize the stored data as a three-dimensional cube. A data cube can grow to n number of dimensions - *hypercube*. Data cubes are created by extracting data from operational databases or the data warehouse. An important characteristic is that data cubes are static, they must be created before they can be used.

Data cube creation is critical and requires in-depth front-end design work. To speed up access, data cubes are normally held in memory in the *cube cache*.

Multidimensional data analysis is also affected by how the database system handles sparsity.

Sparsity - measures the density of the data held in the data cube, it is computed by dividing the total number of actual values in the cube by its total number of cells.

Because the data cubes' dimensions are predefined, not all cells are populated, some cells are empty. Multidimensional databases must handle sparsity effectively to reduce processing overhead and resource requirements.

15.7.7 Relational vs Multidimensional OLAP

Characteristic	**ROLAP**	**MOLAP**
Schema	Uses star schema. Additional dimensions can be added dynamically.	Uses data cubes. Multidimensional arrays, row stores, column stores. Additional dimensions require re-creation of the data cube.
Database size	Medium to large.	Large.
Architecture	Client / server. Standards-based.	Client / server. Open or proprietary, depending on vendor.
Access	Supports ad hoc requests. Unlimited dimensions.	Limited to predefined dimensions. Proprietary access languages.
Speed	Good with small data sets; average for medium to large data sets.	Faster for large data sets with predefined dimensions.

15.8 SQL Extensions for OLAP

Some new SQL extensions have been created to support OLAP-type data manipulation.
Two extensions to the GROUP BY clause are particularly useful -

15.8.1 The Rollup Extension

The ROLLUP extension is used with the GROUP BY clause to generate aggregates by different dimensions.

```
SELECT  column1, column2 ....

FROM  table 1, table2 ...

[WHERE condition]

GROUP BY ROLLUP ( column1, column2 ...)

[HAVING condition]

[ORDER BY  column1, column2 ...]
```

Order within GROUP BY ROLLUP is important - last column in the list will generate a grand total, all other columns will generate sub-totals.

Eg: Within a location hierarchy can use ROLLUP to generate sub-totals by Region, State, City, and Store.

15.8.2 The Cube Extension

The CUBE extension is used within the GROUP BY clause to generate aggregates by the listed columns, including the last one.

```
SELECT  column1, column2 ...

FROM  table1, table2 ...

[WHERE condition]

GROUP BY CUBE (column1, column2 ...)

[HAVING condition]

[ORDER BY column1, column2 ...]
```

The CUBE extension is useful when you want to compute all possible sub-totals within groupings based on multiple dimensions.

15.8.3 Materialized Views

Creating multiple summary Fact tables that use GROUP BY queries with multiple join tables could become quite resource-intensive. Normally, when new data is added to the base Fact tables, the summary Fact tables have to be recreated. This requires that the SQL code be run again to re-create all summary rows, even when only a few rows need to be changed. To save query time, most database vendors have implemented additional functions to manage aggregate summaries more efficiently.

Materialized view - is a dynamic table that not only contains the SQL query command to generate the rows, it stores the actual rows. The materialized view is created the first time the query is run, and the summary rows are stored in the table. The materialized view rows are automatically updated when the base tables are updated.

--ooOoo--

Key Terms

- **algorithms** - A process or set of operations in a calculation. The most common algorithms used in data mining are based on neural networks, decision trees, rules induction, genetic algorithms, classification and regression trees, memory-based reasoning, and nearest neighbour.

- **attribute hierarchy** - A top-down data organization that is used for two main purposes: aggregation and drill-down / roll-up data analysis.

- **business intelligence** (BI) - A comprehensive, cohesive and integrated set of tools and processes used to capture, collect, integrate, store, and analyze data with the purpose of generating and presenting information to support business decision making.

- **cube cache** - In a multi-dimensional OLAP, the shared, reserved memory area where data cubes are held. Using the cube cache assists in speeding up data access.

- **dashboard** - In business intelligence, a Web-based system that presents key business performance indicators or information in a single, integrated view with clear and concise graphics.

- **data analytics** - A subset of business intelligence functions that encompasses a wide range of mathematical, statistical, and modelling techniques with the purpose of extracting knowledge from data.

- **data cube** - The multi-dimensional data structure used to store and manipulate data in a multi-dimensional DBMS. The location of each data value in the data cube is based on its x-, y-, and z-axes. Data cubes are static, meaning they must be created before they are used, so they cannot be created by an ad hoc query.

- **data mart** - A small, single-subject data warehouse subset that provides decision support to a small group of people.

- **data mining** - A process that employs automated tools to analyze data in a data warehouse and other sources, and to proactively identify possible relationships and anomalies.

- **data warehouse** - An integrated, subject-oriented, time-variant, non-volatile collection of data that provides support for decision making, according to Bill Inmon, the acknowledged "father of the data warehouse".

- **decision support system** (DSS) - An arrangement of computerized tools used to assist managerial decision making within a business.

- **dimension tables** - In a data warehouse, tables used to search, filter, or classify facts within a star schema. The fact table is in a one-to-many relationship with dimension tables.

- **dimensions** - In a star schema design, qualifying characteristics that provide additional perspectives to a given fact.

- **drill down** - To decompose data into more atomic components - that is, data at lower levels of aggregation. This approach is used primarily in a decision support system to focus on specific geographic areas, business types, and so on. See also *roll up*.

- **explanatory analytics** - Data analysis that provides ways to discover relationships, trends, and patterns among data.

- **extraction, transformation, and loading** (ETL) - In a data warehousing environment, the integrated processes of getting data from original sources into the data warehouse. ETL includes retrieving data from original data sources (extraction), manipulating the data into an appropriate form (transformation), and storing the data in the data warehouse (loading).
- **fact table** - In a data warehouse, the star schema table that contains facts linked and classified through their common dimensions. A fact table is in a one-to-many relationship with each associated dimension table.
- **facts** - In a data warehouse, the measurements (values) that represent a specific business aspect or activity. For example, sales figures are numeric measurements that represent product or service sales. Facts commonly used in business data analysis include units, costs, prices, and revenues.
- **governance** - In business intelligence, the methods for controlling and monitoring business health and promoting consistent decision making.
- **key performance indicators** (KPIs) - In business intelligence, quantifiable numerical scale-based measurements that assess a companys' effectiveness or success in reaching strategic and operational goals. Examples of KPI are product turnovers, sales by promotion, sales by employee, and earnings per share.
- **master data management** (MDM) - In business intelligence, a collection of concepts, techniques, and processes for the proper identification, definition, and management of data elements within an organization.
- **materialized view** - A dynamic table that not only contains the SQL query command to generate rows, but stores the actual rows. The materialized view is created the first time the query is run and the summary rows are stored in the table. The materialized view rows are automatically updated when the base tables are updated.
- **metric**s - In a data warehouse, numeric facts that measure a business characteristic of interest to the end user.
- **multi-dimensional database management system** (MDDBMS) - A database management system that uses proprietary techniques to store data in matrix like arrays of n dimensions known as cubes.
- **multi-dimensional online analytical processing** (MOLAP) - An extension of on-line analytical processing to multi-dimensional database management systems.
- **online analytical processing** (OLAP) - Decision support system (DSS) tools that use multi-dimensional data analysis techniques. OLAP creates an advanced data analysis environment that supports decision making, business modelling, and operations research.
- **partitioning** - The process of splitting a table into subsets of rows or columns.
- **periodicity** - Information about the time span of data stored in a table, usually expressed as current year only, previous years, or all years.
- **portal** - In terms of business intelligence, a unified, single point of entry for information distribution.
- **predictive analysis** - Data analysis that uses advanced statistical and modelling techniques to predict future business outcomes with great accuracy.

- **relational online analytical processing** (ROLAP) - Analytical processing functions that use relational databases and familiar relational query tools to store and analyze multi-dimensional data.
- **replication** - The process of creating and managing duplicate versions of a database. Replication is used to place copies in different locations and to improve access time and fault tolerance.
- **roll up** - In SQL, an OLAP extension used with the GROUP BY clause to aggregate data by different dimensions. Rolling upcthe data is the exact opposite of drilling down the data. See also *drill down*.
- **slice and dice** - The ability to cut slices off a data cube (drill down or drill up) to perform a more detailed analysis.
- **snowflake schema** - A type of star schema in which dimension tables can have their own dimension tables. The snowflake schema is usually the result of normalizing dimension tables.
- **sparsity** - In multi-dimensional data analysis, a measurement of the data density held in the data cube.
- **star schema** - A data modelling technique used to map multi-dimensional decision support data into a relational database. The star schema represents data using a central table known as a fact table in a 1:M relationship with one or more dimension tables.
- **very large databases** (VLDBs) - Databases that contain huge amounts of data - gigabyte, terabyte, and petabyte ranges are not unusual.

--ooOoo--