BCA-09

Vardhaman Mahaveer Open University, Kota

Introduction to DBMS

BCA-09

Vardhaman Mahaveer Open University, Kota

Introduction to DBMS

Preface

We feel great in brining out this book **"Introduction to DBMS"**, which meets the requirement of students of BCA Part-II. This book is written entirely according to the syllabus of Vardhaman Mahaveer Open University, Kota.

Importance of data is well known. Using facilities of computers large databases are easily handled. The database concepts, technology and architectures have been developed in the last few decades and a course in database management has become established as a required course in both undergraduate and graduate level. This is an important aspect and it should be, considering the central position of the database in the field of computer science and computer applications. A computer professional should be able to understand fundamentals of the database to be successful in the field.

This book provides basic concepts of databases dealing with the complete syllabus. It offers a balanced view of concepts and details with reference to current technology. Database architectures, their design and manipulation that are required to make effective use of database are explained.

Unit - 1 : Introduction to Database Management System

Structure of the Unit

1.0 Objective

At the end of this unit, we should be able to -

- Describe data, field, record and database management system
- Compare database and file oriented approach
- Describe various features of database
- Describe the need and function of the database
- Describe the concept of different keys

1.1 Introduction

Data and its storage may be considered to be the heart of any information system. Data has to be up-to-date, accurate, accessible in the required form and available to one or perhaps many users at the same time.

For data to be a value it must be presented in a form that supports the various operational, financial, managerial, decision-making, administrative and clerical activities within an organization.

To meet these objectives data needs to be stored efficiently that means we wish to avoid lengthy access times and with minimal duplication that means we wish to aovid lengthy update times and the possibility of inconsistency and inaccuracy. For the data stored by a given organization to have any value at all its integrity (consistency and accuracy) must always be assured.

In this unit we consider what is database and DBMS, different needs and functions of database and concepts related to relational database management system.

1.2 Data

Data are the raw facts or figures which are processed to get the information.

1.3 Field

A space allocated for a particular item of information. A tax form, for example, contains a number of fields: one for your name, one for your Social Security number, one for your income, and so on. In database systems, fields are the

smallest units of information you can access. In spreadsheets, fields are called cells

1.4 Record

In database management systems, a complete set of information. Records are composed of fields, each of which contains one item of information. A set of records constitutes a file. For example, a personnel file might contain records that have three fields: a name field, an address field, and a phone number field.

1.5 Data Base

A collection of data stored in a standardized format, designed to be processed, shared by different users. A database may have single table or multiple tables. The data in a database are organized in rows and columns

A collection of information organized in such a way that a computer program can quickly select desired pieces of data. You can think of a database as an electronic filing system.

1.6 Database Management System (DBMS)

It is software that defines a database, stores the data and supports a query language, produces reports,and creates data entry forms.

It is a software package with computer programs that control the creation, maintenance, and the use of a database. It allows organizations to conveniently develop databases for various applications by database administrators (DBAs) and other specialists.

It allows a computer to perform database functions of storing, retrieving, adding, deleting and modifying data.

It allows different user application programs to concurrently access the same database which is an integrated collection of data records, files, and other database objects.

It provides facilities for controlling data access, enforcing data integrity, managing concurrency control, recovering the database after failures and restoring it from backup files, as well as maintaining database security.

Some DBMS software are MS-Access, Oracle, FoxPro, IBM DB2 , MS ACCESS, MYSQL, SYBASE etc.

1.7 Elements of Database Management Systems

The essential elements that are found with just about every example of DBMS currently on the market are

- Schema objects
- Indexes
- Tables
- Fields and columns
- Records and rows
- Keys
- Relationships
- Data types

1.8 Database Schema

A schema is a group of related objects in a database. Within a schema, objects that are related have relationships to one another. It is overall design of the database. There is one owner of a schema, who has access to manipulate the structure of any object in the schema. A schema does not represent a person, although the schema is associated with a user account that resides in the database.

The three models associated with a schema are as follows:

- The conceptual model, also called the logical model, is the basic database model, which deals with organizational structures that are used to define database structures such as tables and constraints.
- The internal model, also called the physical model, deals with the physical storage of the database, as well as access to the data, such as through data storage in tables and the use of indexes to expedite data access. The internal model separates the physical requirements of the hardware and the operating system from the data model.
- The external model, or application interface, deals with methods through which users may access the schema, such as through the use of a data input form. The external model allows relationships to be created between the user application and the data model. Figure 1 depicts a schema in a relational database.

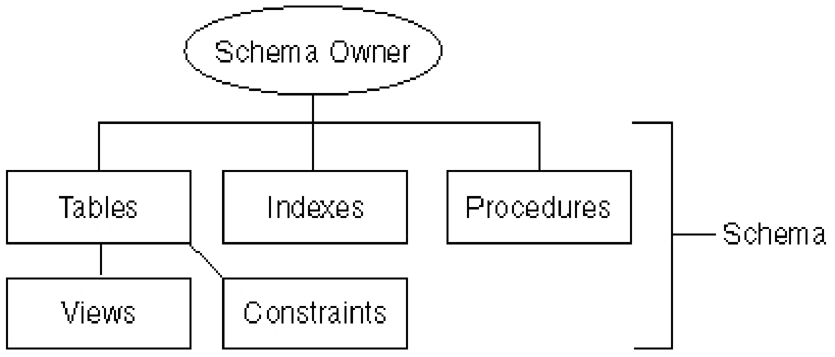


Figure 1 : Collection of objects that comprise a database schema.

1.9 Table

A table is the primary unit of physical storage for data in a database. The table is the most fundamental element found in a database schema. Columns and rows are associated with tables When a user accesses the database, a table is usually referenced for the desired data. Multiple tables might comprise a database, therefore a relationship might exist between tables. Because tables store data, a table requires physical storage on the host computer for the database.

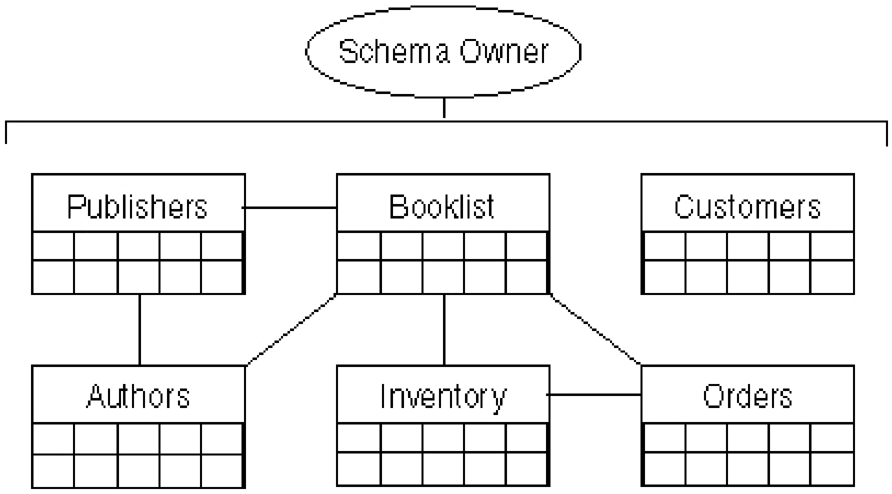


Figure 2 : Database Tables and their Relationships

1.10 Columns

A column, or field, is a specific category of information that exists in a table. A column is to a table what an attribute is to an entity. In other words, when a business model is converted into a database model, entities become tables and attributes become columns. A column represents one related part of a table and is the smallest logical structure of storage in a database. Each column in a table is assigned a data type. The assigned data type determines what type of values that can populate a column. When visualizing a table, a column is a vertical structure in the table that contains values for every row of data associated with a particular column. All of the data in a table associated with a field is called a column.

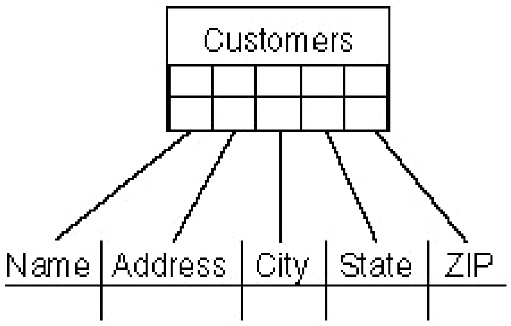


Figure 3 : Columns in a Database Table

1.11 Rows

A row of data is the collection of all the columns in a table associated with a single occurrence. A row of data is a single record in a table. For example, if there are 25,000 book titles with which a bookstore deals, there will be 25,000 records, or rows of data, in the book titles table once the table is populated. The number of rows within the table will obviously change as books' titles are added and removed.

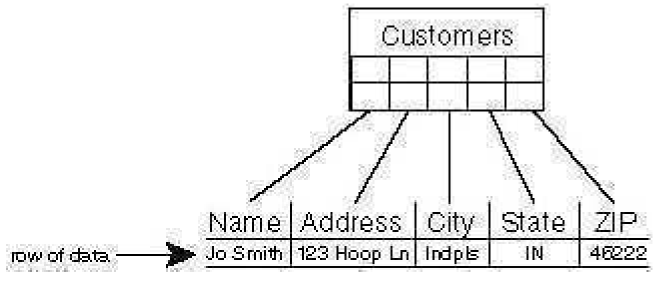


Figure 4 : Row of data in a database table

1.12 Data Types

A data type determines the type of data that can be stored in a database column.

Although many data types are available, three of the most commonly used data types are

- Alphanumeric
- Numeric
- Date and time

Alphanumeric data types are used to store characters, numbers, special characters, or nearly any combination. If a numeric value is stored in an alphanumeric field, the value is treated as a character, not a number. In other words, you should not attempt to perform arithmetic functions on numeric values stored in alphanumeric fields. Numeric data types are used to store only numeric values.

Date and time data types are used to store date and time values, which widely vary depending on the relational database management system (RDBMS) being used.

1.13 Keys

The integrity of the information stored in a database is controlled by keys. A key is a column value in a table that is used to either uniquely identify a row of data in a table, or establish a relationship with another table. A key is normally correlated with one column in table, although it might be associated with multiple columns. There are two types of keys: primary and foreign.

1.13.1 Primary Keys

A primary key is the combination of one or more column values in a table that make a row of data unique within the table. Primary keys are typically used to join related tables. Even if a table has no child table, a primary key can be used to disallow the entry of duplicate records into a table. For example, an employee's social security number is sometimes considered a primary key candidate because all SSNs are unique.

1.13.2 Foreign Keys

A foreign key is the combination of one or more column values in a table that reference a primary key in another table. Foreign keys are defined in child tables. A foreign key ensures that a parent record has been created before a child record. Conversely, a foreign key also ensures that the child record is deleted before the parent record.

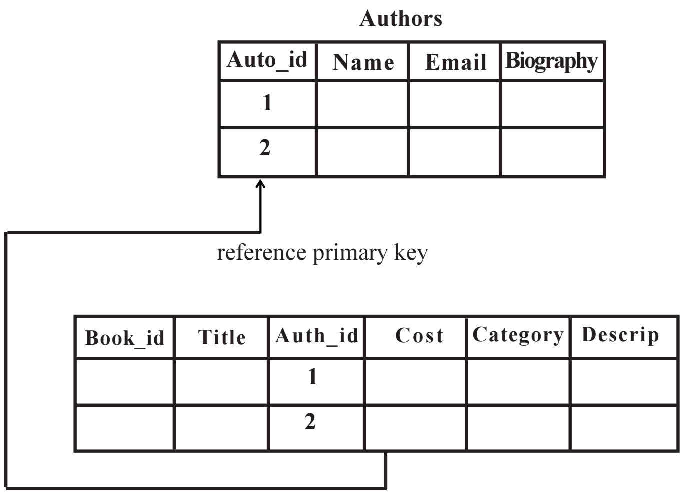


Figure 5 : Referential Integrity, or Parent/Child Relationships.

1.14 Relationship

DBMS relationships depend on the set-up and use of the DBMS. A relationship exists between two database tables when one table has a foreign key that references the primary key of another table.

Types of Database Relationships

There are three different types of database relationships, each named according to the number of table rows that may be involved in the relationship. Each of these three relationship types exists between two tables.

- One-to-one relationships occur when each entry in the first table has one, and only one, counterpart in the second table. One-to-one relationships are rarely used because it is often more efficient to simply put all of the information in a single table.

one record in a table is related to one record in a related table; creates equally dependent tables

Ex. one student has only one PSU ID

- One-to-many relationships are the most common type of database relationship. They occur when each record in the first table corresponds to one or more records in the second table but each record in the second table corresponds to only one record in the first table. For example, the relationship between a Teachers table and a Students table in an elementary school database would likely be a one-to-many relationship, because each student has only one teacher, but each teacher may have multiple students.
- Many-to-many relationships occur when each record in the first table corresponds to one or more records in the second table and each record in the second table corresponds to one or more records in the first table. For example, one student can be enrolled in many courses and each course can have many students enrolled

1.15 Database Management System Vs File Management System

A Database Management System (DMS) is a combination of computer software, hardware, and information designed to electronically manipulate data via computer processing. Two types of database management systems are DBMS's and FMS's. In simple terms, a File Management System (FMS) is a Database Management System that allows access to single files or tables at a time. FMS's accommodate flat files that have no relation to other files. The FMS was the predecessor for the Database Management System (DBMS), which allows access to multiple files or tables at a time.

File Management Systems

Advantages	Disadvantages
Simpler to use	Typically does not support multi-user access
Less expensive·	Limited to smaller databases
Fits the needs of many small businesses	
and home users	Limited functionality (i.e. no support for complicated transactions, recovery, etc.)

Popular FMS's are packaged along with the operating systems of personal computers (i.e. Microsoft Cardfile and Microsoft Works) Decentralization of data Good for database solutions for hand held devices such as Palm Pilot Redundancy and Integrity issues Typically, File Management Systems provide the following advantages and disadvantages:

The goals of a File Management System can be summarized as:

- Data Management. An FMS should provide data management services to the application.
- Generality with respect to storage devices. The FMS data abstractions and access methods should remain unchanged irrespective of the devices involved in data storage.
- Validity. An FMS should guarantee that at any given moment the stored data reflect the operations performed on them.
- Protection. Illegal or potentially dangerous operations on the data should be controlled by the FMS.
- Concurrency. In multiprogramming systems, concurrent access to the data should be allowed with minimal differences.
- Performance. Compromise data access speed and data transfer rate with functionality.

From the point of view of an end user (or application) an FMS typically provides the following functionalities:

- File creation, modification and deletion.
- Ownership of files and access control on the basis of ownership permissions.
- Facilities to structure data within files (predefined record formats, etc).
- Facilities for maintaining data redundancies against technical failure (back-ups, disk mirroring, etc.).
- Logical identification and structuring of the data, via file names and hierarchical directory structures.

1.16 Database Management Systems

Database Management Systems provide the following advantages and disadvantages:

Advantages	**Disadvantages**
Greater flexibility	Difficult to learn
Good for larger databases	Packaged separately from the operating system (i.e. Oracle, Microsoft Access, Lotus/IBM Approach, Borland Paradox, Claris FileMaker Pro)
Greater processing power	Slower processing speeds
Fits the needs of many medium to large-sized organizations	Requires skilled administrators
Storage for all relevant data	Expensive
Provides user views relevant to tasks performed	

Ensures data integrity by managing transactions (ACID test = atomicity, consistency, isolation, durability) Supports simultaneous access Enforces design criteria in relation to data format and structure Provides backup and recovery controls Advanced security	

The goals of a **Database Management System** can be summarized as follows:

- Data storage, retrieval, and update (while hiding the internal physical implementation details)
- A user-accessible catalog
- Transaction support
- Concurrency control services (multi-user update functionality)
- Recovery services (damaged database must be returned to a consistent state)
- Authorization services (security)
- Support for data communication Integrity services (i.e. constraints)
- Services to promote data independence
- Utility services (i.e. importing, monitoring, performance, record deletion, etc.)

The components to facilitate the goals of a DBMS may include the following:

- Query processor
- Data Manipulation Language preprocessor
- Database manager (software components to include authorization control, command processor, integrity checker, query optimizer, transaction manager, scheduler, recovery manager, and buffer manager)
- Data Definition Language compiler
- File manager
- Catalogue Manager

1.17 Advantages of DBMS

- The data independence and efficient access of data.
- It reduces application development time
- It provides data integrity and security
- Easy in data administration or data management
- Provides concurrent access, recovers the data from the crashesCentralized control

Disadvantages of DBMS:

- Problem Associate with centralizedCost of software, hardware and migration.
- Complexity of backup and recovery.

1.18 Features Commonly Offered by Database Management Systems Include

Query ability

Querying is the process of requesting attribute information from various perspectives and combinations of factors. Example: "How many 2-door cars in Texas are green?" A database query language and report writer allow users to interactively interrogate the database, analyze its data and update it according to the users privileges on data.

Backup and replication

Copies of attributes need to be made regularly in case primary disks or other equipment fails. A periodic copy of attributes may also be created for a distant organization that cannot readily access the original. DBMS usually provide utilities to facilitate the process of extracting and disseminating attribute sets. When data is replicated between database servers, so that the information remains consistent throughout the database system and users cannot tell or even know which server in the DBMS they are using, the system is said to exhibit replication transparency.

Rule enforcement

Often one wants to apply rules to attributes so that the attributes are clean and reliable. For example, we may have a rule that says each car can have only one engine associated with it (identified by Engine Number). If somebody tries to associate a second engine with a given car, we want the DBMS to deny such a request and display an error message. However, with changes in the model specification such as, in this example, hybrid gas-electric cars, rules may need to change. Ideally such rules should be able to be added and removed as needed without significant data layout redesign.

Security

For security reasons, it is desirable to limit who can see or change specific attributes or groups of attributes. This may be managed directly on an individual basis, or by the assignment of individuals and privileges to groups, or (in the most elaborate models) through the assignment of individuals and groups to roles which are then granted entitlements.

Computation

Common computations requested on attributes are counting, summing, averaging, sorting, grouping, cross-referencing, and so on. Rather than have each computer application implement these from scratch, they can rely on the DBMS to supply such calculations.

Change and access logging

This describes who accessed which attributes, what was changed, and when it was changed. Logging services allow this by keeping a record of access occurrences and changes.

Automated optimization

For frequently occurring usage patterns or requests, some DBMS can adjust themselves to improve the speed of those interactions. In some cases the DBMS will merely provide tools to monitor performance, allowing a human expert to make the necessary adjustments after reviewing the statistics collected.

1.19 Features of DBMS A Typical DBMS has the Following Features

- Provides a way to structure data as records, tables, or objects
- Accepts data input from operators and stores that data for later retrieval
- Provides query languages for searching, sorting, reporting, and other "decision support" activities that help users correlate and make sense of collected data
- Provides multiuser access to data, along with security features that prevent some users from viewing and/or changing certain types of information
- Provides data integrity features that prevent more than one user from accessing and changing the same information simultaneously

- Provides a data dictionary (metadata) that describes the structure of the database, related files, and record information

1.20 Summary

A **database** is an organized collection of data and the records, which is stored in a system, usually a computer. The organization of data is achieved by structuring it according to the model of the database.

A DBMS can take any one of the several approaches to manage data. Each approach constitutes a database model. A **data model** is a collection of descriptions of data structures and their contained fields, together with the operations or functions that manipulate them. A data model is a comprehensive scheme for describng how data is to be represented for manipulation by humans or computer programs.

DBA stands for **database administrator,** he is a person who is responsible for the features of a databse like databse recovery, integrity, availability, security and so on. DBA has to check these features in order to maintain the database.

A **key** is a field or combination of fields used to identify a record. When a key uniquely identifies a record it is referred to as the **primary key.**

1.21 Self-Assessment Questions

1. What do you understand by DBMS? Explain.
2. How file management system is different from DBMS? Explain.
3. List the advantages of DBMS.
4. What is Key? How primary key is different from foreign key?
5. Explain the concept of relationship with the help of an example.

′ ′ ′

Unit - 2 : Basic concepts of DBMS

Structure of the Unit

2.0 Objective

At the end of this unit, we should be abel to -

- Describe functions of database management system
- Describe the role of DBA, End users and application programmes
- Describe data independence
- Describe database schema and instance
- Describe DDL and DML commands

2.1 Introduction

A database management system (DBMS) is a software system that integrates data in storage and provides easy access to them. The data themselves are placed on disk in a database, which can be thought of as an integrated collection of related files. Although not all databases are organized identically, many of them are composed of files, records and fields. For instance, the product file contains five records one each for skis, boots, poles, bindings, and wax. Finally each record consists of distinct types of data called fileds. The product file stores four fileds for each record product name, product number, supplier, and price.

Database management softaware enables queries and reports to be prepared by extracting information from one file at a time, and, as we will shortly see, from several interrelated files concurrently. Database management system (DBMS) is a software product designed to integrate data and provide easy access to them. Database is an integrated collection of related data files and database management system is a computer programe for database management that links data in related files through common fileds. Database management system stores data in relevant form and give easy access to them. The data placed on disk themselves.

2.2 Purpose of Database Management System

The DBMS (Database Management System) is preferred ever the conventional file processing system due to the following advantages:

1. **Controlling Data Redundancy -** In the conventional file processing system, every user group maintains its own files for handling its data files. This may lead to:

- Duplication of same data in different files.
- Wastage of storage space, since duplicated data is stored.
- Errors may be generated due to updation of the same data in different files.
- Time in entering data again and again is wasted.
- Computer Resources are needlessly used.
- It is very difficult to combine information.

2. **Elimination of Inconsistency -** In the file processing system information is duplicated throughout the system. So changes made in one file may be necessary be carried over to another file. This may lead to inconsistent data. So we need to remove this duplication of data in multiple file to eliminate inconsistency.

For example: - Let us consider an example of student's result system. Suppose that in STUDENT file it is indicated that Roll no= 10 has opted for 'Computer'course but in RESULT file it is indicated that 'Roll No. =l 0' has opted for 'Accounts' course. Thus, in this case the two entries for z particular student don't agree with each other. Thus, database is said to be in an inconsistent state. Sc to eliminate this conflicting information we need to centralize the database. On centralizing the data base the duplication will be controlled and hence inconsistency will be removed.

Data inconsistency are often encountered in every day life Consider an another example, w have all come across situations when a new address is communicated to an organization that we deal it (Eg - Telecom, Gas Company, Bank). We find that some of the communications from that organization are received at a new address while other continued to be mailed to the old address. So combining all the data in database would involve reduction in redundancy as well as inconsistency so it is likely to reduce the costs for collection storage and updating of Data.

Let us again consider the example of Result system. Suppose that a student having Roll no -201 changes his course from 'Computer' to 'Arts'. The change is made in the SUBJECT file but not in RESULT'S file. This may lead to inconsistency of the data. So we need to centralize the database so that changes once made are reflected to all the tables where a particulars field is stored. Thus the update is brought automatically and is known as propagating updates.

3. **Better service to the users -** A DBMS is often used to provide better services to the users. In conventional system, availability of information is often poor, since it normally difficult to obtain information that the existing systems were not designed for. Once several conventional systems are combined to form one centralized database, the availability of information and its updateness is likely to improve since the

data can now be shared and DBMS makes it easy to respond to anticipated information requests.

Centralizing the data in the database also means that user can obtain new and combined information easily that would have been impossible to obtain otherwise. Also use of DBMS should allow users that don't know programming to interact with the data more easily, unlike file processing system where the programmer may need to write new programs to meet every new demand.

4. **Flexibility of the System is Improved -** Since changes are often necessary to the contents of the data stored in any system, these changes are made more easily in a centralized database than in a conventional system. Applications programs need not to be changed on changing the data in the database.

5. **Integrity can be improved -** Since data of the organization using database approach is centralized and would be used by a number of users at a time. It is essential to enforce integrity-constraints.

In the conventional systems because the data is duplicated in multiple files so updating or changes may sometimes lead to entry of incorrect data in some files where it exists.

For example : - The example of result system that we have already discussed. Since multiple files are to maintained, as sometimes you may enter a value for course which may not exist. Suppose course can have values (Computer, Accounts, Economics, and Arts) but we enter a value 'Hindi' for it, so this may lead to an inconsistent data, so lack of Integrity.

Even if we centralized the database it may still contain incorrect data. For example: -

- Salary of full time employ may be entered as Rs. 500 rather than Rs. 5000.
- A student may be shown to have borrowed books but has no enrollment.
- A list of employee numbers for a given department may include a number of non existent employees.

These problems can be avoided by defining the validation procedures whenever any update operation is attempted.

6. **Standards can be enforced -** Since all access to the database must be through DBMS, so standards are easier to enforce. Standards may relate to the naming of data, format of data, structure of the data etc. Standardizing stored data formats is usually desirable for the purpose of data interchange or migration between systems.

7. **Security can be improved -** In conventional systems, applications are developed in an adhoc/ temporary manner. Often different system of an organization would access different components of the operational data, in such an environment enforcing security can be quiet difficult. Setting up of a database makes it easier to enforce security restrictions since data is now centralized. It is easier to control who has access to what parts of the database. Different checks can be established for each type of access (retrieve, modify, delete etc.) to each piece of information in the database.

Consider an Example of banking in which the employee at different levels may be given access to different types of data in the database. A clerk may be given the authority to know only the names of all the customers who have a loan in bank but not the details of each loan the customer may have. It can be accomplished by giving the privileges to each employee.

8. **Organization's requirement can be identified -** All organizations have sections and departments and each of these units often consider the work of their unit as the most important and therefore consider their need as the most important. Once a database has been setup with centralized control, it will be necessary to identify organization's requirement and to balance the needs of the competating units. So it may become necessary to ignore some requests for information if they conflict with higher priority need of the organization.

It is the responsibility of the DBA (Database Administrator) to structure the database system to provide the overall service that is best for an organization.

For example: - A DBA must choose best file Structure and access method to give fast response for the high critical applications as compared to less critical applications.

9. Overall cost of developing and maintaining systems is lower - It is much easier to respond to unanticipated requests when data is centralized in a database than when it is stored in a conventional file system. Although the initial cost of setting up of a database can be large, one normal expects the overall cost of setting up of a database, developing and maintaining application programs to be far lower than for similar service using conventional systems, Since the productivity of programmers can be higher in using non-procedural languages that have been developed with DBMS than using procedural languages.

10. Data Model must be developed - Perhaps the most important advantage of setting up of database system is the requirement that an overall data model for an organization be build. In conventional systems, it is more likely that files will be designed as per need of particular applications demand. The overall view is often not considered. Building an overall view of an organization's data is usual cost effective in the long terms.

11. Provides backup and Recovery - Centralizing a database provides the schemes such as recovery and backups from the failures including disk crash, power failures, software errors which may help the database to recover from the inconsistent state to the state that existed prior to the occurrence of the failure, though methods are very complex.

2.3 Functions of DBMS

1. DBMS free the programmers from the need to worry about the organization and location of the data i.e. it shields the users from complex hardware level details.

2. DBMS can organize process and present data elements from the database. This capability enables decision makers to search and query database contents in order to extract answers that are not available in regular Reports.

3. Programming is speeded up because programmer can concentrate on logic of the application.

4. It includes special user friendly query languages which are easy to understand by non programming users of the system.

The various common examples of DBMS are Oracle, Access, SQL Server, Sybase, FoxPro, Dbase etc.

2.4 The Service Provided by the DBMS Includes

1. Authorization services like log on to the DBMS, start the database, stop the Database etc.
2. Transaction supports like Recovery, Rollback etc,
3. Import and Export of Data.
4. Maintaining data dictionary
5. User's Monitoring

2.5 DBA, Database Designers, End Users & Application Programmers

2.5.1 Database Administrator (DBA)

The DBA is a person or a group of persons who is responsible for the management of the database. The DBA is responsible for authorizing access to the database by grant and revoke permissions to the users, for coordinating and monitoring its use, managing backups and repairing damage due to hardware and/or software failures and for acquiring hardware and software resources as needed. In case of small organization the role of DBA is performed by a single person and in case of large organizations there is a group of DBA's who share responsibilities.

2.5.2 Database Designers

They are responsible for identifying the data to be stored in the database and for choosing appropriate structure to represent and store the data. It is the responsibility of database designers to communicate with all prospective of the database users in order to understand their requirements so that they can create a design that meets their requirements.

2.5.3 End Users

End Users are the people who interact with the database through applications or utilities. The various categories of end users are:

2.5.3.1 Casual End Users : These Users occasionally access the database but may need different information each time. They use sophisticated database Query language to specify their requests. For example: High level Managers who access the data weekly or biweekly.

2.5.3.2 NativeEnd Users : These users frequently query and update the database using standard types of Queries. The operations that can be performed by this class of users are very limited and effect precise portion of the database.

For example: - Reservation clerks for airlines/hotels check availability for given request and make reservations. Also, persons using Automated Teller Machines (ATM's) fall under this category as he has access to limited portion of the database.

2.5.3.3 Standalone end Users/On-line End Users : Those end Users who interact with the database directly via on-line terminal or indirectly through Menu or graphics based Interfaces.

For example: - User of a text package, library management software that store variety of library data such as issue and return of books for fine purposes.

2.5.4 Application Programmers

Application Programmers are responsible for writing application programs that use the database. These programs could be written in General Purpose Programming languages such as Visual Basic, Developer, C, FORTRAN, COBOL etc. to manipulate the database. These application programs operate on the data to perform various operations such as retaining information, creating new information, deleting or changing existing information.

- DBMS engine accepts logical requests from various other DBMS subsystems, converts them into physical equivalents, and actually accesses the database and data dictionary as they exist on a storage device.

- Data definition subsystem helps the user create and maintain the data dictionary and define the structure of the files in a database.

- Data manipulation subsystem helps the user to add, change, and delete information in a database and query it for valuable information. Software tools within the data manipulation subsystem are most often the primary interface between user and the information contained in a database. It allows the user to specify its logical information requirements.

- Application generation subsystem contains facilities to help users develop transaction-intensive applications. It usually requires that the user perform a detailed series of tasks to process a transaction. It facilitates easy-to-use data entry screens, programming languages, and interfaces.

- Data administration subsystem helps users manage the overall database environment by providing facilities for backup and recovery, security management, query optimization, concurrency control, and change management.

2.6 DBMS-Architecture and Data Independence

Database management systems are complex softwares which were often developed and optimised over years. From the view of the user, however, most of them have a quite similar basic architecture.

Three-Schemes Architecture

Knowing about the conceptual and the derived logical scheme (discussed in unit Database Models, Schemes and Instances this unit explains two additional schemes - the external scheme and the internal scheme - which help to understand the DBMS architecture.

External Scheme:

An external data scheme describes the information about the user view of specific users (single users and user groups) and the specific methods and constraints connected with this information. (Translated) (ZEHNDER 1998)

Internal Scheme:

The internal data scheme describes the content of the data and the required service functionality which is used for the operation of the DBMS. (Translated) (ZEHNDER 1998)

Therefore, the internal scheme describes the data from a view very close to the computer or system in general. It completes the logical scheme with data technical aspects like storage methods or help functions for more efficiency.

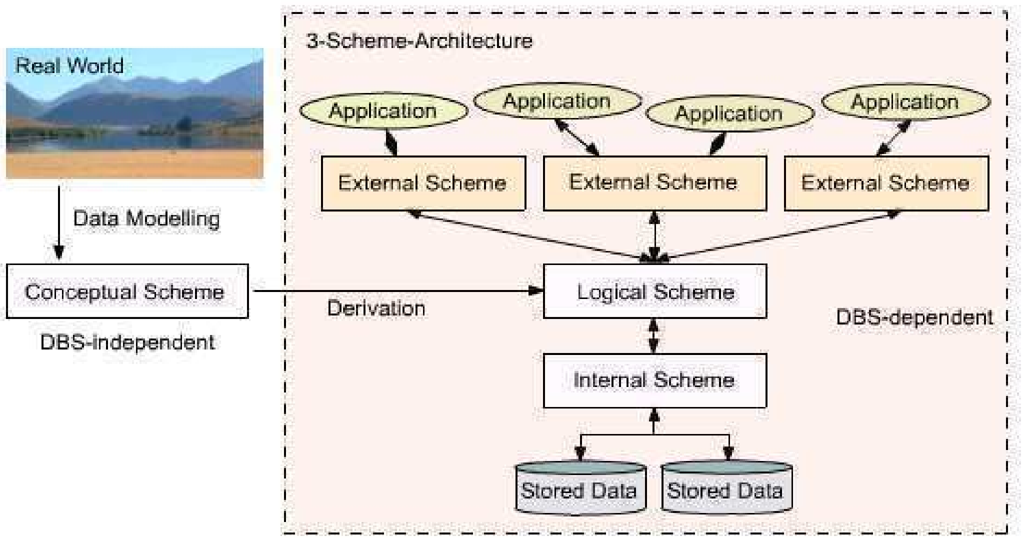


Figure 2.1 : Three-Schemes Architecture

The right hand side of the representation above is also called the three-schemes architecture: internal, logical and external scheme.

While the internal scheme describes the physical grouping of the data and the use of the storage space, the logical scheme (derived from the conceptual scheme) describes the basic construction of the data structure. The external scheme of a specific application, generally, only highlights that part of the logical scheme which is relevant for its application. Therefore, a database has exactly one internal and one logical scheme but may have several external schemes for several applications using this database.

The aim of the three-schemes architecture is the separation of the user applications from the physical database, the stored data. Physically the data is only existent on the internal level while other forms of representation are calculated or derived respectively if needed. The DBMS has the task to realise this representation between each of these levels.

2.7 Data Independence

With knowledge about the three-schemes architecture the term data independence can be explained as followed: Each higher level of the data architecture is immune to changes of the next lower level of the architecture.

2.7.1 Physical Independence

Therefore, the logical scheme may stay unchanged even though the storage space or type of some data is changed for reasons of optimisation or reorganisation.

2.7.2 Logical Independence

Also the external scheme may stay unchanged for most changes of the logical scheme. This is especially desirable as in this case the application software does not need to be modified or newly translated.

2.8 Database Schema

Definition : Overall design of data base. Schema contains 'No of records + Type of data + No of attributes'

1. External level or Sub schema
2. logical schema
3. Physical schema

2.9 Database Instance

The term instance is typically used to describe a complete database environment, including the RDBMS software, table structure, stored procedures and other functionality. It is most commonly used when administrators describe multiple instances of the same database.

The information stored in database at the particular movement is called instance.

Also Known As: environment

Examples: An organization with an employees database might have three different instances: production (used to contain live data), pre-production (used to test new functionality prior to release into production) and development (used by database developers to create new functionality).

There are two different types of languages to make database system. They are 1. To specify the database schema, and 2.to express database queries and updates.

2.10 Data-Definition Language

A database schema is specified by a set of definitions expressed by special language called a data-definition language (DDL). The result of compilation of DDL statements is a set of tables that is stored in a special file called Data dictionary or data directory.

A data dictionary is a file that contains metadata that is, data about data. This file is consulted before actual data are read or modified in the database system. The storage structure and access methods used by the database system are specified by a set of definitions in a special type of DDL called a data storage and definition language.

2.11 Data-Manipulation Language

The levels of abstraction apply not only to the definition or structuring of data, but also the manipulation of data. By data manipulation, it means

- The retrieval of information stored in the database
- The insertion of new information into the database
- The deletion of information from the database

- The modification of information stored in the database

A data-manipulation language (DML) is a language that enables user to access or manipulate data as organized by the appropriate data model. They are basically two types.

- Procedural DMLs require a user to specify what data are needed and how to get those data
- Nonprocedural DMLs require a user to specify what data are needed without specifying how to get those data

Non procedural DMLs are usually easier to learn and use than are procedural DMLs. A query is a statement requesting the retrieval of information. The portion of a DML that involves information retrieval is called a query language.

2.12 Summary

A **database management system (DBMS)** is a software system that integrates data in storage and provides easy access to them. The data themselves are placed on disk in a database, which can be thought of as an integrated collection of related files. Although not all databases are organized identically, many of them are composed of files, records and fields.

2.13 Self-Assessment Questions

1. Explain the different function of DBMS.
2. Explain the role of DBA.
3. Explain the role of different users of DBMS.
4. What is Data Independence? Explain.
5. Explain the concept of DDL and DML commands in DBMS.

′ ′ ′

BCA-09

Vardhaman Mahaveer Open University, Kota

Introduction to DBMS

BCA-09

Vardhaman Mahaveer Open University, Kota

Introduction to DBMS

Preface

We feel great in brining out this book **"Introduction to DBMS"**, which meets the requirement of students of BCA Part-II. This book is written entirely according to the syllabus of Vardhaman Mahaveer Open University, Kota.

Importance of data is well known. Using facilities of computers large databases are easily handled. The database concepts, technology and architectures have been developed in the last few decades and a course in database management has become established as a required course in both undergraduate and graduate level. This is an important aspect and it should be, considering the central position of the database in the field of computer science and computer applications. A computer professional should be able to understand fundamentals of the database to be successful in the field.

This book provides basic concepts of databases dealing with the complete syllabus. It offers a balanced view of concepts and details with reference to current technology. Database architectures, their design and manipulation that are required to make effective use of database are explained.

Unit - 1 : Introduction to Database Management System

Structure of the Unit

1.0 Objective

At the end of this unit, we should be able to -

- Describe data, field, record and database management system
- Compare database and file oriented approach
- Describe various features of database
- Describe the need and function of the database
- Describe the concept of different keys

1.1 Introduction

Data and its storage may be considered to be the heart of any information system. Data has to be up-to-date, accurate, accessible in the required form and available to one or perhaps many users at the same time.

For data to be a value it must be presented in a form that supports the various operational, financial, managerial, decision-making, administrative and clerical activities within an organization.

To meet these objectives data needs to be stored efficiently that means we wish to avoid lengthy access times and with minimal duplication that means we wish to aovid lengthy update times and the possibility of inconsistency and inaccuracy. For the data stored by a given organization to have any value at all its integrity (consistency and accuracy) must always be assured.

In this unit we consider what is database and DBMS, different needs and functions of database and concepts related to relational database management system.

1.2 Data

Data are the raw facts or figures which are processed to get the information.

1.3 Field

A space allocated for a particular item of information. A tax form, for example, contains a number of fields: one for your name, one for your Social Security number, one for your income, and so on. In database systems, fields are the

smallest units of information you can access. In spreadsheets, fields are called cells

1.4 Record

In database management systems, a complete set of information. Records are composed of fields, each of which contains one item of information. A set of records constitutes a file. For example, a personnel file might contain records that have three fields: a name field, an address field, and a phone number field.

1.5 Data Base

A collection of data stored in a standardized format, designed to be processed, shared by different users. A database may have single table or multiple tables. The data in a database are organized in rows and columns

A collection of information organized in such a way that a computer program can quickly select desired pieces of data. You can think of a database as an electronic filing system.

1.6 Database Management System (DBMS)

It is software that defines a database, stores the data and supports a query language, produces reports,and creates data entry forms.

It is a software package with computer programs that control the creation, maintenance, and the use of a database. It allows organizations to conveniently develop databases for various applications by database administrators (DBAs) and other specialists.

It allows a computer to perform database functions of storing, retrieving, adding, deleting and modifying data.

It allows different user application programs to concurrently access the same database which is an integrated collection of data records, files, and other database objects.

It provides facilities for controlling data access, enforcing data integrity, managing concurrency control, recovering the database after failures and restoring it from backup files, as well as maintaining database security.

Some DBMS software are MS-Access, Oracle, FoxPro, IBM DB2 , MS ACCESS, MYSQL, SYBASE etc.

1.7 Elements of Database Management Systems

The essential elements that are found with just about every example of DBMS currently on the market are

- Schema objects
- Indexes
- Tables
- Fields and columns
- Records and rows
- Keys
- Relationships
- Data types

1.8 Database Schema

A schema is a group of related objects in a database. Within a schema, objects that are related have relationships to one another. It is overall design of the database. There is one owner of a schema, who has access to manipulate the structure of any object in the schema. A schema does not represent a person, although the schema is associated with a user account that resides in the database.

The three models associated with a schema are as follows:

- The conceptual model, also called the logical model, is the basic database model, which deals with organizational structures that are used to define database structures such as tables and constraints.
- The internal model, also called the physical model, deals with the physical storage of the database, as well as access to the data, such as through data storage in tables and the use of indexes to expedite data access. The internal model separates the physical requirements of the hardware and the operating system from the data model.
- The external model, or application interface, deals with methods through which users may access the schema, such as through the use of a data input form. The external model allows relationships to be created between the user application and the data model. Figure 1 depicts a schema in a relational database.

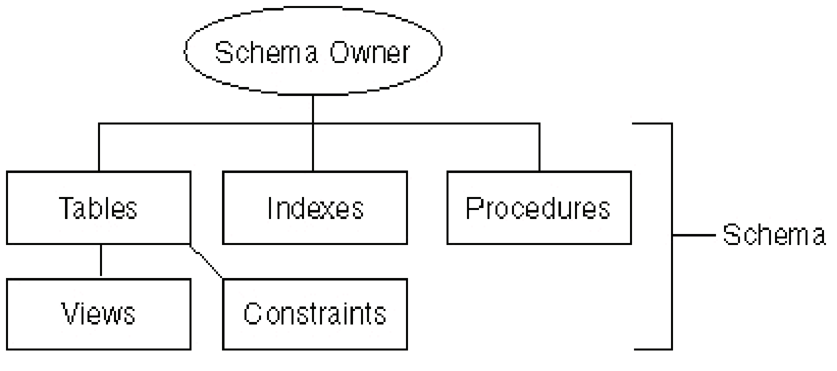


Figure 1 : Collection of objects that comprise a database schema.

1.9 Table

A table is the primary unit of physical storage for data in a database. The table is the most fundamental element found in a database schema. Columns and rows are associated with tables When a user accesses the database, a table is usually referenced for the desired data. Multiple tables might comprise a database, therefore a relationship might exist between tables. Because tables store data, a table requires physical storage on the host computer for the database.

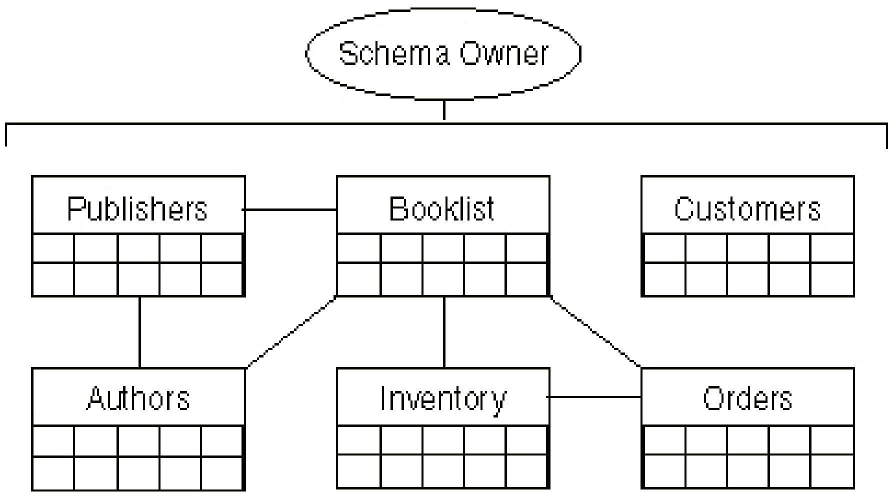


Figure 2 : Database Tables and their Relationships

1.10 Columns

A column, or field, is a specific category of information that exists in a table. A column is to a table what an attribute is to an entity. In other words, when a business model is converted into a database model, entities become tables and attributes become columns. A column represents one related part of a table and is the smallest logical structure of storage in a database. Each column in a table is assigned a data type. The assigned data type determines what type of values that can populate a column. When visualizing a table, a column is a vertical structure in the table that contains values for every row of data associated with a particular column. All of the data in a table associated with a field is called a column.

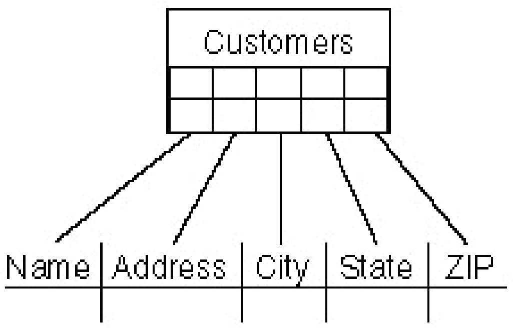


Figure 3 : Columns in a Database Table

1.11 Rows

A row of data is the collection of all the columns in a table associated with a single occurrence. A row of data is a single record in a table. For example, if there are 25,000 book titles with which a bookstore deals, there will be 25,000 records, or rows of data, in the book titles table once the table is populated. The number of rows within the table will obviously change as books' titles are added and removed.

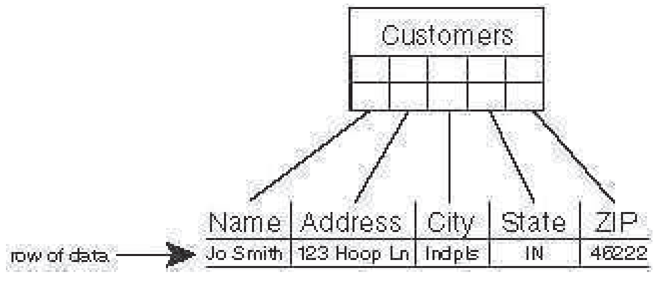


Figure 4 : Row of data in a database table

1.12 Data Types

A data type determines the type of data that can be stored in a database column.

Although many data types are available, three of the most commonly used data types are

- Alphanumeric
- Numeric
- Date and time

Alphanumeric data types are used to store characters, numbers, special characters, or nearly any combination. If a numeric value is stored in an alphanumeric field, the value is treated as a character, not a number. In other words, you should not attempt to perform arithmetic functions on numeric values stored in alphanumeric fields. Numeric data types are used to store only numeric values.

Date and time data types are used to store date and time values, which widely vary depending on the relational database management system (RDBMS) being used.

1.13 Keys

The integrity of the information stored in a database is controlled by keys. A key is a column value in a table that is used to either uniquely identify a row of data in a table, or establish a relationship with another table. A key is normally correlated with one column in table, although it might be associated with multiple columns. There are two types of keys: primary and foreign.

1.13.1 Primary Keys

A primary key is the combination of one or more column values in a table that make a row of data unique within the table. Primary keys are typically used to join related tables. Even if a table has no child table, a primary key can be used to disallow the entry of duplicate records into a table. For example, an employee's social security number is sometimes considered a primary key candidate because all SSNs are unique.

1.13.2 Foreign Keys

A foreign key is the combination of one or more column values in a table that reference a primary key in another table. Foreign keys are defined in child tables. A foreign key ensures that a parent record has been created before a child record. Conversely, a foreign key also ensures that the child record is deleted before the parent record.

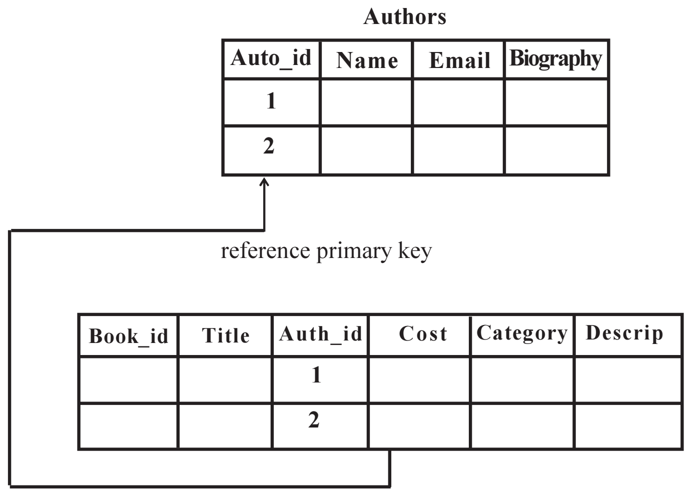


Figure 5 : Referential Integrity, or Parent/Child Relationships.

1.14 Relationship

DBMS relationships depend on the set-up and use of the DBMS. A relationship exists between two database tables when one table has a foreign key that references the primary key of another table.

Types of Database Relationships

There are three different types of database relationships, each named according to the number of table rows that may be involved in the relationship. Each of these three relationship types exists between two tables.

- One-to-one relationships occur when each entry in the first table has one, and only one, counter-part in the second table. One-to-one relationships are rarely used because it is often more efficient to simply put all of the information in a single table.

one record in a table is related to one record in a related table; creates equally dependent tables

Ex. one student has only one PSU ID

- One-to-many relationships are the most common type of database relationship. They occur when each record in the first table corresponds to one or more records in the second table but each record in the second table corresponds to only one record in the first table. For example, the relationship between a Teachers table and a Students table in an elementary school database would likely be a one-to-many relationship, because each student has only one teacher, but each teacher may have multiple students.
- Many-to-many relationships occur when each record in the first table corresponds to one or more records in the second table and each record in the second table corresponds to one or more records in the first table. For example, one student can be enrolled in many courses and each course can have many students enrolled

1.15 Database Management System Vs File Management System

A Database Management System (DMS) is a combination of computer software, hardware, and information designed to electronically manipulate data via computer processing. Two types of database management systems are DBMS's and FMS's. In simple terms, a File Management System (FMS) is a Database Management System that allows access to single files or tables at a time. FMS's accommodate flat files that have no relation to other files. The FMS was the predecessor for the Database Management System (DBMS), which allows access to multiple files or tables at a time.

File Management Systems

Advantages	Disadvantages
Simpler to use	Typically does not support multi-user access
Less expensive·	Limited to smaller databases
Fits the needs of many small businesses	
and home users	Limited functionality (i.e. no support for complicated transactions, recovery, etc.)

Popular FMS's are packaged along with the operating systems of personal computers (i.e. Microsoft Cardfile and Microsoft Works) Decentralization of data Good for database solutions for hand held devices such as Palm Pilot Redundancy and Integrity issues Typically, File Management Systems provide the following advantages and disadvantages:

The goals of a File Management System can be summarized as:

- Data Management. An FMS should provide data management services to the application.
- Generality with respect to storage devices. The FMS data abstractions and access methods should remain unchanged irrespective of the devices involved in data storage.
- Validity. An FMS should guarantee that at any given moment the stored data reflect the operations performed on them.
- Protection. Illegal or potentially dangerous operations on the data should be controlled by the FMS.
- Concurrency. In multiprogramming systems, concurrent access to the data should be allowed with minimal differences.
- Performance. Compromise data access speed and data transfer rate with functionality.

From the point of view of an end user (or application) an FMS typically provides the following functionalities:

- File creation, modification and deletion.
- Ownership of files and access control on the basis of ownership permissions.
- Facilities to structure data within files (predefined record formats, etc).
- Facilities for maintaining data redundancies against technical failure (back-ups, disk mirroring, etc.).
- Logical identification and structuring of the data, via file names and hierarchical directory structures.

1.16 Database Management Systems

Database Management Systems provide the following advantages and disadvantages:

Advantages	**Disadvantages**
Greater flexibility	Difficult to learn
Good for larger databases	Packaged separately from the operating system (i.e. Oracle, Microsoft Access, Lotus/IBM Approach, Borland Paradox, Claris FileMaker Pro)
Greater processing power	Slower processing speeds
Fits the needs of many medium to	Requires skilled administrators
large-sized organizations	
Storage for all relevant data	Expensive
Provides user views relevant to tasks	
performed	

Ensures data integrity by managing transactions (ACID test = atomicity, consistency, isolation, durability) Supports simultaneous access Enforces design criteria in relation to data format and structure Provides backup and recovery controls Advanced security	

The goals of a **Database Management System** can be summarized as follows:

- Data storage, retrieval, and update (while hiding the internal physical implementation details)
- A user-accessible catalog
- Transaction support
- Concurrency control services (multi-user update functionality)
- Recovery services (damaged database must be returned to a consistent state)
- Authorization services (security)
- Support for data communication Integrity services (i.e. constraints)
- Services to promote data independence
- Utility services (i.e. importing, monitoring, performance, record deletion, etc.)

The components to facilitate the goals of a DBMS may include the following:

- Query processor
- Data Manipulation Language preprocessor
- Database manager (software components to include authorization control, command processor, integrity checker, query optimizer, transaction manager, scheduler, recovery manager, and buffer manager)
- Data Definition Language compiler
- File manager
- Catalogue Manager

1.17 Advantages of DBMS

- The data independence and efficient access of data.
- It reduces application development time
- It provides data integrity and security
- Easy in data administration or data management
- Provides concurrent access, recovers the data from the crashesCentralized control

Disadvantages of DBMS:

- Problem Associate with centralizedCost of software, hardware and migration.
- Complexity of backup and recovery.

1.18 Features Commonly Offered by Database Management Systems Include

Query ability

Querying is the process of requesting attribute information from various perspectives and combinations of factors. Example: "How many 2-door cars in Texas are green?" A database query language and report writer allow users to interactively interrogate the database, analyze its data and update it according to the users privileges on data.

Backup and replication

Copies of attributes need to be made regularly in case primary disks or other equipment fails. A periodic copy of attributes may also be created for a distant organization that cannot readily access the original. DBMS usually provide utilities to facilitate the process of extracting and disseminating attribute sets. When data is replicated between database servers, so that the information remains consistent throughout the database system and users cannot tell or even know which server in the DBMS they are using, the system is said to exhibit replication transparency.

Rule enforcement

Often one wants to apply rules to attributes so that the attributes are clean and reliable. For example, we may have a rule that says each car can have only one engine associated with it (identified by Engine Number). If somebody tries to associate a second engine with a given car, we want the DBMS to deny such a request and display an error message. However, with changes in the model specification such as, in this example, hybrid gas-electric cars, rules may need to change. Ideally such rules should be able to be added and removed as needed without significant data layout redesign.

Security

For security reasons, it is desirable to limit who can see or change specific attributes or groups of attributes. This may be managed directly on an individual basis, or by the assignment of individuals and privileges to groups, or (in the most elaborate models) through the assignment of individuals and groups to roles which are then granted entitlements.

Computation

Common computations requested on attributes are counting, summing, averaging, sorting, grouping, cross-referencing, and so on. Rather than have each computer application implement these from scratch, they can rely on the DBMS to supply such calculations.

Change and access logging

This describes who accessed which attributes, what was changed, and when it was changed. Logging services allow this by keeping a record of access occurrences and changes.

Automated optimization

For frequently occurring usage patterns or requests, some DBMS can adjust themselves to improve the speed of those interactions. In some cases the DBMS will merely provide tools to monitor performance, allowing a human expert to make the necessary adjustments after reviewing the statistics collected.

1.19 Features of DBMS A Typical DBMS has the Following Features

- Provides a way to structure data as records, tables, or objects
- Accepts data input from operators and stores that data for later retrieval
- Provides query languages for searching, sorting, reporting, and other "decision support" activities that help users correlate and make sense of collected data
- Provides multiuser access to data, along with security features that prevent some users from viewing and/or changing certain types of information
- Provides data integrity features that prevent more than one user from accessing and changing the same information simultaneously

- Provides a data dictionary (metadata) that describes the structure of the database, related files, and record information

1.20 Summary

A **database** is an organized collection of data and the records, which is stored in a system, usually a computer. The organization of data is achieved by structuring it according to the model of the database.

A DBMS can take any one of the several approaches to manage data. Each approach constitutes a database model. A **data model** is a collection of descriptions of data structures and their contained fields, together with the operations or functions that manipulate them. A data model is a comprehensive scheme for describng how data is to be represented for manipulation by humans or computer programs.

DBA stands for **database administrator,** he is a person who is responsible for the features of a databse like databse recovery, integrity, availability, security and so on. DBA has to check these features in order to maintain the database.

A **key** is a field or combination of fields used to identify a record. When a key uniquely identifies a record it is referred to as the **primary key.**

1.21 Self-Assessment Questions

1. What do you understand by DBMS? Explain.
2. How file management system is different from DBMS? Explain.
3. List the advantages of DBMS.
4. What is Key? How primary key is different from foreign key?
5. Explain the concept of relationship with the help of an example.

′ ′ ′

Unit - 2 : Basic concepts of DBMS

Structure of the Unit

2.0 Objective

At the end of this unit, we should be abel to -

- Describe functions of database management system
- Describe the role of DBA, End users and application programmes
- Describe data independence
- Describe database schema and instance
- Describe DDL and DML commands

2.1 Introduction

A database management system (DBMS) is a software system that integrates data in storage and provides easy access to them. The data themselves are placed on disk in a database, which can be thought of as an integrated collection of related files. Although not all databases are organized identically, many of them are composed of files, records and fields. For instance, the product file contains five records one each for skis, boots, poles, bindings, and wax. Finally each record consists of distinct types of data called fileds. The product file stores four fileds for each record product name, product number, supplier, and price.

Database management softaware enables queries and reports to be prepared by extracting information from one file at a time, and, as we will shortly see, from several interrelated files concurrently. Database management system (DBMS) is a software product designed to integrate data and provide easy access to them. Database is an integrated collection of related data files and database management system is a computer programe for database management that links data in related files through common fileds. Database management system stores data in relevant form and give easy access to them. The data placed on disk themselves.

2.2 Purpose of Database Management System

The DBMS (Database Management System) is preferred ever the conventional file processing system due to the following advantages:

1. **Controlling Data Redundancy -** In the conventional file processing system, every user group maintains its own files for handling its data files. This may lead to:

- Duplication of same data in different files.
- Wastage of storage space, since duplicated data is stored.
- Errors may be generated due to updation of the same data in different files.
- Time in entering data again and again is wasted.
- Computer Resources are needlessly used.
- It is very difficult to combine information.

2. **Elimination of Inconsistency -** In the file processing system information is duplicated throughout the system. So changes made in one file may be necessary be carried over to another file. This may lead to inconsistent data. So we need to remove this duplication of data in multiple file to eliminate inconsistency.

For example: - Let us consider an example of student's result system. Suppose that in STUDENT file it is indicated that Roll no= 10 has opted for 'Computer'course but in RESULT file it is indicated that 'Roll No. =l 0' has opted for 'Accounts' course. Thus, in this case the two entries for z particular student don't agree with each other. Thus, database is said to be in an inconsistent state. Sc to eliminate this conflicting information we need to centralize the database. On centralizing the data base the duplication will be controlled and hence inconsistency will be removed.

Data inconsistency are often encountered in every day life Consider an another example, w have all come across situations when a new address is communicated to an organization that we deal it (Eg - Telecom, Gas Company, Bank). We find that some of the communications from that organization are received at a new address while other continued to be mailed to the old address. So combining all the data in database would involve reduction in redundancy as well as inconsistency so it is likely to reduce the costs for collection storage and updating of Data.

Let us again consider the example of Result system. Suppose that a student having Roll no -201 changes his course from 'Computer' to 'Arts'. The change is made in the SUBJECT file but not in RESULT'S file. This may lead to inconsistency of the data. So we need to centralize the database so that changes once made are reflected to all the tables where a particulars field is stored. Thus the update is brought automatically and is known as propagating updates.

3. **Better service to the users -** A DBMS is often used to provide better services to the users. In conventional system, availability of information is often poor, since it normally difficult to obtain information that the existing systems were not designed for. Once several conventional systems are combined to form one centralized database, the availability of information and its updateness is likely to improve since the

data can now be shared and DBMS makes it easy to respond to anticipated information requests.

Centralizing the data in the database also means that user can obtain new and combined information easily that would have been impossible to obtain otherwise. Also use of DBMS should allow users that don't know programming to interact with the data more easily, unlike file processing system where the programmer may need to write new programs to meet every new demand.

4. **Flexibility of the System is Improved -** Since changes are often necessary to the contents of the data stored in any system, these changes are made more easily in a centralized database than in a conventional system. Applications programs need not to be changed on changing the data in the database.

5. **Integrity can be improved -** Since data of the organization using database approach is centralized and would be used by a number of users at a time. It is essential to enforce integrity-constraints.

In the conventional systems because the data is duplicated in multiple files so updating or changes may sometimes lead to entry of incorrect data in some files where it exists.

For example : - The example of result system that we have already discussed. Since multiple files are to maintained, as sometimes you may enter a value for course which may not exist. Suppose course can have values (Computer, Accounts, Economics, and Arts) but we enter a value 'Hindi' for it, so this may lead to an inconsistent data, so lack of Integrity.

Even if we centralized the database it may still contain incorrect data. For example: -

- Salary of full time employ may be entered as Rs. 500 rather than Rs. 5000.
- A student may be shown to have borrowed books but has no enrollment.
- A list of employee numbers for a given department may include a number of non existent employees.

These problems can be avoided by defining the validation procedures whenever any update operation is attempted.

6. **Standards can be enforced -** Since all access to the database must be through DBMS, so standards are easier to enforce. Standards may relate to the naming of data, format of data, structure of the data etc. Standardizing stored data formats is usually desirable for the purpose of data interchange or migration between systems.

7. **Security can be improved -** In conventional systems, applications are developed in an adhoc/ temporary manner. Often different system of an organization would access different components of the operational data, in such an environment enforcing security can be quiet difficult. Setting up of a database makes it easier to enforce security restrictions since data is now centralized. It is easier to control who has access to what parts of the database. Different checks can be established for each type of access (retrieve, modify, delete etc.) to each piece of information in the database.

Consider an Example of banking in which the employee at different levels may be given access to different types of data in the database. A clerk may be given the authority to know only the names of all the customers who have a loan in bank but not the details of each loan the customer may have. It can be accomplished by giving the privileges to each employee.

8. **Organization's requirement can be identified -** All organizations have sections and departments and each of these units often consider the work of their unit as the most important and therefore consider their need as the most important. Once a database has been setup with centralized control, it will be necessary to identify organization's requirement and to balance the needs of the competating units. So it may become necessary to ignore some requests for information if they conflict with higher priority need of the organization.

It is the responsibility of the DBA (Database Administrator) to structure the database system to provide the overall service that is best for an organization.

For example: - A DBA must choose best file Structure and access method to give fast response for the high critical applications as compared to less critical applications.

9. **Overall cost of developing and maintaining systems is lower -** It is much easier to respond to unanticipated requests when data is centralized in a database than when it is stored in a conventional file system. Although the initial cost of setting up of a database can be large, one normal expects the overall cost of setting up of a database, developing and maintaining application programs to be far lower than for similar service using conventional systems, Since the productivity of programmers can be higher in using non-procedural languages that have been developed with DBMS than using procedural languages.

10. **Data Model must be developed -** Perhaps the most important advantage of setting up of database system is the requirement that an overall data model for an organization be build. In conventional systems, it is more likely that files will be designed as per need of particular applications demand. The overall view is often not considered. Building an overall view of an organization's data is usual cost effective in the long terms.

11. **Provides backup and Recovery -** Centralizing a database provides the schemes such as recovery and backups from the failures including disk crash, power failures, software errors which may help the database to recover from the inconsistent state to the state that existed prior to the occurrence of the failure, though methods are very complex.

2.3 Functions of DBMS

1. DBMS free the programmers from the need to worry about the organization and location of the data i.e. it shields the users from complex hardware level details.

2. DBMS can organize process and present data elements from the database. This capability enables decision makers to search and query database contents in order to extract answers that are not available in regular Reports.

3. Programming is speeded up because programmer can concentrate on logic of the application.

4. It includes special user friendly query languages which are easy to understand by non programming users of the system.

The various common examples of DBMS are Oracle, Access, SQL Server, Sybase, FoxPro, Dbase etc.

2.4 The Service Provided by the DBMS Includes

1. Authorization services like log on to the DBMS, start the database, stop the Database etc.
2. Transaction supports like Recovery, Rollback etc,
3. Import and Export of Data.
4. Maintaining data dictionary
5. User's Monitoring

2.5 DBA, Database Designers, End Users & Application Programmers

2.5.1 Database Administrator (DBA)

The DBA is a person or a group of persons who is responsible for the management of the database. The DBA is responsible for authorizing access to the database by grant and revoke permissions to the users, for coordinating and monitoring its use, managing backups and repairing damage due to hardware and/or software failures and for acquiring hardware and software resources as needed. In case of small organization the role of DBA is performed by a single person and in case of large organizations there is a group of DBA's who share responsibilities.

2.5.2 Database Designers

They are responsible for identifying the data to be stored in the database and for choosing appropriate structure to represent and store the data. It is the responsibility of database designers to communicate with all prospective of the database users in order to understand their requirements so that they can create a design that meets their requirements.

2.5.3 End Users

End Users are the people who interact with the database through applications or utilities. The various categories of end users are:

2.5.3.1 Casual End Users : These Users occasionally access the database but may need different information each time. They use sophisticated database Query language to specify their requests. For example: High level Managers who access the data weekly or biweekly.

2.5.3.2 NativeEnd Users : These users frequently query and update the database using standard types of Queries. The operations that can be performed by this class of users are very limited and effect precise portion of the database.

For example: - Reservation clerks for airlines/hotels check availability for given request and make reservations. Also, persons using Automated Teller Machines (ATM's) fall under this category as he has access to limited portion of the database.

2.5.3.3 Standalone end Users/On-line End Users : Those end Users who interact with the database directly via on-line terminal or indirectly through Menu or graphics based Interfaces.

For example: - User of a text package, library management software that store variety of library data such as issue and return of books for fine purposes.

2.5.4 Application Programmers

Application Programmers are responsible for writing application programs that use the database. These programs could be written in General Purpose Programming languages such as Visual Basic, Developer, C, FORTRAN, COBOL etc. to manipulate the database. These application programs operate on the data to perform various operations such as retaining information, creating new information, deleting or changing existing information.

- DBMS engine accepts logical requests from various other DBMS subsystems, converts them into physical equivalents, and actually accesses the database and data dictionary as they exist on a storage device.
- Data definition subsystem helps the user create and maintain the data dictionary and define the structure of the files in a database.
- Data manipulation subsystem helps the user to add, change, and delete information in a database and query it for valuable information. Software tools within the data manipulation subsystem are most often the primary interface between user and the information contained in a database. It allows the user to specify its logical information requirements.
- Application generation subsystem contains facilities to help users develop transaction-intensive applications. It usually requires that the user perform a detailed series of tasks to process a transaction. It facilitates easy-to-use data entry screens, programming languages, and interfaces.
- Data administration subsystem helps users manage the overall database environment by providing facilities for backup and recovery, security management, query optimization, concurrency control, and change management.

2.6 DBMS-Architecture and Data Independence

Database management systems are complex softwares which were often developed and optimised over years. From the view of the user, however, most of them have a quite similar basic architecture.

Three-Schemes Architecture

Knowing about the conceptual and the derived logical scheme (discussed in unit Database Models, Schemes and Instances this unit explains two additional schemes - the external scheme and the internal scheme - which help to understand the DBMS architecture.

External Scheme:

An external data scheme describes the information about the user view of specific users (single users and user groups) and the specific methods and constraints connected with this information. (Translated) (ZEHNDER 1998)

Internal Scheme:

The internal data scheme describes the content of the data and the required service functionality which is used for the operation of the DBMS. (Translated) (ZEHNDER 1998)

Therefore, the internal scheme describes the data from a view very close to the computer or system in general. It completes the logical scheme with data technical aspects like storage methods or help functions for more efficiency.

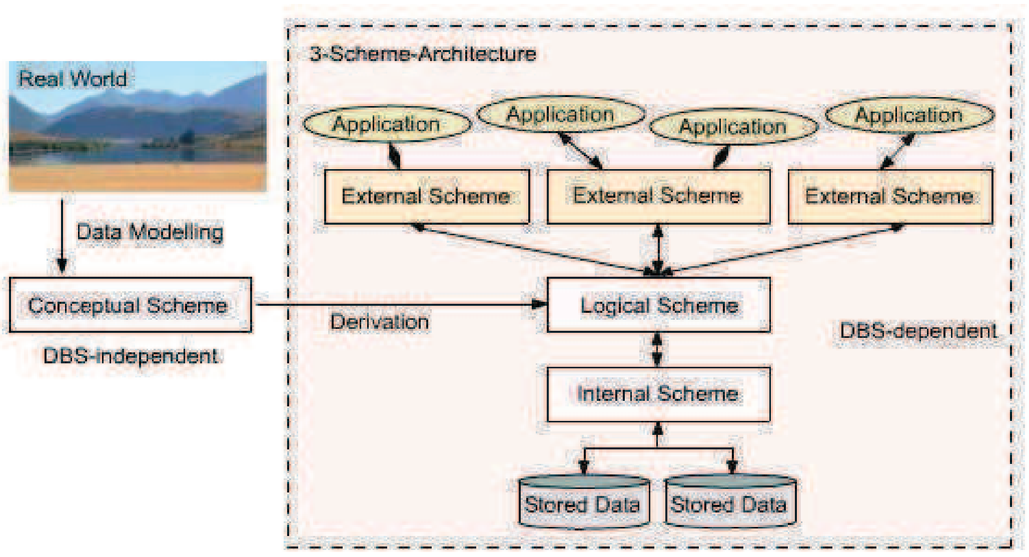


Figure 2.1 : Three-Schemes Architecture

The right hand side of the representation above is also called the three-schemes architecture: internal, logical and external scheme.

While the internal scheme describes the physical grouping of the data and the use of the storage space, the logical scheme (derived from the conceptual scheme) describes the basic construction of the data structure. The external scheme of a specific application, generally, only highlights that part of the logical scheme which is relevant for its application. Therefore, a database has exactly one internal and one logical scheme but may have several external schemes for several applications using this database.

The aim of the three-schemes architecture is the separation of the user applications from the physical database, the stored data. Physically the data is only existent on the internal level while other forms of representation are calculated or derived respectively if needed. The DBMS has the task to realise this representation between each of these levels.

2.7 Data Independence

With knowledge about the three-schemes architecture the term data independence can be explained as followed: Each higher level of the data architecture is immune to changes of the next lower level of the architecture.

2.7.1 Physical Independence

Therefore, the logical scheme may stay unchanged even though the storage space or type of some data is changed for reasons of optimisation or reorganisation.

2.7.2 Logical Independence

Also the external scheme may stay unchanged for most changes of the logical scheme. This is especially desirable as in this case the application software does not need to be modified or newly translated.

2.8 Database Schema

Definition : Overall design of data base. Schema contains 'No of records + Type of data + No of attributes'

1. External level or Sub schema
2. logical schema
3. Physical schema

2.9 Database Instance

The term instance is typically used to describe a complete database environment, including the RDBMS software, table structure, stored procedures and other functionality. It is most commonly used when administrators describe multiple instances of the same database.

The information stored in database at the particular movement is called instance.

Also Known As: environment

Examples: An organization with an employees database might have three different instances: production (used to contain live data), pre-production (used to test new functionality prior to release into production) and development (used by database developers to create new functionality).

There are two different types of languages to make database system. They are 1. To specify the database schema, and 2.to express database queries and updates.

2.10 Data-Definition Language

A database schema is specified by a set of definitions expressed by special language called a data-definition language (DDL). The result of compilation of DDL statements is a set of tables that is stored in a special file called Data dictionary or data directory.

A data dictionary is a file that contains metadata that is, data about data. This file is consulted before actual data are read or modified in the database system. The storage structure and access methods used by the database system are specified by a set of definitions in a special type of DDL called a data storage and definition language.

2.11 Data-Manipulation Language

The levels of abstraction apply not only to the definition or structuring of data, but also the manipulation of data. By data manipulation, it means

- The retrieval of information stored in the database
- The insertion of new information into the database
- The deletion of information from the database

- The modification of information stored in the database

A data-manipulation language (DML) is a language that enables user to access or manipulate data as organized by the appropriate data model. They are basically two types.

- Procedural DMLs require a user to specify what data are needed and how to get those data
- Nonprocedural DMLs require a user to specify what data are needed without specifying how to get those data

Non procedural DMLs are usually easier to learn and use than are procedural DMLs. A query is a statement requesting the retrieval of information. The portion of a DML that involves information retrieval is called a query language.

2.12 Summary

A **database management system (DBMS)** is a software system that integrates data in storage and provides easy access to them. The data themselves are placed on disk in a database, which can be thought of as an integrated collection of related files. Although not all databases are organized identically, many of them are composed of files, records and fields.

2.13 Self-Assessment Questions

1. Explain the different function of DBMS.
2. Explain the role of DBA.
3. Explain the role of different users of DBMS.
4. What is Data Independence? Explain.
5. Explain the concept of DDL and DML commands in DBMS.

′ ′ ′

Unit - 3 : Database Models

Structure of the Unit

3.0 Objective

In this unit we will discuss the fundamental characteristics of the database approach is that it provides some level of Underlying the structure of a database is the data model: a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints. This unit provides a general overview of the nature and purpose of database systems. We explain the how the concept of a database system has developed, what the common features of database systems are, what a database system does for the user, and how a database system interfaces with operating systems.

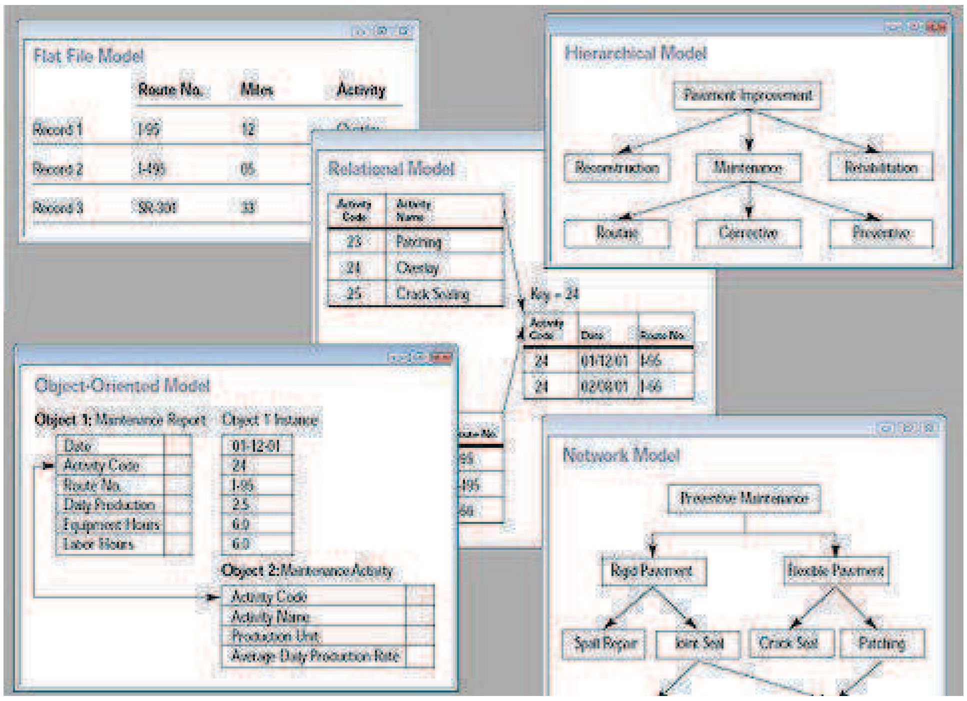
3.1 Introduction

A **database model** s a collection of concepts that can be used to describe the structure of a database. By structure of a database we mean the data types, relationships and constraints that should hold on the data. A **database model** is the theoretical foundation of a database and fundamentally determines in which manner data can be stored, organized, and manipulated in a database system. It thereby defines the infrastructure offered by a particular database system. The most popular example of a database model is the relational model.

Data modeling is a method used to define and analyze data requirements needed to support the business processes of an organization and by defining the data structures and the relationships between data elements.

Data modeling techniques are used :

- to manage data as a resource (migrate, merge, data quality…)
- for designing computer databases.
- to better cope with change, by allowing to make changes into the model, that will automatically induce changes in the database and programs



3.2 Types of Database Models

Data models representations usually graphical of complex real-world data structures it facilitate interaction among the designer, the applications programmer and the end user. End-users must have different views and needs for data. Data model organizes data for various users.

Data models can be broadly distinguished into 3 main categories-

1. High-level or conceptual data models (based on entities & relationships) :

It provides concepts that are close to the way many users perceive data. Conceptual data models use concepts such as entities, attributes and relationships .An entity represents a real world object or concepts, such as employee or a project that is described in the database. An attribute represents some property of interest that further describes an entity, such as the employee's name or salary. A relationship among two or more entities represents an interaction among the entities

3.2.1 The conceptual model describes the system information, as represented by a particular data manipulation technology type : e.g. flat file system, hierarchical DBMS(IMS,...), network DBMS (IDMS, IDS2,..), relational DBMS (DB2, ORACLE, SQL SERVER,...). A logical model consists of descriptions of entities (called « segments » in hierarchical DBMS, « records » in network DBMS, « tables » in relational DBMS) and attributes (called « data » in hierarchical and network DBMS, « columns » in relational DBMS), data access keys, type of links beetween entities (called « sets » in network DBMS, « foreigns keys » in relational DBMS) among other things;

2. Low level or physical data models :

It provides concepts that describe the details of how data is stored in the computer.These concepts are meant for computer specialist, not for typical end users. **The physical model** describes the physical means by which data are stored in a particular DBMS product (flat files, XML files, IMS, IDS2, IDMS, ORACLE, DB2, ...) . This is concerned with partitions, CPUs, tablespaces, and the like.

3. Representational or implementation data models (record-based, object-oriented) :

It provide concepts that can be understood by end users. These hide some details of data storage but can be implemented on a computer system directly.

3.2.2 The representational model

Describes the semantics of a domain (for example, it may be a model of the interest area of an organization or industry), i.e. define the meaning of data within the context of its interrelationships and constraints with other data. It is an abstraction which defines how the stored symbols relate to the real world. Thus, the semantic model must be a true representation of the real world. A semantic model consists of entity classes, representing kinds of things of significance in the domain, and relationships assertions about associations between pairs of entity classes. A semantic model specifies the kinds of facts or propositions that can be expressed using the model. In that sense, it defines the allowed expressions in an artificial 'language' with a scope that is limited by the scope of the model;

A data model is a collection of concepts for describing data, its relationships, and its constraints provides a clearer and more accurate description and representation of data Standard platform that enables database designers and end-users to communicate Come in three varieties:

3.2.3 Object Based Models or Object Oriented Model (conceptual schema)

Theses models are used in describing data at the logical and view levels. They are characterized by the fact that they provide fairly flexible structuring capabilities and allow data constraints to be specified explicitly. They are many different models, and more are likely to come. Several of the more widely known ones are:

- The entity relationship model
- The object-oriented model
- The semantic data model
- The functional data model

The entity-relationship(E-R) data model is based on a perception of a real world that consists of a collection of basic concepts, called entities, and of relationships among these objects. An entity is a "thing" or "object" in the real world that is distinguishable from other objects. A relationship is an association among several entities. In addition to entities and relationships, the E-R model represents certain constraints to which the contents of a database must conform. One important constraint is mapping cardinalities, which express the number of entities to which another entity can be associated via a relationship set.

The overall logical structure of a database can be expressed graphically by an E-R diagram, which is built up from the following components:

- **Rectangles**, which represent entity sets
- **Ellipses**, which represent attributes
- **Diamonds**, which represent relationships among entity sets
- **Lines**, which link attributes to entity sets and entity sets to relationships

Each component is labeled with the entity or relationship that it represents.

Like the E-R model , the object-oriented model is based on a collection of a objects. An object contains values stored in instance variables within the objects. An object also contains bodies of code that operate on the objects. These bodies of code are called methods.

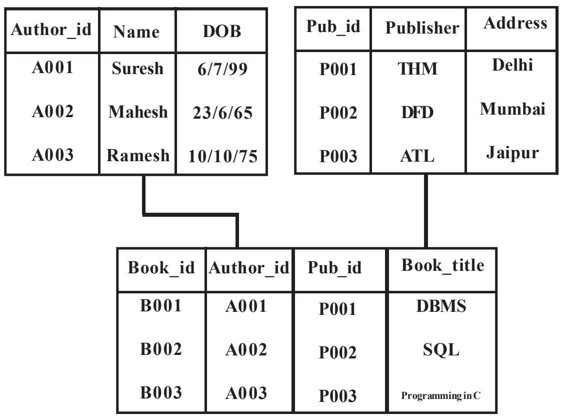
Objects that contain the same types of values and the same methods are grouped together into classes. A class may be viewed as a type definition for objects. This combination of data and methods comprising a type definition is similar to a programming-language abstract data type.

3.2.4 Record-based models (external schema)

Theses models are used in describing data at the logical and view levels. In contrast to object-oriented data models, they are used both to specify the overall logical structure of the database and to provide a higher-level description of the implementation. Record-based models are so named because the database is structured in fixed-format records of several types. Each record type defines a fixed number of fields, or attributes, and each field is usually of a fixed length.

A record based data model is used to specify the overall logical structure of the database. In this model the database consists of a no. of fixed formats of different types. Each record type defines a fixed no. of fields having a fixed length. There are 3 principle types of record based data model. They are:

- **Relational model**
- **Network model**
- **Hierarchical model**



Author_id	Name	DOB
A001	Suresh	6/7/99
A002	Mahesh	23/6/65
A003	Ramesh	10/10/75

Pub_id	Publisher	Address
P001	THM	Delhi
P002	DFD	Mumbai
P003	ATL	Jaipur

Book_id	Author_id	Pub_id	Book_title
B001	A001	P001	DBMS
B002	A002	P002	SQL
B003	A003	P003	Programming in C

The relational model uses a collection of tables to represent both data and the relationships among those data. Each table has multiple columns, and each column has a unique name.

Data in the network model are represented by collections of records and the relationships among data are represented by links , which can be viewed as pointers. The records in the database are organized as collections of arbitrary graphs.

The hierarchical model is similar to the network model in the sense that data and relationships among data are represented by records and links, respectively. It differs from the network model in the that the network in that the records are organized as collections of trees rather than arbitrary graphs .

3.2.5 Physical data models (internal schema)

Physical data models are used to describe data at the lowest level. In contrast to logical data models, there are few physical data models in use. Two of the widely known ones are the unifying model and the frame-memory model.

3.3 Introduction to Hierarchical Model

In a hierarchical model , data is organized into a tree-like structure, implying a single upward link in each record to describe the nesting, and a sort field to keep the records in a particular order in each same-level list. Hierarchical structures were widely used in the early mainframe database management systems, such as the information Management System (IMS) by IBM, and now describe the structure of XML documents. This structure allows one 1:M relationship between two types of data. This structure is very efficient to describe many relationships in the real world; recipes, table of contents, ordering of paragraphs/verses, any nested and sorted information. However, the hierarchical structure is inefficient for certain database operations when a full path (as opposed to upward link and sort field) is not also included for each record.

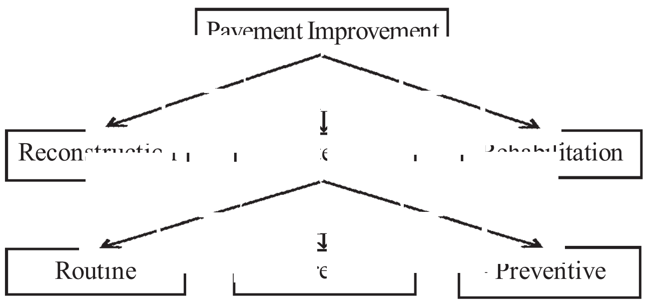
Mother–child relationship: Child may only have one mother but a mother can have multiple children. Mothers and children are tied together by links called "pointers". A mother will have a list of pointers to each of her children.

Hierarchical DBMSs were popular from the late 1960s, with the introduction of IBM's Information Management System (IMS) DBMS, through the 1970s.

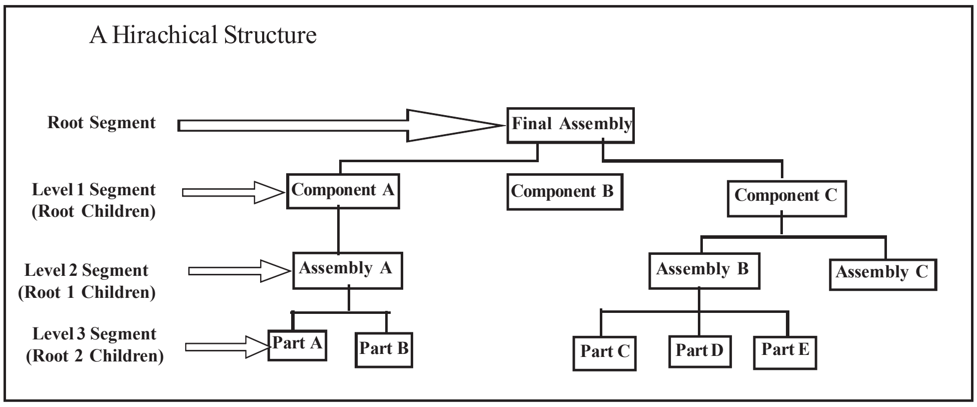
The hierarchical data model organizes data in a tree structure. There is a hierarchy of parent and child data segments. There is a hierarchy of parent and child data segments. This structure implies that a record can have repeating information, generally in the child data segments. Data in a series of records, which have a set of field values attached to it. It collects all the instances of a specific record together as a record type. These record types are the equivalent of tables in the relational model, and with the individual records being the equivalent of rows. To create links between these record types, the hierarchical model uses Parent Child Relationships.

These are a 1 : N mapping between record types. This is done by using trees, like set theory used in the relational model, "borrowed" from maths. For example, an organization might store information about an employee, such as name, employee number, department, salary. The organization might also store information about an employee's children, such as name and date of birth. The employee and children data forms a hierarchy, where the employee data represents the parent segment and the children data represents the child segment. If an employee has three children, then there would be three child segments associated with one employee segment. In a hierarchical database the parent-child relationship is one to many. This restricts a child segment to having only one parent_segment.

Hierarchical Model



Data in a series of records, which have a set of field values attached to it.



Basic Features of Hierarchical Model :

(a) Logical structure represented as an upside-down "tree"

(b) Hierarchical structure contains levels or segments

(c) Depicts a set of one-to-many (1: M) relationships Between a parent and it's children segments

(d) Each parent can have many children

(e) Each child has only one parent

Advantages & Disadvantages of Hierarchical model

Advantages :

(a) Many features form the foundation for current data models

(b) Generated a large installed base of programmers Who developed solid business applications

Disadvantages :

(a) Complex to implement

(b) Difficult to manage

(c) Lacks structural independence

(d) Implementation limitations

(e) Lack of standards (Company vs. Industry or Open)

3.4 Introduction to Network Data Model

The popularity of the network data model coincided with the popularity of the hierarchical data model. Some data were more naturally modeled with more than one parent per child. So, the network model permitted the modeling of many-to-many relationships in data. In 1971, the Conference on Data Systems Languages (CODASYL) formally defined the network model. The basic data modeling construct in the network model is the set construct. A set consists of an owner record type, a set name, and a member record type. A member record type can have that role in more than one set; hence the multiparent concept is supported. An owner record type can also be a member or owner in another set.

The data model is a simple network, and link and intersection record types (called junction records by IDMS) may exist, as well as sets between them. Thus, the complete network of relationships is represented by several pairwise sets; in each set some (one) record type is owner (at the tail of the network arrow) and one or more record types are members (at the head of the relationship arrow). Usually, a set defines a 1: M relationship, although 1:1 is permitted. The CODASYL network model is based on mathematical set theory. In the network data model, the database consists of a collection of set-type occurrences.

The network model (defined by the CODASYL specification) organizes data using two fundamental constructs, called *records* and *sets*. Records contain fields (which may be organized hierarchically, as in the programming language COBOL). Sets (not to be confused with mathematical sets) define one-to-many relationships between records: one owner, many members. A record may be an owner in any number of sets, and a member in any number of sets.

The network model is a variation on the hierarchical model, to the extent that it is built on the concept of multiple branches (lower-level structures) emanating from one or more nodes (higher-level structures), while the model differs from the hierarchical model in that branches can be connected to multiple nodes. The network model is able to represent redundancy in data more efficiently than in the hierarchical model.

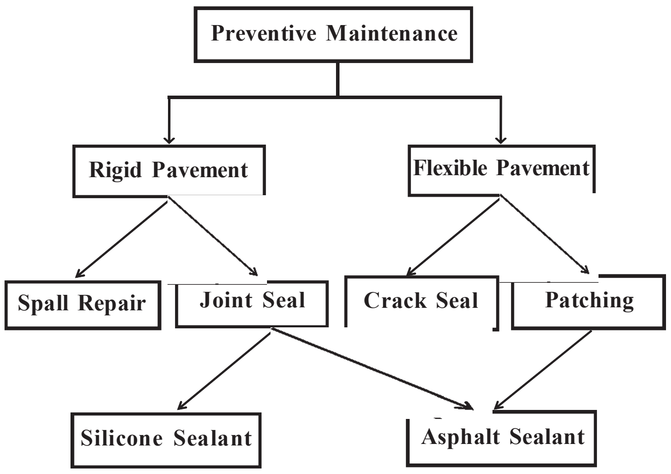
The operations of the network model are navigational in style: a program maintains a current position, and navigates from one record to another by following the relationships in which the record participates. Records can also be located by supplying key values.

Although it is not an essential feature of the model, network databases generally implement the set relationships by means of pointers that directly address the location of a record on disk. This gives excellent retrieval performance, at the expense of operations such as database loading and reorganization.

Most object database use the navigational concept to provide fast navigation across networks of objects, generally using object identifiers as “smart” pointers to related objects. Objectivity/DB, for instance, implements named 1:1, 1:many, many:1 and many: many named relationships that can cross databases. Many object databases also support SQL, combining the strengths of both models.

Each set-type occurrence has one occurrence of OWNER RECORD, with zero or more occurrences of MEMBER RECORDS.

Network Model



Basic features of Network Model :

- Resembles hierarchical model
- Difference child can have multiple parents
- Collection of records in 1: M relationships
- Set – Relationship of at least two record types
  - • Owner – Equivalent to the hierarchical model's parent
  - • Member – Equivalent to the hierarchical model's child
- In the network data model, the database consists of a collection of set-type occurrences.
- Each set-type occurrence has one occurrence of OWNER RECORD, with zero or more occurrences of MEMBER RECORDS.

  -> The member sets belonging to different owners are disjoint.

  -> To define a network database one needs to define:

  (a) the database record types which consist of data items, and

  (b) the set-types.

The member sets belonging to different owners are disjoint.To define a network database one needs to define:

(a) The database record types which consist of data items, and

(b) The set-types.

3.5 Introduction to Relational Model

The relational model uses a collection of tables to represent both data and the relationships among those data. Each table has multiple columns, each column has a unique name. (RDBMS - relational database management system) A database based on the relational model developed by E.F. Codd. A relational database allows the definition of data structures, storage and retrieval operations and integrity constraints. In such a database the data and relations between them are organised in tables. A table is a collection of records and each record in a table contains the same fields.

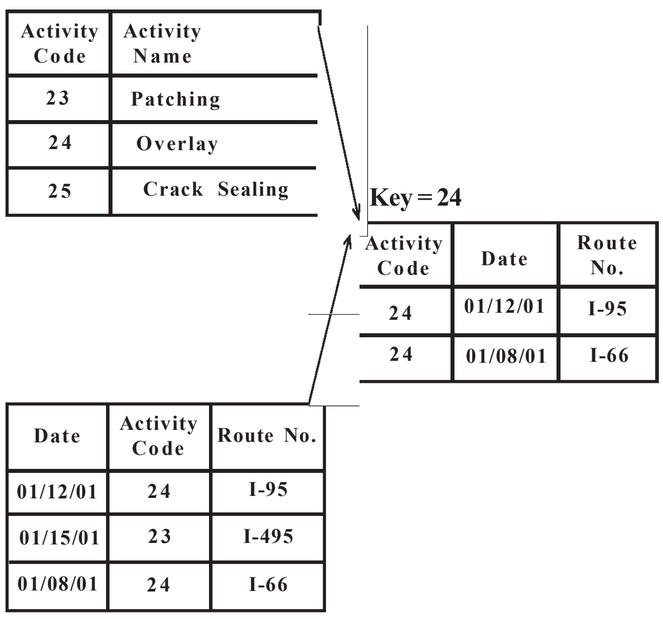
Properties of Relational Tables:

- Values Are Atomic
- Each Row is Unique
- Column Values Are of the Same Kind
- The Sequence of Columns is Insignificant
- The Sequence of Rows is Insignificant
- Each Column Has a Unique Name

Certain fields may be designated as keys, which means that searches for specific values of that field will use indexing to speed them up. Where fields in two different tables take values from the same set, a join operation can be performed to select related records in the two tables by matching values in those fields. Often, but not always, the fields will have the same name in both tables. For example, an "orders" table might contain (customer-ID, product-code) pairs and a "products" table might contain (product-code, price) pairs so to calculate a given customer's bill you would sum the prices of all products ordered by that customer by joining on the product-code fields of the two tables.

This can be extended to joining multiple tables on multiple fields. Because these relationships are only specified at retreival time, relational databases are classed as dynamic database management system. The RELATIONAL database model is based on the Relational Algebra.

Relational Model



Activity Code	Activity Name
23	Patching
24	Overlay
25	Crack Sealing

Key = 24

Activity Code	Date	Route No.
24	01/12/01	I-95
24	01/08/01	I-66

Date	Activity Code	Route No.
01/12/01	24	I-95
01/15/01	23	I-495
01/08/01	24	I-66

The products that are generally referred to as relational database in fact implement a model that is only an approximation to the mathematical model defined by Codd. Three key terms are used extensively in relational database models: relations , *attributes*, and domains. A relation is a table with columns and rows. The named columns of the relation are called attributes, and the domain is the set of values the attributes are allowed to take.

The basic data structure of the relational model is the table, where information about a particular entity (say, an employee) is represented in rows (also called tuples) and columns. Thus, the " relation" in "relational database" refers to the various tables in the database; a relation is a set of tuples. The columns enumerate the various attributes of the entity (the employee's name, address or phone number, for example), and a row is an actual instance of the entity (a specific employee) that is represented by the relation. As a result, each tuple of the employee table represents various attributes of a single employee.

All relations (and, thus, tables) in a relational database have to adhere to some basic rules to qualify as relations. First, the ordering of columns is immaterial in a table. Second, there can't be identical tuples or rows in a table. And third, each tuple will contain a single value for each of its attributes.

A relational database contains multiple tables, each similar to the one in the "flat" database model. One of the strengths of the relational model is that, in principle, any value occurring in two different records (belonging to the same table or to different tables), implies a relationship among those two records. Yet, in order to enforce explicit integrity constraints, relationships between records in tables can also be defined explicitly, by identifying or non-identifying parent-child relationships characterized by assigning cardinality (1:1, (0)1:M, M:M). Tables can also have a designated single attribute or a set of attributes that can act as a "key", which can be used to uniquely identify each tuple in the table.

A key that can be used to uniquely identify a row in a table is called a primary key. Keys are commonly used to join or combine data from two or more tables. For example, an *Employee* table may contain a column named *Location* which contains a value that matches the key of a *Location* table. Keys are also critical in the creation of indexes, which facilitate fast retrieval of data from large tables. Any column can be a key, or multiple columns can be grouped together into a compound key. It is not necessary to define all the keys in advance; a column can be used as a key even if it was not originally intended to be one.

A key that has an external, real-world meaning (such as a person's name, a book's ISBN, or a car's serial number) is sometimes called a "natural" key. If no natural key is suitable (think of the many people named *Brown*), an arbitrary or surrogate key can be assigned (such as by giving employees ID numbers). In practice, most databases have both generated and natural keys, because generated keys can be used internally to create links between rows that cannot break, while natural keys can be used, less reliably, for searches and for integration with other databases. (For example, records in two independently developed databases could be matched up by social security number, except when the social security numbers are incorrect, missing, or have changed.)

3.6 Summary

A **data model** is a collection of concepts for describing data, its relationships, and its constraints provides a clearer and more accurate description and representation of data.

Data models can be broadly distinguished into 3 main categories- High-level or conceptual data models, Low level or physical data models, Representational or implementation data models.

In a **hierarchical model** , data is organized into a tree-like structure, implying a single upward link in each record to describe the nesting, and a sort field to keep the records in a particular order in each same-level list.

The **network model** (defined by the CODASYL specification) organizes data using two fundamental constructs, called *records* and *sets*. Records contain fields (which may be organized hierarchically, as in the programming language COBOL).

All relations (and, thus, tables) in a **relational model** have to adhere to some basic rules to qualify as relation.

3.7 Self Assessment Questions

1. Explain why we study the data models in databases.
2. What is data model and write down the basic characteristics of the databases?
3. What is the difference between E-R model and Relational model?
4. What is the difference between network model and hierarchical model?
5. How can you define the relational model and its characteristics?

Unit - 4 : The Entity Relationship Models

Structure of the Unit

4.0 Objective

In this unit, we will follow the traditional approach for concentrating on the database structures and constraints during database design. We will present the modeling concepts of the **Entity-Relationship model,** which is a popular high-level conceptual data model. This model and its variations are frequently used for the conceptual design of database applications, and many databases design tools employ its concepts. We describe the basic data structuring concepts and constraints of the E-R model and discuss their use in the design of conceptual schemas for database applications. The entity relationship model(E-R) data model is based on a perception of a real world that consists of a set of basic objects called entities, and of relationships among these objects.

4.1 Introduction

In 1976, Chen developed the **Entity-Relationship (ER) model**, a high-level data model that is useful in developing a conceptual design for a database. Creation of an ER diagram, which is one of the first steps in designing a database, helps the designer(s) to understand and to specify the desired components of the database and the relationships among those components. An ER model is a diagram containing entities or "items", relationships among them, and attributes of the entities and the relationships.

We can consider the database on three levels of abstraction: external, conceptual, and internal.

- The **external level** has the users' views of the database. Depending on their needs, different users access different parts of the database. For example, a doctor performing drug tests should be able to access the patients' medical data but not their hospital bills. However, a billing clerk should have a very different view of the database.
- The **conceptual level** describes the logical structure of an entire database, including descriptions of the data and relationships among the data. For example, at this level we would describe a row of the table *StudyA05* as containing the values for *SSN* and *placebo*. However, we would not give the details of the physical storage of the fields and records.
- The **internal level** gives the details of the physical storage of the database on the computer. This level contains such details as the number of bytes for each data item, ordering of records, and data compression techniques. For example, at this level we would describe the attribute *SSN* as 10 bytes to store the nine-digit social security number.

For example: Three-Level Architecture

External Level

Doctor's View   Billing Office View

SSN   *dosage*   *SSN*   *LastName*   *FirstName*   …

Conceptual Level

SSN   *Dosage*   *LastName*   *FirstName*   …

Internal Level

```
struct {    string SSN;    double dosage;    string LastName;    string FirstName;    ...    }
```

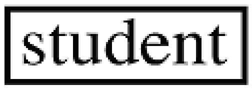
E-R model

To make the description of the model more complete, we consider the example of a physics department at a college that maintains a database of experimental results. Throughout a laboratory, students collaborate and share their results and access data sets from other semesters on a computer system. For example, in the laboratory session on "Freely Falling Objects with Significant Drag," students determine the drag coefficient by dropping dust balls from different heights and measuring the times they take to fall. Each team enters its results into the distributed database, and the class analyzes the data. After a team enters data into the web-accessed database, all students can obtain the measurements simultaneously. To simplify the analysis, we assume that the database only stores results related to this experiment over a period of several years.

4.2 Entity

An **Entity** is a real-world item or concept that exists on its own. In our example, a particular student (such as, "Emanuel Vagas"), team, lab section, or experiment is an entity. The set of all possible values for an entity, such as all possible students, is the entity type. In an ER model, we diagram an entity type as a rectangle containing the type name, such as student .

For example: E-R diagram notation for entity student



An ***entity*** is an object that exists and which is distinguishable from other objects. An entity can be a person, a place, an object, an event, or a concept about which an organization wishes to maintain data. The following are some examples of entities:

Person: STUDENT, EMPLOYEE, CLIENT

Object: COUCH, AIRPLANE, MACHINE

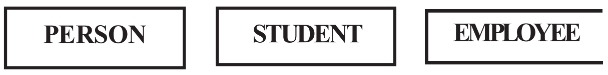
Place: CITY, NATIONAL PARK, ROOM, WAREHOUSE

Event: WAR, MARRIAGE, LEASE

Concept: PROJECT, ACCOUNT, COURSE

It is important to understand the distinction between an *entity type*, an *entity instance*, and an *entity set*. An ***entity type*** defines a collection of entities that have same attributes. An ***entity instance*** is a single item in this collection. An ***entity set*** is a set of entity instances. The following example will clarify this distinction: STUDENT is an entity type; a student with ID number 555-55-5555 is an entity instance; and a collection of all students is an entity set.

In the E-R diagram, we assign a name to each entity type. When assigning names to entity types, we follow certain naming conventions. An entity name should be a concise singular noun that captures the unique characteristics of the entity type. An E-R diagram depicts an entity type using a rectangle with the name of the entity inside .

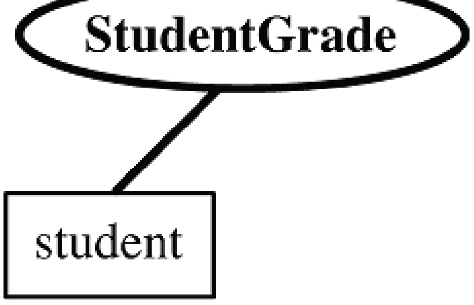


4.3 Attribute

Each entity has attributes, or particular properties that describe the entity. Attributes are descriptive properties possessed by each member of an entity set. The designation of an attribute for an entity set expresses that the databases stores similar information concerning each entity in the entity set, each entity may have its own value for each attribute. For each attribute there is a set of permitted values, called the domain , or value set, of that attribute.

For example, student Emanuel Vagas has properties of his own Student Identification number, name, and grade. A particular value of an attribute, such as 93 for the grade, is a value of the attribute. Most of the data in a database consists of values of attributes. The set of all possible values of an attribute, such as integers from 0 to 100 for a grade, is the attribute domain. In an ER model, an attribute name appears in an oval that has a line to the corresponding entity box.

ER diagram notation for an attribute domain (*StudentGrade*) of an entity type (*student*)



We represent an entity with a set of attributes. An ***attribute*** is a property or characteristic of an entity type that is of interest to an organization. Some attributes of common entity types include the following:

STUDENT = {Student ID, SSN, Name, Address, Phone, Email, DOB}

ORDER = {Order ID, Date of Order, Amount of Order}

ACCOUNT = {Account Number, Account Type, Date Opened, Balance}

CITY = {City Name, State, Population}

We use the following conventions while naming attributes:

1. Each word in a name starts with an uppercase letter followed by lower case letters.
2. If an attribute name contains two or more words, the first letter of each subsequent word is also in uppercase, unless it is an article or preposition, such as "a," "the," "of," or "about".

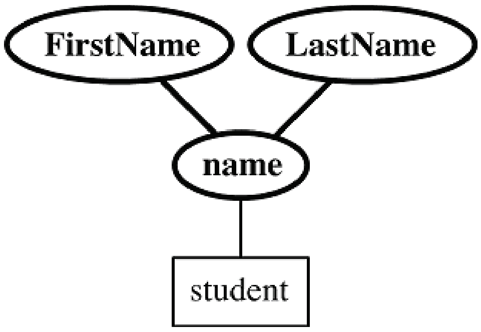
Types of Attributes

Several types of attributes our in the E-R model. Which are as follows :

(a) Simple and Composite Attributes :

An attribute can be simple or composite. A simple attribute is one component that is atomic. They are not divided into subparts . A simple attribute, such as grade, is one component that is atomic. A composite attribute, on the other hand, can be divided into subparts. it has multiple components, each of which is atomic or composite. If we consider the name in two parts, last name and first name, then the name attribute is a composite. A composite attribute, such as "Emanuel Vagas", has multiple components, such as "Emanuel" and "Vagas"; and each component is atomic or composite. We illustrate this composite nature in the ER model by branching off the component attributesUsing composite attributes in a design schema is a good choice if a user will wish to refer to an entire attribute on some occasions, and to only a component of the attribute on other occasions.

For example, ER diagram notation for composite attribute domain, *name*

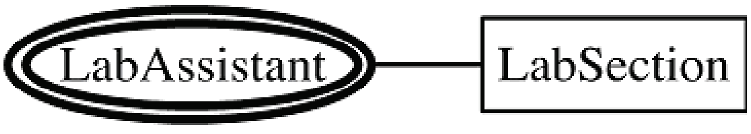


(b) Single Valued and Multi-Valued Attributes :

For a particular entity, an entity attribute that holds exactly one value is a **single-valued attribute**. A **multi-valued attribute** has more than one value for a particular entity.

Another way to classify attributes is either as single-valued or multi-valued. For an entity an attribute, such as *StudentGrade*, usually holds exactly one value, such as 93, and thus is a **single-valued attribute**. However, two lab assistants might assist in a laboratory section. Consequently, the *LabAssistant* attribute for the entity *LabSection* is multi-valued. A **multi-valued attribute** has more than one value for a particular entity. We illustrate this situation with a double oval around the lab assistant type, *LabAssistant* .

For example: ER diagram notation for multi-valued attribute domain, *LabAssistant*

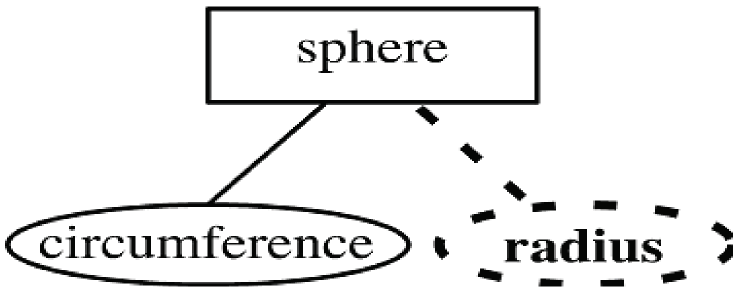


(c) Stored and Derived attributes :

The value of a ***derived attribute*** can be determined by analyzing other attributes. For example, in Figure 3.3 *Age* is a derived attribute because its value can be derived from the current date and the attribute *DateofBirth*. An attribute whose value cannot be derived from the values of other attributes is called a ***stored attribute***. As we will learn, a derived attribute *Age* is not stored in the database. Derived attributes are depicted in the E-R diagram with a dashed ellipse.

A **derived attribute** can be obtained from other attributes or related entities. For example, the radius of a sphere can be determined from the circumference. We request the derived attribute with a dotted oval and line. As another example, consider that the employee entity set has the related attributes start_date and employment_length, which represent the first day an employee began working for the bank and the total length of time an employee has worked for the bank, respectively. The value for employment_length can be derived from the value for start_date and the current date . In this case, start_date may be referred to as a base attribute , or a stored attribute.

ER diagram notation for derived attribute, *radius*

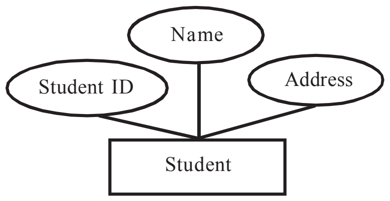


(d) Key Attribute :

A ***key attribute*** (or identifier) is a single attribute or a combination of attributes that uniquely identify an individual instance of an entity type. No two instances within an entity set can have the same key attribute value. For the STUDENT entity , *StudentID* is the key attribute since each student identification number is unique. *Name*, by contrast, cannot be an identifier because two students can have the same name. We underline key attributes in an E-R diagram .

Sometimes no single attribute can uniquely identify an instance of an entity type. However, in these circumstances, we identify a set of attributes that, when combined, is unique for each entity instance. In this case the key attribute, also known as ***composite key***, is not a simple attribute, but a composite attribute that uniquely identifies each entity instance.

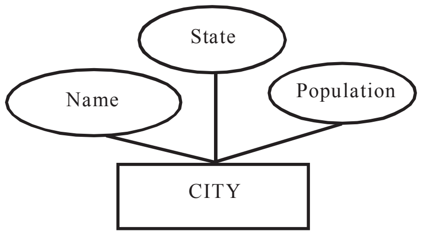
The concept of a key allows us to make such distinctions. However, to determine the class we need a **composite key** that consists of several attributes, such as catalogue number, section, semester, and year. We underline the composite key, *class*



The Key Attribute

A composite key must be minimal in the sense that no subset of a composite key can form the key of the entity instance. For example, if a composite key has four attributes, A1 to A4, then any subset, say A2, A4 or A2, A3 (or any of 16 combinations), should not form a key for an entity. In other words, we need all attributes, A1–A4, to identify each instance of an entity uniquely. In the E-R diagram, we underline each attribute in the composite key.For example, consider the CITY entity type. This category includes, potentially,all the cities in the United States. Notice that none of the attributes (i.e. *Name*, *State* or

Population) can serve as a key attribute since there are many cities in each state and two cities could possibly have the same name or population. However, the composite attribute {*Name*, *State*} is a valid key attribute for the CITY entity as no two cities within a state can have the same name.An entity can have more than one attribute that qualifies to be an identifier. For the entity, each of the attributes *Name*, *StateAbbr*, and *UnionOrder* (the order in which the state entered the union of the United States) can be an identifier. In this case, it is a matter of preference as to which attribute is made an identifier or key attribute.



The composite key attribute

(e) Null attributes :

Sometimes the value of an attribute is unknown or missing, and sometimes a value is not applicable. In such cases, the attribute can have the special value of **null**. **Null** is the special attribute value that indicates an unknown or missing value.

For example, until the professor grades a laboratory assignment, the team grade is missing or null. For a student who is auditing a course but participating as a team member, it is not applicable for that student to have an individual grade; the student's grade can have the value of null.

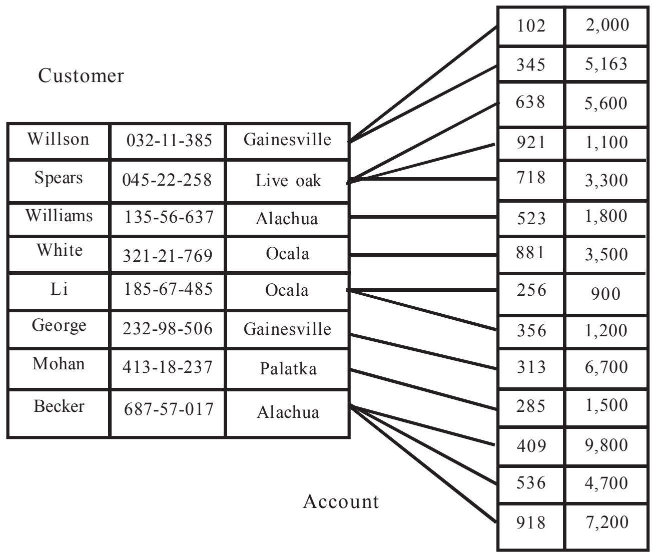
4.4 Relationships

Entities in an organization do not exist in isolation but are related to each other. Students take courses and each STUDENT entity is related to the COURSE entity. Faculty members teach courses and each FACULTY entity is also related to the COURSE entity. Consequently, the STUDENT entity is related to the FACULTY entity through the COURSE entity. E-R diagrams can also illustrate relationships between entities.

A ***relationship*** **set** is a mathematical relation among $n = 2$ entities, each taken from entity sets $\{(e_1, e_2, \dots e_n) \mid e_1 . E_1, e_2 . E_2, \dots, e_n . E_n\}$ where $(e_1, e_2, \dots, e_n)$ is a relationship

We define a ***relationship*** as an association among several entities. Consider, for example, an association between customers of a bank. If customer Williams has a bank account number 523, then the quality of ownership constitutes a ***relationship instance*** that associates the CUSTOMER instance Williams with the ACCOUNT instance *523*. We can think of the relationship instance as a verb that links a subject and an object: customer Williams *has* an account; student John *registers* for a course; professor Smith *teaches* a course. A ***relationship set*** is a grouping of all matching relationship instances, and the term ***relationship type*** refers to the relationship between entity types.

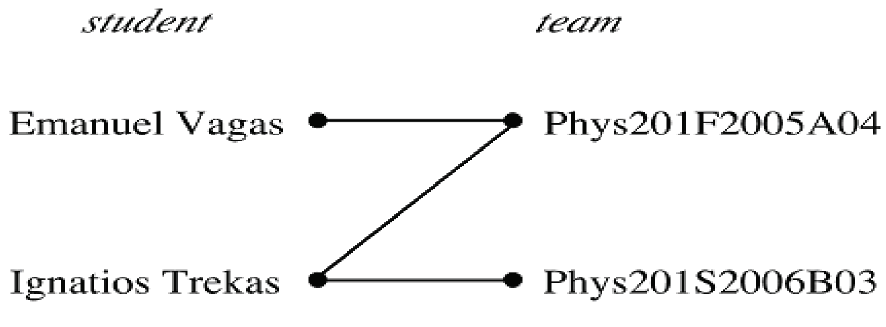
In an E-R diagram, we represent relationship types with diamond-shaped boxes connected by straight lines to the rectangles that represent participating entity types. A relationship type is a given name that is displayed in this diamond-shaped box and typically takes the form of a present tense verb or verb phrase that describes the relationship. An E-R diagram may depict a relationship as the following example of the relationship between the entities CUSTOMER and ACCOUNT does:



The relationship set between the CUSTOMER and ACCOUNT entities.

A **relationship type** is a set of associations among entity types. For example, the *student* entity type is related to the *team* entity type because each student is a member of a team. In this case, a **relationship** or **relationship instance** is an ordered pair of a specific student and the student's particular physics team, such as (Emanuel Vagas, Phys201F2005A04), where Phys201F2005A04 is Emanuel's team number. illustrates three relationships. Unfortunately, Itnatios Trekas had to drop the course and retake it another semester. Consequently, his name is associated with two team numbers.

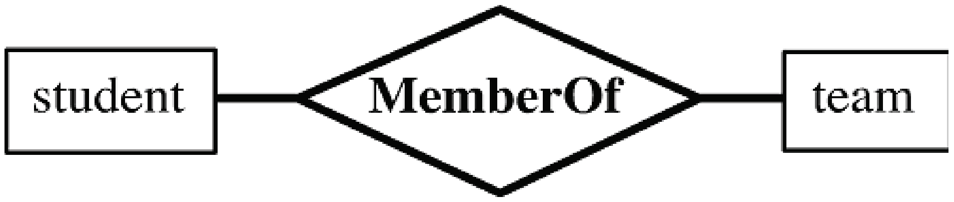
Relationships (Emanuel Vagas, Phys201F2005A04), (Ignatios Trekas,Phys201F2005A04), and (Ignatios Trekas, Phys201S2006B03)



We use a diamond to illustrate the relationship type in an ER diagram . We arrange the diagram so that the relationship reads from left to right, "a student is a member of a team." Alternatively, we can arrange the components from top to bottom.

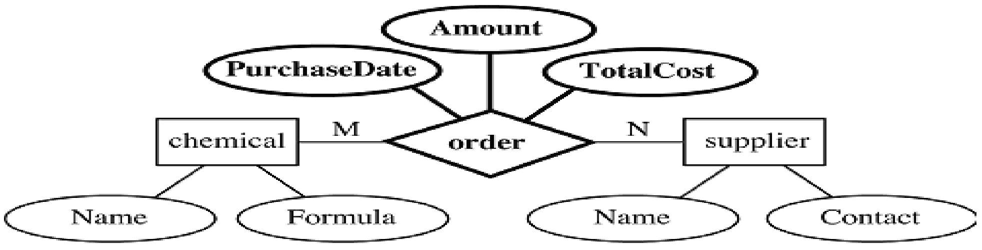
A **relationship type** is a set of associations among entity types. A **relationship** or **relationship instance** is an ordered pair consisting of particular related entities.

For example: ER diagram notation for relationship type, *MemberOf*



A relationship type can also have attributes. The relationship type *order* connects entities *chemical* and *supplier*. The relationship is many-to-many because each chemical can be from several suppliers and each supplier has a number of chemicals. An order has a purchase date, amount, and total cost as well as the chemical and supplier information. Thus, *order* has attributes *PurchaseDate*, *amount*, and *TotalCost* that we cannot appropriately associate with *chemical* or *supplier*.

For example: Relationship type with attributes



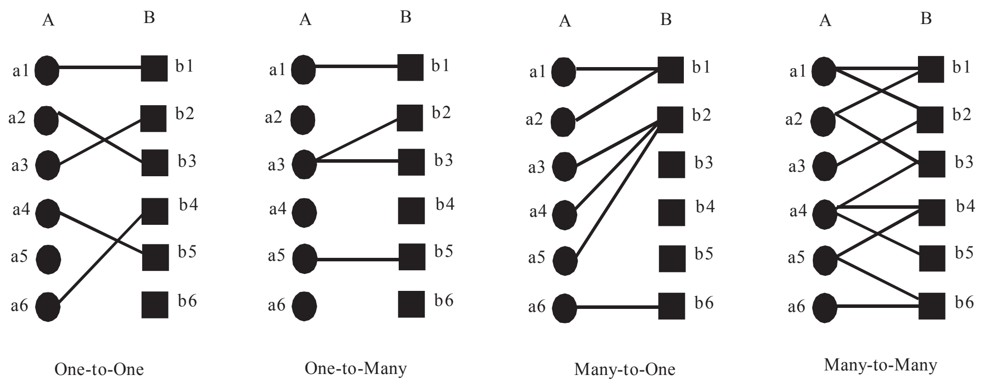
4.5 Constraints

Relationship types usually have certain constraints that limit the possible combinations of entities that may participate in the corresponding relationship set. These constraints are determined from the miniworld situation that the relationships present. An important constraint on the entities of an entity type is the key or uniqueness constraint on attributes. An entity type usually has an attribute whose values are distinct for each individual entity in the collection.

4.5.1 Cardinality Constraints

The term *cardinal number* refers to the number used in counting. An *ordinal number*, by contrast, emphasizes the order of a number (1st, 7th, etc.). When we say cardinality of a relationship, we mean the ability to count the number of entities involved in that relationship. For example, if the entity types A and B are connected by a relationship, then the ***maximum cardinality*** represents the maximum number of instances of entity B that can be associated with any instance of entity A.

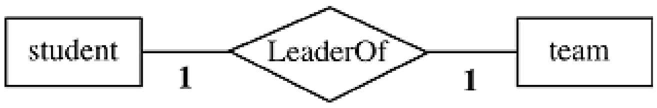
However, we don't need to assign a number value for every level of connection in a relationship. In fact, the term *maximum cardinality* refers to only two possible values: one or many. While this may seem to be too simple, the division between one and many allows us to categorize all of the permutations possible in any relationship. The maximum cardinality value of a relationship, then, allows us to define the four types of relationships possible between entity types A and B.



One-to-One Relationship : In a one-to-one relationship, at most one instance of entity B can be associated with a given instance of entity A and vice versa.

If each team can have at most one student leader and a student can be a leader of at most one team, we have a **1:1** or **one-to-one relationship**. We can illustrate this ratio by writing ones on the lines indicating the relationship.

For example : ER diagram notation for one-to-one relationship

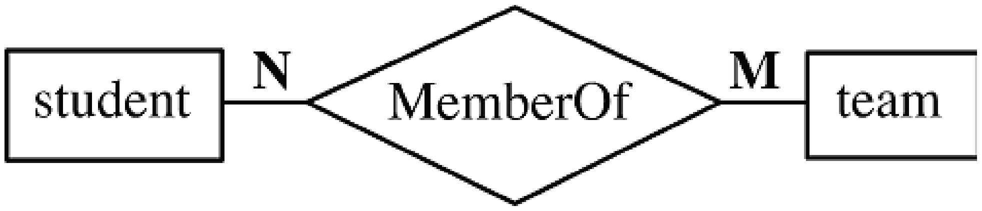


One-to-Many Relationship : In a one-to-many relationship, many instances of entity B can be associated with a given instance of entity A. However, only one instance of entity A can be associated with a given instance of entity B. For example, while a customer of a company can make many orders, an order can only be related to a single customer.

Many-to-Many Relationship : In a many-to-many relationship, many instances of entity A can be associated with a given instance of entity B, and, likewise, many instances of entity B can be associated with a given instance of entity A. For example, a machine may have different parts, while each individual part may be used in different machines.

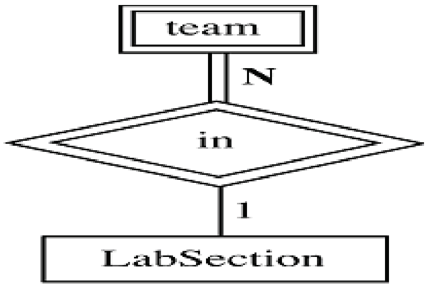
Another example is that a student can retake a course, a student could be a member of more than one team. Thus, we would need different variables, say *N* and *M*, for the numbers of student and team entities, respectively; and the ratio would be ***N:M*** in this **many-to-many relationship**. For example, a team has any number of students, and a student can participate in several teams during his or her college career. Ratios, such as 1:1, 1:*N*, *N*:1, and *N:M*, give a **cardinality constraint** or numeric restriction on the possible relationships.

For example :ER diagram notation for many-to-many relationship



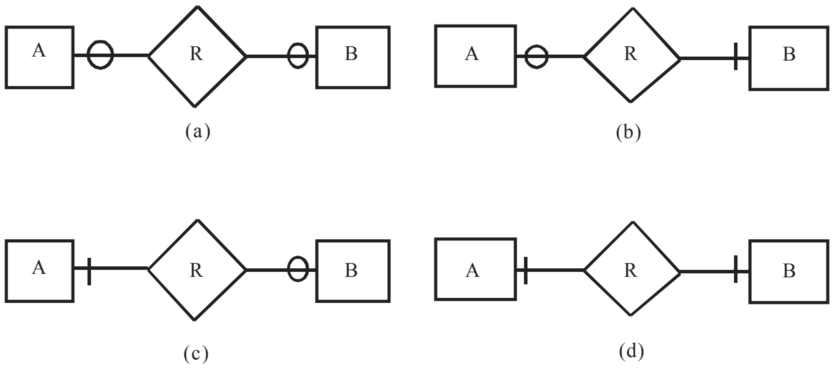
Many-to-one Relationship : Several teams can be in each lab section, students as members, so we write the cardinality ratio as ***N*:1** in this **many-to-one relationship** and draw the diagram. Similarly, we can have a **one-to-many relationship**.

For example : ER diagram notation for many-to-one relationship



Representing Relationship Types : It defines that how we represent different relationship types in an E-R diagram. An entity on the one side of the relationship is represented by a vertical line, “I,” which intersects the line connecting the entity and the relationship. Entities on the many side of a relationship are designated by a crowfoot.

We will now discuss the minimum cardinality of a relationship. The ***minimum cardinality*** between two entity types A and B is defined as the minimum number of instances of entity B that must be associated with each instance of entity A. In an E-R diagram, we allow the minimum cardinality to take two values: zero or one. If the minimum cardinality is zero, we say that entity type B is an *optional* participant in the relationship; otherwise, it is a *mandatory* participant. An optional relationship is represented by an "O" and mandatory relationship is represented by "|" in an E-R diagram.



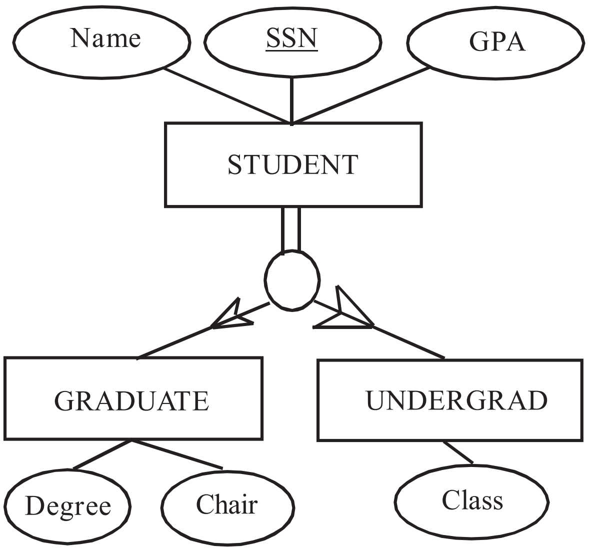
The given figure shows the four possibilities of the minimum cardinality of a relationship between two entity types A and B. Figure (a) depicts a situation in which no minimum cardinality constraints exist between the instances of entities A and B, meaning both entities A and B are optional participants in the relationship. Figure (b) illustrates a situation in which each instance of entity B must be associated with at least one instance of entity A, but no association is required for an instance of entity A. Figure (c) illustrates a situation in which each instance of entity A must be associated with at least one instance of entity B, but no association is required for an instance of entity B. Finally, Figure (d) illustrates a situation in which each instance of entity A and B must be associated with at least one instance of entity B and A, respectively.

An E-R diagram displays both the maximum and the minimum cardinalities of the relationships between two entities. Since there are four basic possibilities of maximum cardinalities and four possibilities of minimum cardinalities between two entities, there are 16 types of relationships possible between two entities in terms of cardinality. We will see several examples of these relationships while studying unary, binary, and ternary relationships.

4.5.2 Participation Constraints

4.5.2.1 Participation constraints dictate whether each instance (member) of a superclass must participate as an instance (member) of a subclass. A participation of superclass instance may be mandatory or optional in one or more subclasses. The mandatory constraint is also known as a total participation (constraint) or total specialization rule, while an optional constraint is known as a partial participation (constraint) or partial specialization rule.

4.5.2.2 Total Participation Rule In total participation, membership is mandatory. Each instance of a superclass must be an instance of at least one subclass.



Every instance of the superclass STUDENT must be an instance of either the GRADUATE student or UNDERGRAD student subclass. That is, if John Doe is an instance of a STUDENT, then John must be a graduate or undergraduate student. However, whether John Doe belongs to the graduate, undergraduate, or both entity types is answered by the disjoint rule, which we will define in a moment. We use a double line between the superclass entity type and the circle to represent the total participation.

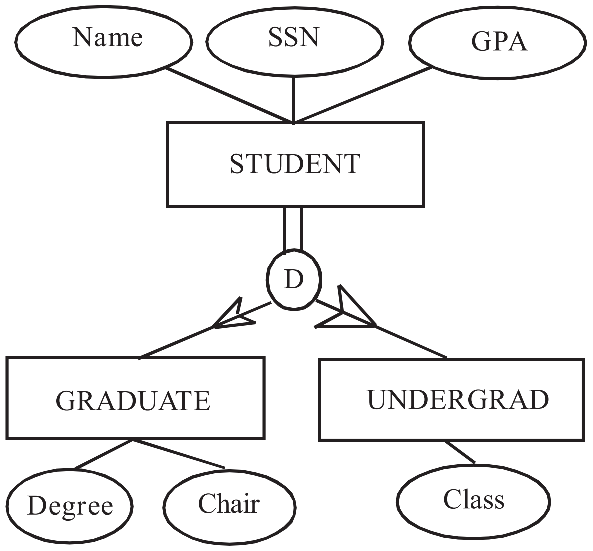
4.5.2.3 Partial Participation Rule Membership is optional in a partial participation. An instance of a superclass does not have to be an instance of any of the subclasses. An instance of the LIBRARY ITEM superclass can be a member of BOOK, VIDEO CD, or JOURNALS; however it is not mandatory for an instance to belong to any of these subclasses.

If the library item *Newspaper* is an instance of a superclass, it does not have to be included in one of the subclasses; it can stay at the superclass level without having values for any subclass attributes. We use a single line between the superclass entity type and the circle (the default notation) to represent partial participation.

4.5.3 Disjoint Constraints

4.5.3.1 Disjoint constraints define whether it is possible for an instance of a superclass to simultaneously be a member of one or more subclasses. Disjoint constraints indicate whether a superclass instance can be disjointed or overlap more than one subclass.

4.5.3.2 Disjoint Rule The disjoint rule states that if an instance of a superclass is a member of any subclass, then it cannot be a member of more than one subtype (note that the participation rule will dictate whether the instance is a member of a subclass or not). We put a constraint of disjoint rule to indicate that a student must be either a graduate or an undergraduate student but cannot belong to both subclasses simultaneously. We indicate the disjoint rule by putting a letter "D" in the joining circle of the superclass/ subclass relationship.

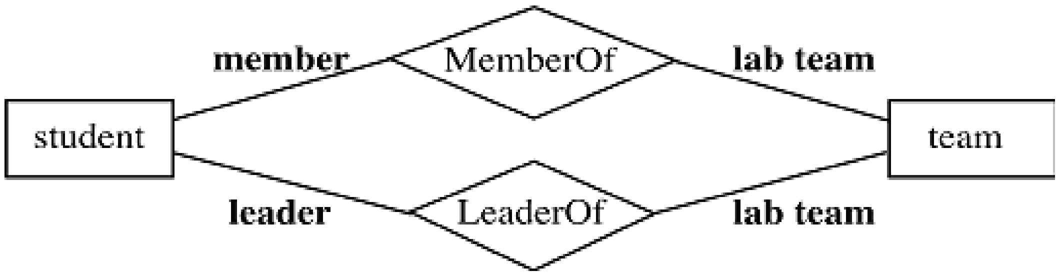


An Example of the Disjoint Rule

4.6 Degree and Domain

The **degree** of a relationship type is the number of entity types that participate. Thus, the *LabSecMemberOf* relationship type has degree 2, which we call a *binary* relationship type. To clarify the role that an entity plays in each relationship instance, we can label a connecting edge with a **role name** that indicates the purpose of an entity in a relationship. For example, we can have two binary relationship types associating the *student* and *team* types, *TeamMemberOf* and *LeaderOf*. In the former case, a student entity is a member of a team entity; in the later case, a student can be a leader of a team.

The **degree** of a relationship type is the number of entity types that participate. If two entity types participate, the relationship type is **binary**. A **role name** indicates the purpose of an entity in a relationship. R diagram notation with roles *member*, *leader*, and *lab team*

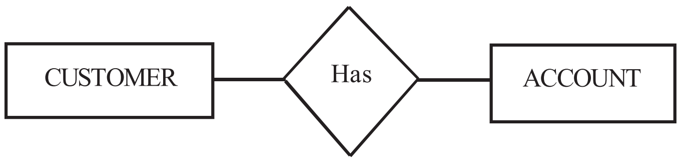


The number of entity sets that participate in a relationship is called the ***degree of relationship***.

For example, the degree of the relationship featured in Figure 3.8 is two because CUSTOMER and ACCOUNT are two separate entity types that participate in the relationship. The three most common degrees of a relationship in a database are unary (degree 1), binary (degree 2), and ternary (degree 3). We will briefly define these degrees and then explore each kind of relationship in detail in subsequent sections.

Let $E_1, E_2, \ldots, E_n$ denote *n* entity sets and let R be the relationship. The degree of the relationship can also be expressed as follows:

4.6.1 Unary Relationship A unary relationship R is an association between two instances of the same entity type (i.e., R . $E_1 \times E_1$). For example, two students are roommates and stay together in an apartment. Because they share the same address, a unary relationship exists between them for the attribute *Address.*



4.6.2 Binary Relationship A binary relationship R is an association between two instances of two different entity types (i.e., R . E1 × E2). For example, in a university, a binary relationship exists between a student (STUDENT entity) and an instructor (FACULTY entity) of a single class; an instructor *teaches* a student.

4.6.3 Ternary Relationship A ternary relationship R is an association between three instances of three different entity types (i.e., R . E1 × E2 × E3). For example, consider a student using certain equipment for a project. In this case, the STUDENT, PROJECT, and EQUIPMENT entity types relate to each other with ternary relationships: a student *checks out* equipment for a project.

4.7 Data modeling using the Entity Relationship Model

Conceptual modeling is an important phase in designing a successful database application. Generally, the term database application refers to a particular database. These programs often provide user-friendly graphical user interfaces(GUI's) utilizing forms and menus. Hence, part of the database application will require the design, implementation and testing of these application programs. Traditionally the design and testing of application programs has bee considered to be more in the realm of the software Engineering domain than in the database domain.

4.7.1 E-R model concepts

The E-R data model gives us substantial flexibility in designing a database schema to model a given enterprise. In this section we consider how a database designer may select from the wide range of alternatives. Among the decisions to be made are the following :

(a) Whether to use an attribute or an entity set to present an object.

(b) Whether a real-world concept is expressed most accurately by an entity set or by a relationship set.

(c) Whether to use a ternary relationship or a pair of binary relationships.

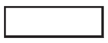
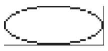
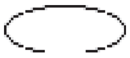
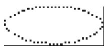
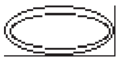
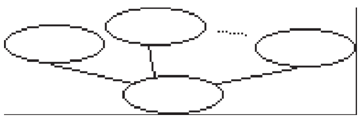
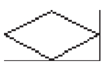
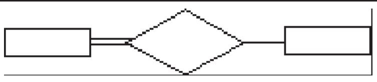
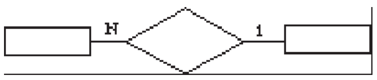
(d) Whether to use a strong or a weak entity set(section 4.8.3), a strong entity set and its dependent weak entity set may be regarded as a single 'object' in the database, since weak entities are existence dependent on a strong entity set.

(e) Whether using generalization is appropriate , generalization is a hierarchy of a relationships contributes to modularity by allowing common attributes of similar entity sets to be presented in one place in an E-R diagram.

(f) Whether using aggregation is appropriate, aggregation groups a part of an E-R diagram into a single entity set, allowing us to treat the aggregate entity set as a single unit without concern for the details of its internal structure.

4.7.2 Notation for E-R Diagram

The overall logical structure of a database can be expressed graphically by an E-R diagram. The relative simplicity and pictorial clarity of this diagramming may well account in large part for the widespread use of the E-R model. The choices of names for entity types, attributes, relationship types and roles is not always straightforward. One should choose names that convey as much as possible , the Such a diagram consists of the following major components. Summary of the ER diagram notation are as follows:

Notation	Meaning
	Entity type
	Attribute
	Key attribute
	Derived attribute
	Multivalued attribute
	Composite attribute
	Relationship type
	Total participation
	Many-to-one relationship

Rectangles : which represent entity sets.

Ellipses : which represent attributes.

Diamonds : which represent relationship sets.

Lines : which link attributes to entity sets and entity sets to relationship sets.

Double ellipses : which represent multivalued attributes.

Dashed ellipses : which denote derived attributes.

Double lines : which indicate total participation of an entity in a relationship set.

Directed line : a directed line from relationship set to the entity set specifies that relationship is either a one-to one, or many to one relationship set.

Undirected line : a undirected line from the relationship set relationship to the entity set specifies that relationship is either a many to many, or a one to many relationship set .

Underlined ellipses : which indicate key attributes.

4.7.3 Entity sets

We know that a database usually contains group of entities that are similar. An entity type defines a collection of entities that have the same attributes. Each entity type in the database is described by its name and attributes. The collection of all entities of a particular entity type in the database at any point in time is called an entity set, the entity set is usually referred to using the same name as the entity type. There are times you might wish to define an entity set even though its attributes do not formally contain a key.

Entity sets do not need to be disjoint. For example, it is possible to define the entity set of all employees of a bank and the entity set of all customers of the bank. A person entity may be an employee entity, a customer entity, both or neither.

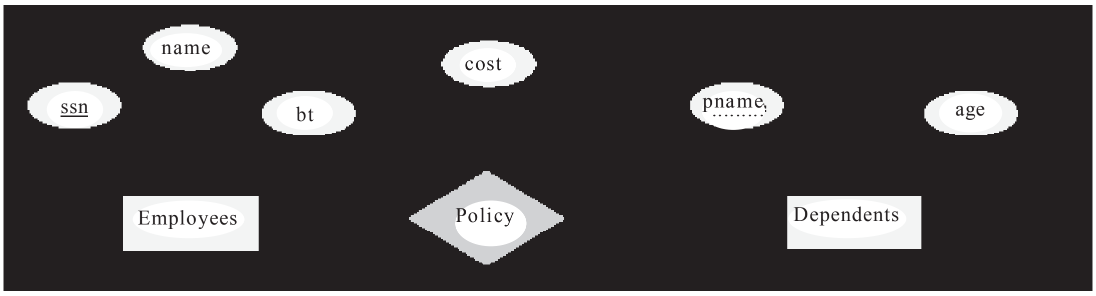
An entity set may not have sufficient attributes to form a primary key. Such an entity set is termed a weak entity set. Usually, this is the case only because the information represented in such an entity set is only interesting when combined through an **identifying relationship set** with another entity set we call the **identifying owner**.

We will call such a set a **weak entity set**, and insist on the following:

- The weak entity set must exhibit a key constraint with respect to the identifying relationship set.
- The weak entity set must have total participation in the identifying relationship set.

Together, this assures us that we can uniquely identify each entity from the weak set by considering the primary key of its identifying owner together with a **partial key** from the weak entity.

In our ER diagrams, we will represent a weak entity set by outlining the entity and the identifying relationship set with dark lines. The required key constraint and total participation are diagrammed with our existing conventions. We underline the partial key with a dotted line.



In the physics laboratory ER model example, the entity type *student* is **strong** because its existence does not depend on some other entity type. However, the *team* entity type is **weak**. The existence of *team* depends on the existence of *LabSection,* and we call the *in* identifying relationship. We draw double lines around the identifying relationship, the *team* entity type, and the line connecting the two to indicate the weak entity type.

A weak entity set does not have sufficient attributes to form a primary key (not globally unique), a discriminator is selected for the weak entity set.

- The discriminator of a weak entity set is a set of attributes that allow us to distinguish entities that are all dependent on one particular strong entity set.

  Example (of discriminator):

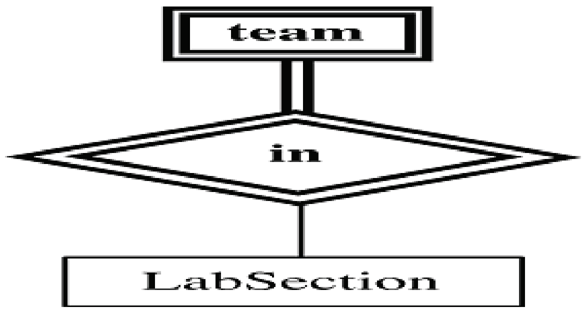
  { Section number }
- The primary key of a weak entity set can be formed by the primary key of the strong entity set on which it is existence dependent, plus its discriminator.

  Example:

  {Course name, section number }

  or, {Course number, section number }

Note : An entity type is **strong** if its existence does not depend on some other entity type. Otherwise, the entity type is **weak**.



4.8 Keys

An attribute or set of attributes that uniquely identifies a particular entity is a **key**. For example, Emanuel Vagas' Student Identification Number uniquely identifies him. It is important to be able to specify how entities within a given entity set and relationships within a given relationship set are distinguished..

4.8.1 Concepts of Super Key

A super key of an entity set is a set of one or more attributes whose values uniquely determine each entity. The combination of primary keys of the participating entity sets forms a super key of a relationship set.

(customer-id, account-number) is the super key of depositor

Note : This means a pair of entity sets can have at most one relationship in a particular relationship set. E.g. if we wish to track all access-dates to each account by each customer, we cannot assume a relationship for each access.

Superkey: a set of one or more attributes which, taken collectively, allow us to identifying an entity instance in an entity set uniquely.

Example:

{ student ID, name}

{ social insurance number, address }

{ row number, column number }

4.8.2 Concepts of Candidate Key

A minimal set of attributes in a table that uniquely identifies a record. When there is more than one attribute in the candidate key, it is called composite key. A candidate key that is chosen to represent a record uniquely. Each entity type must have an attribute or set of attributes that distinguishes one instance from other instances of the same type. Attribute (or combination of attributes) that uniquely identifies each instance of an entity type .

Candidate key : a smallest possible superkey. i.e. a superkey for which no proper subset is a superkey.

Example :

{ student ID }

{ social insurance number }

{ row number, column number }

4.8.3 Concepts of Primary Key

A table can have at most one primary key, but more than one unique key. A primary key is a combination of columns which uniquely specify a row. It is a special case of unique keys. One difference is that primary keys have an implicit NOT NULL constraint while unique keys do not. Thus, the values in unique key columns may or may not be NULL, and in fact such a column may contain at most one NULL fields.

Another difference is that primary keys must be defined using another syntax. That is, a table may consist of many candidate keys, but ONLY ONE can be selected as a primary key. attribute we say that the primary key is a composite primary key. One of the common mistakes students have made is to refer to one of the attributes in the composite key as primary key. Please note that when there is more than one attribute in the primary key ALL ATTRIBUTES TOGETHER are called the primary key since they together define the record uniquely.

Primary key: a candidate key chosen by database designer as the principal means of identifying entities within an entity set.

Example :

{ student ID }

Primary key selection criteria:

{ not change over time}

{ no null value}

4.9 Extended E-R Features

The processes of *specialization, generalization* and *Aggregation* are used to find such opportunities. These processes serve as conceptual models for the development of superclass/subclass relationships.

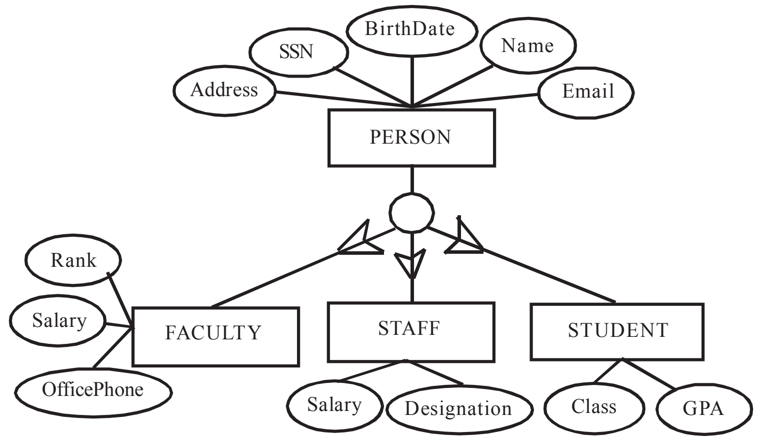
4.9.1 Generalization

Generalization is the process of defining general entity types from a set of specialized entity types by identifying their common characteristics. In other words, this process minimizes the differences between entities by *identifying a general entity type* that features the common attributes of specialized entities. Generalization is a bottom-up approach as it starts with the specialized entity types (subclasses) and forms a generalized entity type (superclass).

For example, suppose that someone has given us the specialized entity types FACULTY, STAFF, and STUDENT, and we want to represent these entity types separately in the E-R model as depicted in Figure 3.25(a). However, if we examine them closely, we can observe that a number of attributes are common to all entity types, while others are specific to a particular entity.

For example, FACULTY, STAFF, and STUDENT all share the attributes *Name*, *SSN*, *Birth Date*, *Address*, and *Email*. On the other hand, attributes such as *GPA*, *Class*, and *MajorDept* are specific to the STUDENTS; *OfficePhone* is specific to FACULTY, and *Designation* is specific to STAFF. Common attributes suggest that each of these three entity types is a form of a more general entity type. This general entity type is simply a PERSON superclass entity with common attributes of three subclasses .

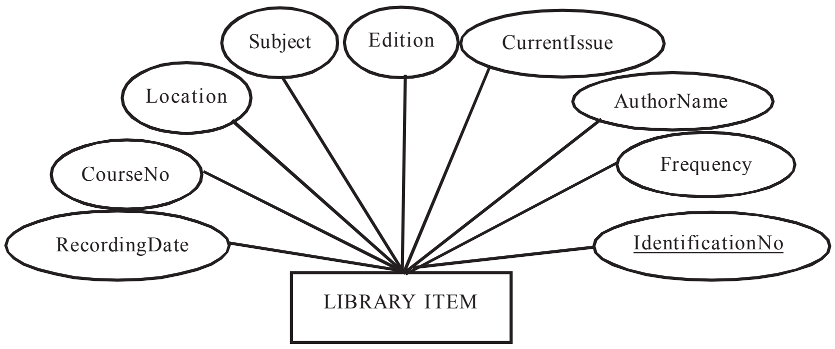
Thus, in the generalization process, we group specialized entity types to form one general entity type and identify common attributes of specialized entities as attributes of a general entity type. The general entity type is a superclass of specialized entity types or subclasses.

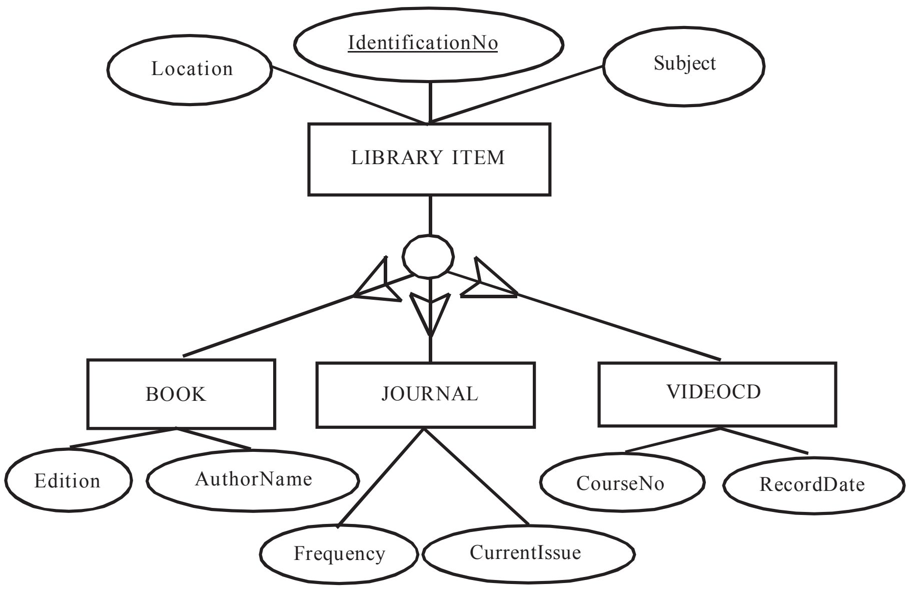


4.9.2 Specialization

Specialization is the process of defining one or more subclasses of a superclass by identifying its distinguishing characteristics. Unlike generalization, specialization is thus a top-down approach. It starts with the general entity (superclass) and forms specialized entity types (subclasses) based on specialized attributes or relationships specific to a subclass.

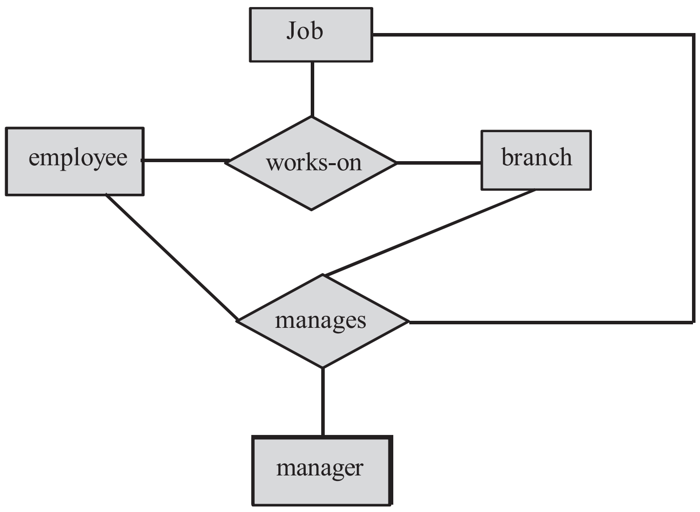
For example, consider LIBRARY ITEM is an entity type with several attributes such as *IdentificationNo*, *RecordingDate*, *Frequency*, and *Edition*. After careful review of these items, it should become clear that some items such as books do not have values for attributes such as *Frequency*, *RecordingDate*, and *CourseNo*, while Video CDs do not have an *Author* or an *Edition*. In addition, all items have common attributes such as *IdentificationNo*, *Location*, and *Subject*. Someone creating a library database, then, could use the specialization process to identify superclass and subclass relationships. In this case, the original entity LIBRARY ITEM forms a superclass entity type made up of attributes shared by all items, while specialized items with distinguishing attributes, such as BOOK, JOURNALS, and VIDEOCD, form subclasses.



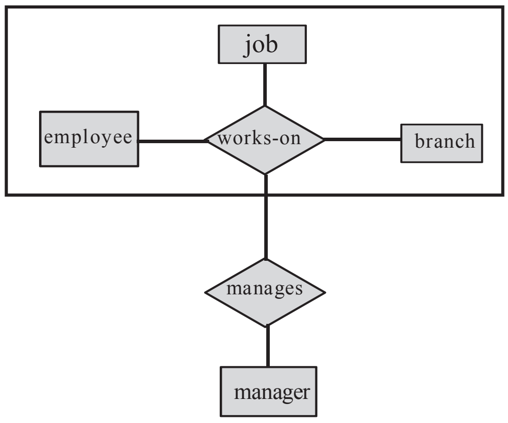


4.9.3 Aggregation

One limitation of the E-R model is that it is not possible to express relationships among relationships. To illustrate the need for such a construct, we consider again a database describing information about employee and branch. Suppose that each employee-branch pair may work with the managers. Using our basic E-R modeling constructs, we obtain the E-R diagram. It appears that the relationship set works-on and manages can be combined into one single relationship set. We should not combine them, because doing so would obscure the logical structure of this schema. For example if we consider the works-on and manages relationship sets, then this combination specifies that a manages must be assigned to every employee-branch pair, which is not true.



There is redundant information in the resultant figure, however , since every employee-branch pair in manages is also in works-on. The best way to model a situation such as the one just described is to use aggregation. Aggregation is an abstraction through which relationships are treated as higher-level entities. Thus we regard the relationship set work-on and the entity sets employee and branch as a higher-level entity set called works-on. Such en entity set is treated in the same manner as in any other entity set.



4.10 Summary

- Attribute - a property or description of an entity. A toy department employee entity could have attributes describing the employee's name, salary, and years of service.
- Domain - a set of possible values for an attribute.
- Entity - an object in the real world that is distinguishable from other objects such as the green dragon toy.
- Relationship - an association among two or more entities.
- Entity set - a collection of similar entities such as all of the toys in the toy department.
- Relationship set - a collection of similar relationships
- One-to-many relationship - A key constraint that indicates that one entity can be
- Participation constraint - a participation constraint determines whether relationships must involve certain entities. An example is if every department entity has a manager entity. Participation constraints can either be total or partial. A total participation constraint says that every department has a manager. A partial participation constraint says that every employee does not have to be a manager.
- Weak entity set - an entity that cannot be identified uniquely without considering some primary key attributes of another identifying owner entity. An example is including Dependent information for employees for insurance purposes.
- Aggregation - a feature of the entity relationship model that allows a relationship set to participate in another relationship set. This is indicated on an ER diagram by drawing a dashed box around the aggregation.
- Role indicator - If an entity set plays more than one role, role indicators describe the different purpose in the relationship. An example is a single Employee entity set with a relation Reports-To that relates supervisors and subordinates.

4.11 Self Assessment Questions

Q.1 Explain the following terms using an example: entity-relationship model, entity type, weak entity, attribute, key attribute, derived attribute, multi-valued attribute.

Q.2 Explain, using an example (other than the one discussed in the book), the contrast between the following terms:

(a) entity type; entity instance

(b) strong entity type; weak entity type

(c) simple attribute; composite attribute

(d) stored attribute; derived attribute

Q.3 Provide an example of multiple relationships between entities. Draw the E-R diagram.

Q.4 Under what conditions is a relationship converted to an associative entity type? Give an example.

Q.5 Explain why we study the E-R model of a database.

Q.6 A department in a university stores the information about its students and courses in a database. The administrative assistant manages the database. At the end of the semester, he prepares a report about each course. Is the E-R diagram correct? If not, explain why and draw the correct diagram.

Q.7 Explain the distinctions among the terms primary key, candidate key, and super key.

Q.8 Define the concepts of Aggregation. Give two examples of where this concept is useful.

Q.9 Explain the difference between a Strong and weak entity set.

Q.10 Explain the distinction between total and partial design constraints. Explain your answer.

Unit - 5 : Relational Model

Structure of the Unit

5.0 Objective

The objective of this module is to make student familier with relational data base. After learning this module student able to write database queries in the relational algebra. They also know how to querying the data.

5.1 Introduction

E.F. Codd's give the concept of relational algebra in 1970. Codd proposed an algebra as a basis for database query languages.Relational algebra based on relational model. R elational model is a part of record base data mode .It define data at logiacl and view level.

5.2 Data Models

The data model are collection of tools for describing: data, data relationships, data semantics, data constraints or we can say data models are collection of conceptual tools describing data relationship among data consistency constraints. Basic data model are :

5.2.1 Object-Based Logical Models

This data model is based on real world that consists of basic objects called entities and of relationship among these objects. Entities are described in a database by a set of attributes. Example of object based logical models is :

5.2.1.1 Object Oriented Model

This model is based on collection of objects. An object contains values stored in instance variables with in the object. An object also contains bodies of code that operate on the object. These bodies of code are called methods. Objects that contain same types of values and the same methods are grouped together into classes.

Example of Object-based logical models is entity-relationship model, object-oriented model.

5.2.2 Record-Based Logical Model

It also describe be data at the conceptual and view levels. Unlike object-oriented models, are used to specify overall logical structure of the database, and provide a higher-level description of the implementation. The database is structured in fixed-format records of several types. Each record type defines a fixed number of fields, or attributes. Each field is usually of a fixed length (this simplifies the implementation). Record-based models do not include a mechanism for direct representation of code in the database. Separate languages associated with the model are used to express database queries and updates. The three most widely-accepted models are the relational, network, and hierarchical.

5.2.2.1 Relational Model

The relational model for database management is a database model proposed in 1969 by E.F. Codd. Data and relationships are represented by a collection of tables. Each table has a number of columns with unique names, e.g. Name ,EmpId, DeptName and City .

Examples of relational database a[illegible]QL, SQL Server, Access, Oracle, Sybase, DB2.

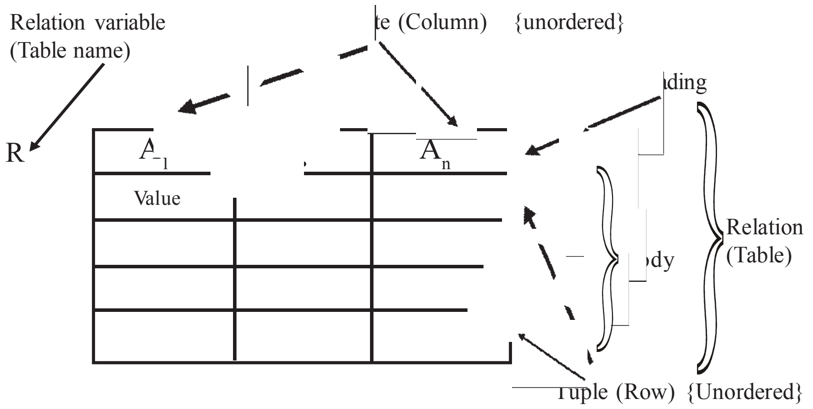


Figure 5.1 : A sample relational database.

Relation :

A relation is a table structure definition (a set of column definitions) along with the data appearing in that structure. The structure definition is the heading and the data appearing in it is the body, a set of rows. A database relvar (relation variable) is commonly known as a base table.Figure 5.2 show the employee is a relation.

Employee			
Name	**Emp Id**	**DeptName**	**City**
Harry	3415	Finance	Jaipur
Sally	2241	Sales	Delhi
Georgy	3401	Finance	Jaipur
Harriet	2202	Sales	Kota

Figure 5.2 : Employee- relational database.

Attribute/Column:

In the context of a relational database table, a column is a set of data values of a particular simple type, one for each row of the table.The columns provide the structure according to which the rows are composed. The term field is often used interchangeably with column, although many consider it more correct to use field (or field value) to refer specifically to the single item that exists at the intersection between one row and one column.

In relational database terminology, column's equivalent is called attribute. An attribute is an ordered pair of attribute name and type name. An attribute value is a specific valid value for the type of the attribute. This can be either a scalar value or a more complex type.

For example, The Employee relation have the following columns:

Name ,EmpId (identifier, unique to each row), DeptName, City.

Each row would provide a data value for each column and would then be understood as a single structured data value.The following terms are use with relational database .

Row/tuple :

A tuple is an ordered set of attribute values or a tuple is basically the same thing as a row. In a relation a row represent a record . For example in employee relation a row is like

Harry	3415	Finance	Jaipur

Cardinality :

The **cardinality** of a set is a measure of the "number of elements of the set". For example, the set A = {2, 4, 6} contains 3 elements, and therefore A has a cardinality of 3. similarly cardinality of a relation is the "Number of rows " in a relation. For example in employee relation total 4 rows or records so we can say cardinality is 4.

Degree :

Number of attributes in a relation is called Degree of a relation .In the given example Employee relation have 4 attribute (Name,EmpId,DeptName ,City) then degree is 4.

Domain :

A domain is a set of all possible data values. For example the domain of name is set of Name ('Harry',' Sally',' George',' Harriet').

	Column 1	**Column 2**
Row 1	Row1, Column 1	Row 1, Column 2
Row 2	Row2, Column 1	Row 2, Column 2
Row 3	Row 3, Column 1	Row 3, Column 2
Row 4	Row 4, Column 1	Row 4, Column 2

Figure 5.3 : Structure of a relation

In figure 5.3 ,the cardinality is 4 becose there are 4 records ,degree is 2 becase there is two columns.

The consistency of a relational database is enforced, not by rules built into the applications that use it, but rather by constraints , declared as part of the logical schema and enforced by the DBMS for all applications. In general, constraints are expressed using relational comparison operators, of which just one, "is subset of" is theoretically sufficient. In practice, several useful shorthands are expected to be available, of which the most important are candidate key (really, superkey) and foreign key constraints.

Self-Learning Exercise :

1. What is a Relation Schema and a Relation?

5.2.2.2 Network Model

In network model data are represented by collections of records. Relationships among data are represented by links. Organization is that of an arbitrary graph.

Figure 5.4 shows a sample network database that is the equivalent of the relational database.

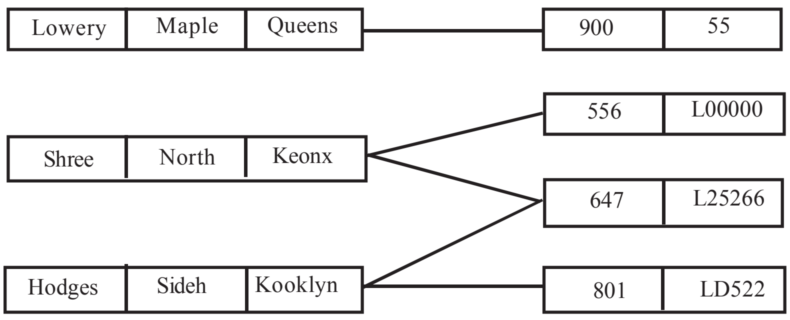


Figure 5.4 : A Sample Network Database

5.2.2.3 Hierarchical Model

It is similar to the network model. Organization of the records is as a collection of **trees**, rather than arbitrary graphs. Figure 5.5 shows a sample hierarchical database that is the equivalent of the relational database.

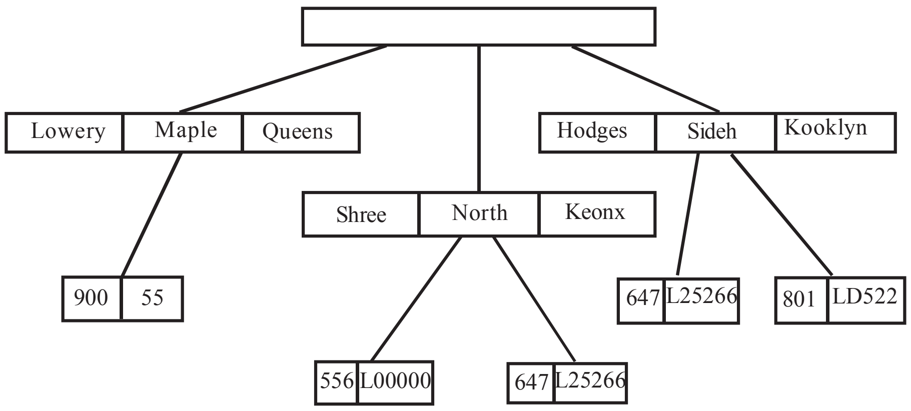


Figure 5.5 : A sample Hierarchical Database

The relational model does not use pointers or links, but relates records by the values they contain. This allows a formal mathematical foundation to be defined.example of hierarchical model is IMS.

5.2.3 Physical Data Models

The physical data model are used to describe data at the lowest level of abstraction, which is physical level. Very few models, e.g. Unifying model., Frame memory.

5.3 Relational Algebra

IBM's original implementation of Codd's ideas was System R. There have been several commercial and open source products based on Codd's ideas, including IBM's DB2, Oracle Database, Microsoft SQL Server, PostgreSQL, MySQL, and many others. Most of these use the SQL data definition and query language. A table in an SQL database schema corresponds to a predicate variable; the contents of a table to a relation; key constraints, other constraints, and SQL queries correspond to predicates. However, it must be noted that SQL databases, including DB2, deviate from the relational model in many details; Codd fiercely argued against deviations that compromise the original principles.Relational algebra is essentially equivalent in expressive power to relational calculus (and thus first-order logic); this result is known as Codd's theorem. We must be careful to avoid a mismatch, that may arise between the two languages since negation, applied to a formula of the calculus, constructs a formula that may be true on an infinite set of possible tuples, while the difference operator of relational algebra always returns a finite result. To overcome these difficulties, Codd restricted the operands of relational algebra to finite relations only and also proposed restricted support for negation (NOT) and disjunction (OR). Analogous restrictions are found in many other logic-based computer languages. Codd defined the term relational completeness to refer to a language that is complete with respect to first-order predicate calculus apart from the restrictions he proposed. In practice the restrictions have no adverse effect on the applicability of his relational algebra for database purposes.

A query language is a language in which user requests information from the database. It can be categorized as either procedural or nonprocedural. In a procedural language the user instructs the system to do a sequence of operations on database to compute the desired result. In nonprocedural language the user describes the desired information without giving a specific procedure for obtaining that information.

The relational algebra is a procedure language .It consist set of operation that take one and two relation as their inputs and produce a new relation as their result .

As in any algebra, some operators are primitive and the others are derived in terms of the primitive ones. It is useful if the choice of primitive operators parallels the usual choice of primitive logical operators. Although it is well known that the usual choice in logic of AND, OR and NOT is somewhat arbitrary, Codd made a similar arbitrary choice for his algebra. Five primitive operators of Codd's algebra are the selection, the projection, the Cartesian product (also called the cross product or cross join), the set union, the set difference . Another operator, rename was not noted by Codd, These six operators are fundamental in the sense that if you omit any one of them, you will lose expressive power. Many other operators have been defined in terms of these six. Among the most important are set intersection,division,and the natural join.

5.4 Fundamental Operation

Selection,Projection and Rename operation are called unary operations,because they operate on one relation.The other operation are called binary operation because They operate on more then one relation.

Symbolic Notation

From the example, one can see that for complicated cases a large amount of the answer is formed from operator names, such as PROJECT and JOIN.The following symbolic notation are use to represent the operators-

- SELECT ->σ (sigma)
- PROJECT -> π(pi)
- PRODUCT -> ×(times)
- JOIN -> |×| (bow-tie)
- UNION -> U (cup)
- INTERSECTION ->)”(cap)
- DIFFERENCE -> - (minus)
- RENAME ->ρ (rho)

RELATION r : Figure 5.6 show a relation

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

Figure 5.6 Relation r

1. Projection (Π)

A **projection** is a unary operation written as $\pi_{a_1,\ldots,a_n}(R)$ where $a_1,\ldots,a_n$ is a set of attribute names. The result of such projection is defined as the set that is obtained when all tuples in R are restricted to the set $\{a_1,\ldots,a_n\}$.

$\pi_{a_1,\ldots,a_n}(\pi_{b_1,\ldots,b_m}(R)) = \pi_{a_1,\ldots,a_n}(R)$ *where* $\{a_1,\ldots,a_n\} \subseteq \{b_1,\ldots,b_m\}$

Example of projection operation is $\pi_{A,C}$ (r) result of this operation is given below, here r is a relaion.

A	C
α	1
α	5
β	12
α	23

Figure 5.7 : $\pi_{A,C}$ (r) result

If we have employee relation then

Employee		
Name	**EmpId**	**Dept Name**
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

Suppose if we want to find name and empid for all the employee from employee then we write following statement -

$$\pi_{\text{Name,EmpId}} \text{ (Employee)}$$

Output of this statement is

Employee	
Name	**EmpId**
Harry	3415
Sally	2241
George	3401
Harriet	2202

Figure 5.8 : result of $\pi_{\text{Name,EmpId}}$ (Employee)

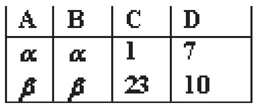
2. Selection (σ)

A generalized selection is a unary operation written as $\sigma_{\varphi}(R)$where $\varphi$is a propositional formula that consists of atoms as allowed in the normal selection and the logical operators $\wedge$(and), $\vee$(or) and $\neg$(negation). This selection selects all those tuples in R for which holds.

Example of selection operation

$\sigma_{A=B \wedge D>5}$ (r)

Result of this operation is

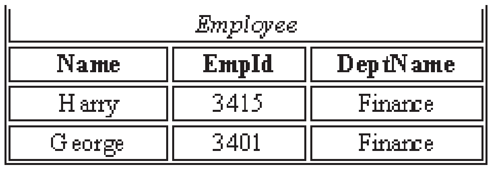


A	B	C	D
α	α	1	7
β	β	23	10

If we want to display the employee whose department is Finance from Employee relation then we write following statement:

$\sigma_{\text{DeptName= Finance}}$ (Employee)

Output of this statement is :



Employee		
Name	**EmpId**	**DeptName**
Harry	3415	Finance
George	3401	Finance

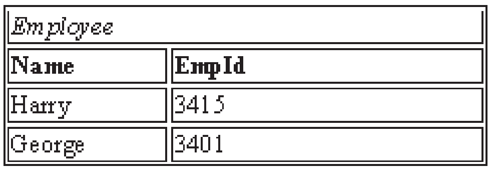
Figure 5.9 : result of $\sigma_{\text{DeptName= Finance}}$ (Employee)

3. Combine Selection and Projection $\subseteq$

If we combine both selection and project into a single relation e.g. selected attribute display according to the specific condition.

If we wants to display only name and employee id from employee relation whose department is Finance then we write following statement -

$\pi_{\text{Name,EmpId}}$ ($\sigma_{\text{DeptName= Finance}}$ (Employee))



Employee	
Name	**EmpId**
Harry	3415
George	3401

Figure 5.10 : result of $\pi_{\text{Name,EmpId}}$ ($\sigma_{\text{DeptName= Finance}}$ (Employee))

4. Rename (Ρ)

A rename is a unary operation written as $\rho_{a/b}$(R) where the result is identical to R except that the b field in all tuples is renamed to an a field. This is simply used to rename the attribute of a relation or the relation itself.Allows us to name, and therefore to refer to, the results of relational-algebra expressions. Allows us to refer to a relation by more than one name. Example:

ρ_X (E)

Returns the expression E under the name X

If a relational-algebra expression E has arity n, then

$\rho_X (A_{1,} A_2 , A_3 \ldots\ldots A_n)(E)$

Returns the result of expression E under the name X, and with the Attributes renamed to $A_{1,} A_2$, A_3...... A_n .

5. Set Operators

Although three of the six basic operators are taken from set theory, there are additional constraints that are present in their relational algebra counterparts: For set union and set difference, the two relations involved must be union-compatible—that is, the two relations must have the same set of attributes. Because set intersection can be defined in terms of set difference, the two relations involved in set intersection must also be union-compatible.

(a) Union :

Notation : r U s

Defined as : For r U s to be valid.

1. r, s must have the same **arity** (same number of attributes)
2. The attribute domains must be compatible (example: 2nd column of r deals with the same type of values as does the 2nd column of s)

r U s = {t | t $\in$ r or t $\in$ s} here t is tuple or row this expression is written in TRC which is explain latter

Example :

Employee		
Name	**EmpId**	**Dept Name**
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

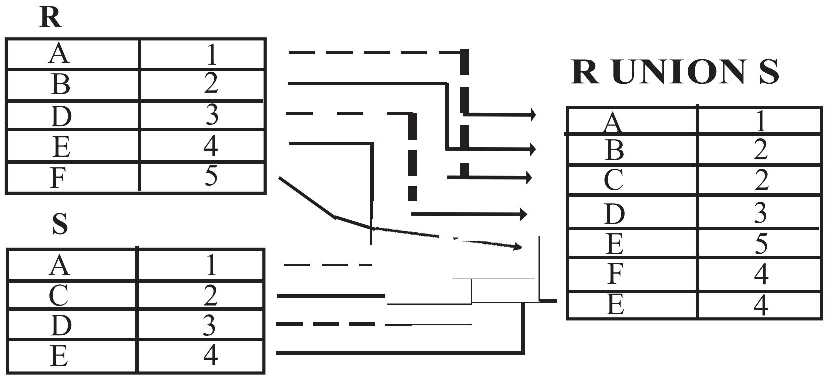
Dept.	
Dept Name	**Manager**
Finance	George
Sales	Harriet
Production	Charles

σ_{DeptName} (Employee) U σ_{DeptName} (Dept)

Dept Name
Finance
Sales
Production

Figure 5.11 : result of σ_{DeptName} (Employee) U σ_{DeptName} (Dept)

The union builds a relation consisting of all tuples appearing in either or both of two specified relations. If we have we relations then union operation is as follows.



R

A	1
B	2
D	3
E	4
F	5

S

A	1
C	2
D	3
E	4

R UNION S

A	1
B	2
C	2
D	3
E	5
F	4
E	4

Figure 5.12 UNION operation

Example of UNION

UNION of R and S

The union of two relations is a relation that includes all the tuples that are either in R or in S or in both R and S. Duplicate tuples are eliminated.

(b) Set Intersect

Notation: $r \cap s$

Defined as:

Note: $r \cap s = r - (r - s)$

DeptName
Finance
Sales

$\pi_{DeptName}$ **(Employee)** $\cap$ $\pi_{DeptName}$ (Dept**)**

R

A	1
B	2
D	3
F	4
E	5

R INTERSECTION S

A	1
D	3

S

A	1
C	2
D	3
E	4

Figure 5.13 INTERSECT ($\cap$)operation

Set intersection , the two relations involved must be union-compatible—that is, the two relations must have the same set of attributes. Because set intersection can be defined in terms of set difference, the two relations involved in set intersection must also be union-compatible.Intersection of R and S the intersection of R and S is a relation that includes all tuples that are both in R and S.

(c) Set Difference Operation :

Set difference denoted by _ allows us to find out the tuple that are in one relation but not in another .

Notation $r - s$

Defined as : Set differences must be taken between **compatible** relations.

1. r and s must have the same arity
2. Attribute domains of r and s must be compatible

$\pi_{DeptName}$ **(Dept) - $\pi_{DeptName}$ (Employee)**

result of this operation is

DeptName
Production

R

A	1
B	2
D	3
F	4
E	5

R DIFFERENCE S

B	2
F	4
E	5

S

A	1
C	2
D	3
E	4

S DIFFERENCE R

C	2
F	4

Figure 5.14 : DIFFERENCE operation

Difference of R and S

The difference of R and S is the relation that contains all the tuples that are in R but that are not in S.

6. Cartesian Product Cross Product :

The Cartesian product is defined differently from the one in set theory in the sense that tuples are considered to be 'shallow' for the purposes of the operation. That is, the Cartesian product of an n-tuple by an m-tuple has the 2-tuple "flattened" into an n+m-tuple. In set theory, the Cartesian product is a set of 2-tuples. More formally, R × S is defined as follows:

$R \times S = \{(r_1, r_2, ..., r_n, s_1, s_2, ..., s_m) \mid (r_1, r_2, ..., r_n) \text{ U } R, (s_1, s_2, ..., s_m) \text{ U } S\}$

Like the Cartesian product, the cardinality of the result is the product of the cardinalities of its factors, i.e., |R × S| = |R| × |S|. In addition, for the Cartesian product to be defined, the two relations involved must have disjoint headers—that is, they must not have a common attribute name.

The Cartesian Product is also an operator which works on two sets. It is sometimes called the CROSS PRODUCT or CROSS JOIN.

It combines the tuples of one relation with all the tuples of the other relation.

Notation r × s

R

A	1
B	2
D	3
F	4
E	5

S

A	1
C	2
D	3
E	4

R CROSS S

A	1	A	1
A	1	C	2
A	1	D	3
A	1	E	4
B	2	A	1
B	2	C	2
B	2	D	3
B	2	E	4
D	3	A	1
D	3	C	2
D	3	D	3
D	3	E	4
F	4	A	1
F	4	C	2
F	4	D	3
F	4	E	4
E	5	A	1
E	5	C	2
E	5	D	3
E	5	E	4

Figure 5.15 : CROSS PRODUCT operation

7. Joins and Join-Like Operators :

(a) Natural Joins ($\bowtie$) :

Natural joins () is a binary operator that is written as (R S) where R and S are relations For an example consider the tables Employee and Dept and their natural join: This can also be used to define composition of relations. In category theory, the join is precisely the fiber product.

Dept	
DeptName	**Manager**
Finance	George
Sales	Harriet
Production	Charles

Dept	
DeptName	**Manager**
Finance	George
Sales	Harriet
Production	Charles

Employee [x] Dept			
Name	**EmpId**	**DeptName**	**Manager**
Harry	3415	Finance	George
Sally	2241	Sales	Harriet
George	3401	Finance	George
Harriet	2202	Sales	Harriet

Figure 5.16 : Natural join operation

The natural join is arguably one of the most important operators since it is the relational counterpart of logical AND. Note carefully that if the same variable appears in each of two predicates that are connected by AND, then that variable stands for the same thing and both appearances must always be substituted by the same value. In particular, natural join allows the combination of relations that are associated by a foreign key.

For example, in the above example a foreign key probably holds from **Employee.DeptName** to **Dept.DeptName** and then the natural join of Employee and Dept combines all employees with their departments. Note that this works because the foreign key holds between attributes with the same name. If this is not the case such as in the foreign key from Dept.manager to Employee.Name then we have to rename these columns before we take the natural join. Such a join is sometimes also referred to as an **equijoin** (see θ-join).

It is usually required that R and S must have at least one common attribute, but if this constraint is omitted, and R and S have no common attributes, then the natural join becomes exactly the Cartesian product.

(b) Join and Equijoin :

If we want to combine tuples from two relations where the combination condition is not simply the equality of shared attributes then it is convenient to have a more general form of join operator, which is the θ-join (or theta-join). The θ-join is a binary operator that is written as or where a and b are attribute names, is a binary relation in the set {<, d", =, >, e"}, v is a value constant, and R and S are relations. The result of this operation consists of all combinations of tuples in R and S that satisfy the relation θ. The result of the θ-join is defined only if the headers of S and R are disjoint, that is, do not contain a common attribute.The simulation of this operation in the fundamental operations is therefore as follows:

$$\text{Employee} \bowtie_{\textbf{DeptName}} \text{Dept} = \sigma_{\textbf{DeptName} = \text{Finance}}(\text{Employee} \times \text{Dept})$$

That are unique to the relation S (those that are not attributes of R). Then the left outer join can be described in terms of the natural join (and hence using basic operators) as follows:

Employee [x] Dept			
Name	**EmpId**	**DeptName**	**Manager**
Harry	3415	Finance	George
George	3401	Finance	George

Let $r_1, r_2, ..., r_n$ be the attributes of the relation R and let {(ω, ..., ω)} be the singleton relation on the attributes

Figure 5.17 : result of Employee $_{\textbf{DeptName}}$ Dept $= \sigma_{\textbf{DeptName} = \text{Finance}}$(Employee × Dept)

In case the operator θ is the equality operator (=) then this join is also called an **equijoin**.

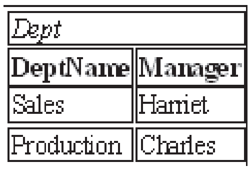
$R \ _\theta S = \sigma_\theta(R \times S)$

(c) Left Outer Join (⟕):

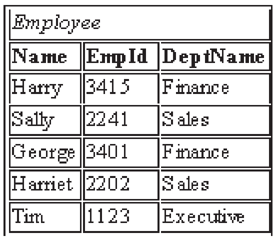
The left outer join is written as R S where R and S are relations. The result of the left outer join is the set of all combinations of tuples in R and S that are equal on their common attribute names, in addition (loosely speaking) to tuples in R that have no matching tuples in S.

For an example consider the tables Employee and Dept and their left outer join:

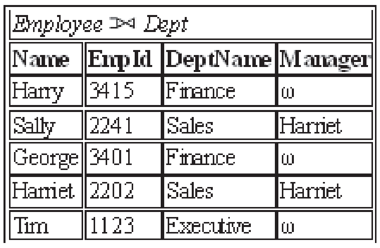
In the resulting relation, tuples in S which have no common values in common attribute names with tuples in R take a **null** value, **ω**.



Dept	
DeptName	Manager
Sales	Harriet
Production	Charles



Employee		
Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales
Tim	1123	Executive



Employee ⟕ Dept			
Name	EmpId	DeptName	Manager
Harry	3415	Finance	ω
Sally	2241	Sales	Harriet
George	3401	Finance	ω
Harriet	2202	Sales	Harriet
Tim	1123	Executive	ω

Figure 5.18 Result of left outer join

Since there are no tuples in Dept with a DeptName of Finance or Executive, ωs occur in the resulting relation where tuples in DeptName have tuples of Finance or Executive.

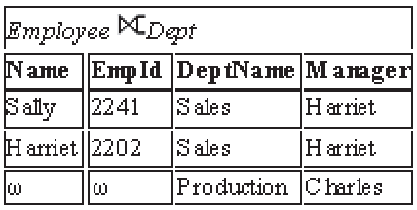
Let $r_1, r_2, ..., r_n$ be the attributes of the relation R and let {(ω, ..., ω)} be the singleton relation on the attributes that are unique to the relation S (those that are not attributes of R). Then the left outer join can be described in terms of the natural join (and hence using basic operators) as follows:

$$(R \bowtie S) \cup ((R - \pi_{r_1, r_2, \dots, r_n}(R \bowtie S)) \times \{(\omega, \dots \omega)\})$$

(d) Right Outer Join (⟖)

The right outer join behaves almost identically to the left outer join, but the roles of the tables are switched.The right outer join of relations R and S is written as R S. The result of the right outer join is the set of all combinations of tuples in R and S that are equal on their common attribute names, in addition to tuples in S that have no matching tuples in R.

For example consider the tables Employee and Dept and their right outer join:



Employee ⟖ Dept			
Name	**EmpId**	**DeptName**	**Manager**
Sally	2241	Sales	Harriet
Harriet	2202	Sales	Harriet
ω	ω	Production	Charles

Figure 5.19 : Result of right outer join

In the resulting relation, tuples in R which have no common values in common attribute names with tuples in S take a null value, ω.

Since there are no tuples in Employee with a DeptName of Production, ωs occur in the Name attribute of the resulting relation where tuples in DeptName had tuples of Production.

Let $s_1, s_2, ..., s_n$ be the attributes of the relation S and let {(ω, ..., ω)} be the singleton relation on the attributes that are unique to the relation R (those that are not attributes of S). Then, as with the left outer join, the right outer join can be simulated using the natural join as follows:

$$(R \bowtie S) \cup (\{(\omega, \ldots, \omega)\} \times (S - \pi_{s_1, s_2, \ldots, s_n}(R \bowtie S)))$$

(e) Full Outer Join (⟗)

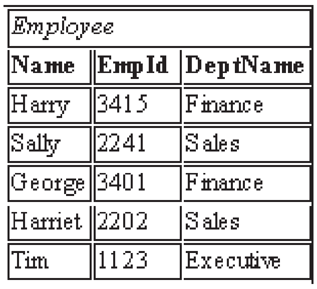
The **outer join** or **full outer join** in effect combines the results of the left and right outer joins.

The full outer join is written as R S where R and S are relations. The result of the full outer join is the set of all combinations of tuples in R and S that are equal on their common attribute names, in addition to tuples in S that have no matching tuples in R and tuples in R that have no matching tuples in S in their common attribute names.

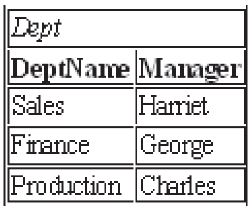
The result of a join consists of tuples formed by combining matching tuples in the two operands, an outer join contains those tuples and additionally some tuples formed by extending an unmatched tuple in one of the operands by "fill" values for each of the attributes of the other operand.

In the resulting relation, tuples in R which have no common values in common attribute names with tuples in S take a null value, ω. Tuples in S which have no common values in common attribute names with tuples in R also take a null value, ω.

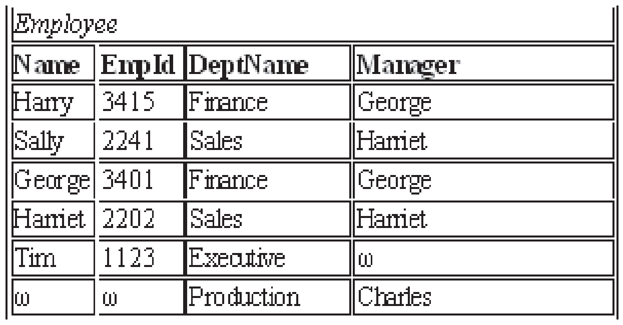
For an example consider the tables Employee and Dept and their full outer join :



Employee		
Name	**EmpId**	**DeptName**
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales
Tim	1123	Executive



Dept	
DeptName	**Manager**
Sales	Harriet
Finance	George
Production	Charles



Employee			
Name	**EmpId**	**DeptName**	**Manager**
Harry	3415	Finance	George
Sally	2241	Sales	Harriet
George	3401	Finance	George
Harriet	2202	Sales	Harriet
Tim	1123	Executive	ω
ω	ω	Production	Charles

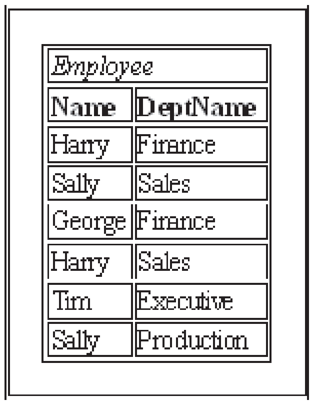
Figure 5.20 : Result of full outer join

The full outer join can be simulated using the left and right outer joins (and hence the natural join and set union) as follows:

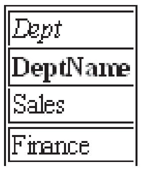
$R ⟗ S = (R \quad S) \quad (R \quad S)$

8. Division (÷)

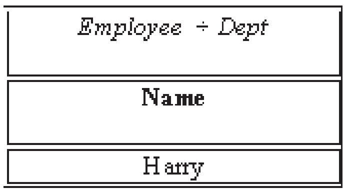
The division is a binary operation that is written as R ÷ S. The result consists of the restrictions of tuples in R to the attribute names unique to R, i.e., in the header of R but not in the header of S, for which it holds that all their combinations with tuples in S are present in R. For an example see the tables



Employee	
Name	**DeptName**
Harry	Finance
Sally	Sales
George	Finance
Harry	Sales
Tim	Executive
Sally	Production



Dept
DeptName
Sales
Finance



Employee ÷ Dept
Name
Harry

Figure 5.21 : Result of Division operation

More formally the semantics of the division is defined as follows:

Let r(R) and s(S) be given relations, with $S \subseteq R$:

$$r \div s = \pi_{R-S}^{-}(r) - \pi_{R-S}^{-}(\pi_{R-S}^{-}(r) \times s) - \pi_{R-S^{-},S}(r))$$

5.5 Extended Relational Algebra Operations

Generalized Projection :

It extends the projection operation by allowing arithmetic functions to be used in projection list.

$\mathbf{\Pi_{F1,F2 \ldots Fn}(E)}$

Where E: relational algebra expression Fi: arithmetic expression General format used for grouping :

$G_1, G_2, \ldots G_n \;\; {}_{F^-1\,A1,\,F2\,A2,\,\ldots\,Fm\,Am}$ (E) ,Where E is the relational algebra expression $G_1, G_2, \ldots G_n$ are a list of attribute on which to groupFi are aggregate functions and Ai are attribute names

Example :

Table "Credit-info" :-

Customer-name	Limit	Credit_Balance
Abc	2000	500
Xyz	500	250
Pqr	700	100
mno	1500	1000

Figure 5.22 : "Credit-info" relation

Find how much money a person can spend. result of this quary is as-

$\Pi_{\text{Customer-name,Limit, Credit\_balance}}$**(Credit-info)**

Aggregate Function :

It takes a collection of values and returns a single value as a result. The aggregate functions are sum () for summation, max () for fiding maximum, min () for minimum, avg () for finding average value, count () for finding count,

Table "stud" :-

Student	Marks	Address
Abc	20	Garia
Xyz	50	|Behala
Pqr	70	Bajaj nagar
Mno	60	Malivya nagar

Figure 5.23 : "stud" relation

So the Aggregate Functions are use as follows : -

SUM () : The sum() function find the total of the attribut on which we apply .

To find total of the Marks attribute of Record

$G_{\text{sum (Marks)}}$ **(stud)**

AVG (): The avg() function find the average of the attribute on which we apply .

To find average of the Marks attribute of Record

$G_{\text{average (Marks)}}$**(stud)**

MIN (): The MIN () function find the minimum of of the attribute on which we apply.

To find the minimum of the Marks attribute of Record.

$G_{\text{min (Marks)}}$ **(stud)**

MAX(): The MAX () function find the maximum of the attribute on which we apply.

To find the maximum of the Marks attribute of Record.

$G_{\text{max (Marks)}}$ **(stud)**

COUNT(): The COUNT () function find the count of the attribute on which we apply.

To find the number of distinct values of Address attribute of Record

$G_{\text{count distinct (Address)}}$ (stud)

Let's assume that we have a table named Account with three columns, name Account_Number, Branch_Name and Balance.

We wish to find the maximum balance of each branch. **Code: $\Pi_{\text{Branch\_Name}} G_{\text{Max(Balance)}}$(Account).**

To find the highest balance of all accounts regardless of branch.

Code : $G_{\text{Max(Balance)}}$(Account).

Limitations Of Relational Algebra :

Although relational algebra seems powerful enough for most practical purposes, there are some simple and natural operators on relations, which cannot be expressed by relational algebra. The transitive closure of a binary relation is one of them.

5.6 Modification of the Database

There are certain operations for database modification .we express data modification by assignment operation.

Insert :

To insert data information into a relation.

$r \leftarrow r \cup E$

where r is a relation and E is a relational algebra expression .

If we want insert the information about Smith with his new ecode number '017' and address as abc.

Relational algebraic expression for insertion:

0mployee ← Employee ∪ {("07 ", "Smith ", "abc")}

Insertion of loan information in loan relation then the expression is as follows:

loan ← loan ∪ {("L-7", "SBI")}

Delete :

To delete data information into a relation.

$r \leftarrow r - E$

where r is a relation and E is a relational algebra expression .

Write a relational algebraic expression to delete employee who live in city Kota.

Employee ← Employee - {($\sigma_{\text{city='Kota'}}$(Employee))}

Update :

To change a value in a tuple without changing all values in the tuple. General formats for updating is :

Generalized projection operator can be used. ie.,

$r \leftarrow \pi_{F1\ F2\ \ldots\ Fn}^{-}(r)$

where r is the relational algebra expression.

Fi is either ith attribute of r to be updated

Example expression for illustration to increase balance with amount 100.

account ← $\pi_{\text{acc\_no, branch\_name, balance + 100}}^{-}$(account)

If we want to select only some tuples and update them, we can use the following expression.ie.,

$r \leftarrow_{F1\ F2 \ldots Fn}{}^{-}(\sigma_P(r)) \cup (r - \sigma_P(r))$

where

r is the relational algebra expression.

Fi is either ith attribute of r to be updated

P denotes the selection condition

Write a relational algebraic expression to update by adding the balances more than 20000 with Rs.60/- interest and otherwise Rs. 50/- interest.

Relational algebraic expression for updating:

$account \leftarrow \pi_{branch\_name,\ acc\_no,\ balance \leftarrow balance + 60}(\sigma_{balance > 20000}(account)) \cup \pi_{branch\_name,\ acc\_no,\ balance \leftarrow balance + 50}(\sigma_{balance \leq 20000}(account))$

5.7 Views

Any relation that is not a part of the logical model but is made visible to the user as a virtual (imaginary) relation is called a view. So a view is a virtual table.

Statement for creating view is as follows:

Create view view_name **as** <relational algebra expression>

Example : If a person wants to see a relation consisting of the employees with city 'jaipur' weite expression as follows:

Create view emp as

$(\sigma_{city = \text{“Jaipur”}}(Employee))$

Here employee is a relation and emp is a view name.

Materialized views :

Definitions :

Certain database systems allow view relations to be stored, but they make sure that if the actual relations used in the view definition change then the view is kept up to date. Such views are called materialized views.

The process of keeping views up to date is called view maintenance.

Use of materialized views :

If the views are used frequently then materialized views are used. But benefits of materialization must be weighed against the storage cost and the added overhead of updates.

The following Example illustration updation cannot be perform Using views -

To create a view called loan_branch to display the loan relation without the amount information.

Create view loan_branch as $\pi_{loan\_no,\ branch\_name}(loan)$

Now if insertion of loan information is done to the view, then the expression is as follows:

loan_branch loan_branch $\cup$ {(“L-37”, “Perryridge”)}

But while inserting into a view, it will directly insert the values into the loan relation and it will be reflected back into the view by the view definition. But we need amount information to be added to the loan relation since amount is set as NOT NULL constraint.

So, two chances will occur in this situation:

- Reject the insertion to the view and return an error message to the user.

- Insert a tuple with values as {("L-37", Peeryridge", null)} by dropping the constraint in the amount attribute of loan relation.

5.8 Tuple Relational Calculus

A nonprocedural query language. It describes information without giving specific procedure for obtaining that information .A query or expression can be expressed in tuple relational calculus as

{ t | P(t)}

which means the set of all tuples 't' such that predicate P is true for 't'.

Notations used :

- $t[A] \rightarrow$ the value of tuple 't' on attribute, A
- $t \in r \rightarrow$ tuple 't' is in relation 'r'
- $\exists \rightarrow$ there exists

  Definition for 'there exists' ($\exists$):

  $\exists\, t \in r(Q(t))$

which means there exists a tuple 't' in relation 'r' such that predicate Q(t) is true.

- $\forall$ for all

Definition for 'for all' $\forall$:

$\forall \quad t \in r(Q(t))$

which means Q(t) is true for all tuples 't' in relation 'r'.

- $\Rightarrow \rightarrow$ Implication

Definition for Implication ($\Rightarrow$):

P$\Rightarrow$Q means if P is true then Q must be true.

t | t denotes that tuple t is in relation and P is a formula similar to that of the predicate calculation. Set of comparison operators: (e.g., <, =, >) and Set of connectives: and (Λ), or (V), not (Ø) .

The equivalence rules of tuple relational calculus.

- $P_1 \wedge P_2$ is equivalent to -_$\neg(\neg P_1 \vee \neg P_2)$
- $\forall \quad t \in r(P(t))$ is equivalent to $\neg\exists\, t \in r(\neg P(t))$
- $P_1 \Rightarrow P_{2-}$ is equivalent to $\neg P_1 \vee P_2$

Division Operation in TRC :

$R \div S = \{ t[a_1,...,a_n] : t \in R \wedge \forall s\ S ((t[a_1,...,a_n]\ s) R) \}$

where $\{a_1,...,a_n\}$ is the set of attribute names unique to R and $t[a_1,...,a_n]$ is the restriction of t to this set. It is usually required that the attribute names in the header of S are a subset of those of R because otherwise the result of the operation will always be empty.

Natural Join Operation Statement in Relational Algebra :

The result of the natural join is the set of all combinations of tuples in R and S that are equal on their common attribute names.

Self learning Exercise :

2. Write queries in tuple relational calculus in following form using following schema -

branch (branch_name, branch_city, assets)

customer (customer_name, customer_street, customer_city) account (account_number, branch_name, balance)

loan (loan_number, branch_name, amount)

depositor (customer_name, account_number)

borrower (customer_name, loan_number)

(a) Find the loan number for each loan of an amount greater than \$1200.

(b) Find the names of all customers having a loan, an account, or both at the bank

(c) Find the names of all customers having a loan at the Perryridge branch.

Safety of Expressions:

It is possible to write tuple calculus expressions that generate infinite relations. For example, $\{ t \mid \neg t \in r \}$ results in an infinite relation if the domain of any attribute of relation r is infinite. To guard against the problem, we restrict the set of allowable expressions to safe expressions.

An expression $\{t \mid P (t)\}$ in the tuple relational calculus is safe if every component of t appears in one of the relations, tuples, or constants that appear in P or we can say

Safety of expression in tuple relational calculus :

Query or expression $\{t \mid P (t)\}$ is safe if all values that appear in the result are values from dom (p), where dom (p) is domain of P. Else it is unsafe.

NOTE : This is more than just a syntax condition.

Example : $\{ t \mid \neg (t \in Emp) \}$ is not safe , it defines an infinite set with attribute values that do not appear in any relation or tuples or constants in P.

5.9 Domain Relational Calculus

A nonprocedural query language equivalent in power to the tuple relational calculus. Each query is an expression of the form :

$\{ < x1, x2, \ldots, xn > \mid P (x1, x2, \ldots, xn)\}$

where $x_1, x_n, \ldots x_n$ represents domain variables.

P represents a formula composed of atoms or domain

There are some question in DRC for which we can know how to write statement in DRC.

1. Suppose if we find the loan_number, branch_name, and amount for loans of over \$1200 we write the following expression-

   $\{< l, b, a > \mid < l, b, a > \text{ loan } \Lambda \; a > 1200\}$

Note : There are three attrinbute in loan relation l for loan_no,b for branch_name ,a for amount.

2. Suppose if we find the names of all customers who have a loan of over \$1200

   $\{< c > \mid \exists \, l, b, a \, (< c, l > \in \text{ borrower } \Lambda < l, b, a > \in \text{ loan } \Lambda \; a > 1200)\}$

Note : There are two attrinbute in borrower relation c for customer name l for loan_no, and there are three attrinbute in loan relation l for loan_no,b for branch_name ,a for amount.

3. Suppose if we find the names of all customers who have a loan from the "SBI" branch and the loan amount :

   $\{< c, a > \mid \; l \, (< c, l > \in \text{ borrower } \exists \, b \, (< l, b, a > \in \text{ loan } \Lambda \; b = \text{“SBI”}))\}$

   OR

   $\{< c, a > \mid \exists \, l \, (< c, l > \in \text{ borrower } \Lambda < l, \text{“SBI”}, a \in \text{ loan})\}$

4. Suppose if we find the loan_number, branch_name, and amount for loans of over \$1200

   $\{< l, b, a > \mid < l, b, a >$ loan $\Lambda\ a > 1200\}$

5. Suppose if we find the names of all customers having a loan, an account, or both at the "SBI" branch:

   $\{< c > \mid\ 1 (< c, l > \in$ borrower

   $\Lambda\ \exists$ b,a $(< l, b, a > \in$ loan Λ b = "SBI")) $\Lambda\exists$ a $(< c, a > \in$ depositor

   $\Lambda\exists$ b,n $(< a, b, n > \in$ account Λ b = "SBI"))}

Note : Account relation have three attributes are account_number, branch_name, balance, depositor have customer_name,account_number, borrower have customer_name,loanno.

6. Suppose if we find the names of all customers who have an account at all branches located in Jaipur:

   $\{< c > \mid \exists$ s,n $(< c, s, n >$ customer) Λ

   x,y,z $(< x, y, z > \in$ branch Λ y = "Jaipur")

   $\exists$ a,b $(< x, y, z > \in$ account $\Lambda\ < c,a > \in$ depositor)}

Note : Branch relation have branchname,branch city,assets,customer relation have customername, account_number, account relation have three attributes are account_number, branch_name, balance and depositor have customer_name,account_number

Safety of Expressions :

The expression:

$\{ < x1, x2, \ldots, xn > \mid P (x1, x2, \ldots, xn)\}$

is safe if all of the following hold :

1. All values that appear in tuples of the expression are values from dom (P) (that is, the values appear either in P or in a tuple of a relation mentioned in P).
2. For every "there exists" subformula of the form \$ x (P1(x)), the subformula is true if and only if there is a value of x in dom (P1)such that P1(x) is true.
3. For every "for all" subformula of the form "x (P1 (x)), the subformula is true if and only if P1(x) is true for all values x from dom (P1).

Self-learning Exercise :

3. Write an expression to find all customers who have an account, a loan or both at the Perryridge branch in domain relational calculus.

5.10 Codd'S Rules

The consistency of a relational database is enforced, not by rules built into the applications that use it, but rather by consteraints declared as part of the logical schema and enforced by the DBMS for all applications. In practice, several useful shorthands are expected to be available, of which the most important are candidate key (really, superkey) and foreign key constraints. Dr. E. F. Codd's 12 rules.

For defining a fully relational database

Note that based on these rules there is no fully relational database management system available today. In particular, rules 6, 9, 10, 11 and 12 are difficult to satisfy.

1. **Foundation Rule :**

   A relational database management system must manage its stored data using only its relational capabilities.

2. **Information Rule :**

All information in the database should be represented in one and only one way - as values in a table.

3. **Guaranteed Access Rule :**

Each and every datum (atomic value) is guaranteed to be logically accessible by resorting to a combination of table name, primary key value and column name.

4. **Systematic Treatment of Null Values :**

Null values (distinct from empty character string or a string of blank characters and distinct from zero or any other number) are supported in the fully relational DBMS for representing missing information in a systematic way, independent of data type.

5. **Dynamic On-line Catalog Based on the Relational Model :**

The database description is represented at the logical level in the same way as ordinary data, so authorized users can apply the same relational language to its interrogation as they apply to regular data.

6. **Comprehensive Data Sublanguage Rule :**

A relational system may support several languages and various modes of terminal use. However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and whose ability to support all of the following is comprehensible:

(a) data definition

(b) view definition

(c) data manipulation (interactive and by program)

(d) integrity constraints

(e) authorization

(f) transaction boundaries (begin, commit, and rollback).

7. **View Updating Rule :**

All views that are theoretically updateable are also updateable by the system.

8. **High-level Insert, Update, and Delete :**

The capability of handling a base relation or a derived relation as a single operand applies nor only to the retrieval of data but also to the insertion, update, and deletion of data.

9. **Physical Data Independence :**

Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representation or access methods.

10. **Logical Data Independence :**

Application programs and terminal activities remain logically unimpaired when information preserving changes of any kind that theoretically permit unimpairment are made to the base tables.

11. **Integrity Independence :**

Integrity constraints specific to a particular relational database must be definable in the relational data sublanguage and storable in the catalog, not in the application programs.

12. **Distribution Independence :**

The data manipulation sublanguage of a relational DBMS must enable application programs and terminal activities to remain logically unimpaired whether and whenever data are physically centralized or distributed.

13. **Nonsubversion Rule :**

If a relational system has or supports a low-level (single-record-at-a-time) language, that low-level language cannot be used to subvert or bypass the integrity rules or constraints expressed in the higher-level (multiple-records-at-a-time) relational language.

5.11 Summary

The relation algebra is a procedural language define set of algebraic operations that operate on tables,and output tables are their results .there are two types of operation Selection,Projection and Rename operation are called unary operations,because they operate on one relation.The other operation are called binary operation because They operate on more then one relation

Database can be modify by insert ,delete and update commands we use assignment operator to express these modification.

Tuple relational and domain relational calculus are non procedural query language .

Any relation that is not a part of the logical model but is made visible to the user as a virtual (imaginary) relation is called a view. So a view is a virtual table.updatable views are those which are define on a single relation because through the view we can modify the base table on which we can define a view.

5.12 Aanswer to Self Learning Exercise

Ans. 1 A relation Schema denoted by $R(A_1, A_2, \ldots, A_n)$ is made up of the relation name R and the list of attributes Ai that it contains. A relation is defined as a set of tuples. Let r be the relation which contains set tuples $(t_1, t_2, t_3, ..., t_n)$. Each tuple is an ordered list of n-values $t=(v_1,v_2, ..., v_n)$.

Ans. 2 (a) {t | s ∈ loan (t [loan_number] = s [loan_number] Λ s [amount] > 1200)}

Notice that a relation on schema [loan_number] is implicitly defined by the query.

(b) {t | s ∈borrower (t [customer_name] = s [customer_name])

Λ ∃ u ∈ depositor (t [customer_name] = u [customer_name]) }

(c) {t | ∃ s ∈ borrower (t [customer_name] = s [customer_name]

Λ u [loan_number] = s [loan_number]) Λ∃u ∈ loan (u [branch_name] =

"Perryridge")}

Ans. 3 {<c> | ∃ l (<c,l> ∈ borrower ∧ ∃ b,a (<l,b,a> ∈ loan ∧ b = "Perryridge")) ∨ ∃ a (<c,a> ∈ depositor ∧ ∃ b,bal (<a,b,bal> ∈ account ∧ b = "Perryridge"))}

5.13 Self Assessment Questions

1. Explain about the relational algebra?
2. Explain the various basic relational algebra operations in detail.
3. Write a note on relational calculus.
4. Consider the following tables:

   Employee (Emp_no, Name, Emp_city)

   Company (Emp_no, Company_name, Salary)

   writes the following queries in relational algebra

   (i) Write a query display Employee name and company name.

   (ii) Write a query to display employee name, employee city ,company name and salary of all the employees whose salary >10000

(iii) Write a query to display all the employees working in 'XYZ' company.

5. What is meant by a unary and binary operation? What are they?
6. How 'Natural –Join' operation is performed?
7. What is a join? What are the benefits of joins?
8. What is equijoin and non-equijoin?
9. List the operations of Relational algebra.
10. Differentiate between Cartesian product and natural join operations used in relationa
11. Define the following terms :

    (a) Tuple (b) Domain

    (c) Relation (d) Entity

    (e) Regular entities
12. Discuss the various type of join operations ? Why are these join required.
13. List the operations of relational algebra and purpose of each.
14. What is the difference between tuple relational calculus and domain relational calculus?
15. SQL is called as non-procedural language. Explain?
16. What you mean by attribute? Explain various types of attributes.
17. How does Tuple-oriented relational calculus differ from domain-oriented relational calculus.

□□□

Unit - 6 : RDBMS

Structure of the Unit

6.0 Objective

The objective of this module to understand about Relational database management system. A type of database in which the data can be spread across several tables that are related together. Data in related tables are associated by shared attributes. Any data element can be found in the database through the name of the table, the attribute (column) name, and the attribute values that uniquely identify each row. After learning this module they differentiated between DBMS and RDBMS. They understand that a RDBMS requires few assumptions about how data is related or how it will be extracted from the database. As a result, the data can be arranged in different combinations.

6.1 Introduction

A relational DBMS is special system software that is used to manage the organization, storage, access, security and integrity of data. This specialized software allows application systems to focus on the user interface, data validation and screen navigation. When there is a need to add, modify, delete or display data, the application system simply makes a "call" to the RDBMS. Although there are many different types of database management systems, relational databases are by far the most common. Other types include hierarchical databases and network databases.

The database management systems have been around since the 1960s; relational databases didn't become popular until the 1980s .A relational DBMS stores information in a set of "tables", each of which has a unique identifier or "primary key". The tables are then related to one another using "foreign keys". A foreign key is simply the primary key in a different table. Diagrammatically, a foreign key is depicted as a line with an arrow at one end.

In the Figure 6.1 , "Customer ID" is the primary key (PK) in one table and the foreign key (FK) in another. The arrow represents a one-to-many relationship between the two tables. The relationship indicates that one customer can have one or more orders. One and only one customer, however, can initiate a given order. By storing data in a RDBMS, undesirable data redundancy can be avoided. This not only makes data management easier, but it also makes for a flexible database that can respond to changing requirements.

Customer Customer ID (PK) Last Name First NamePhone Number

Order Order Number (PK)Customer ID (FK) Order DateOrder Fill Date

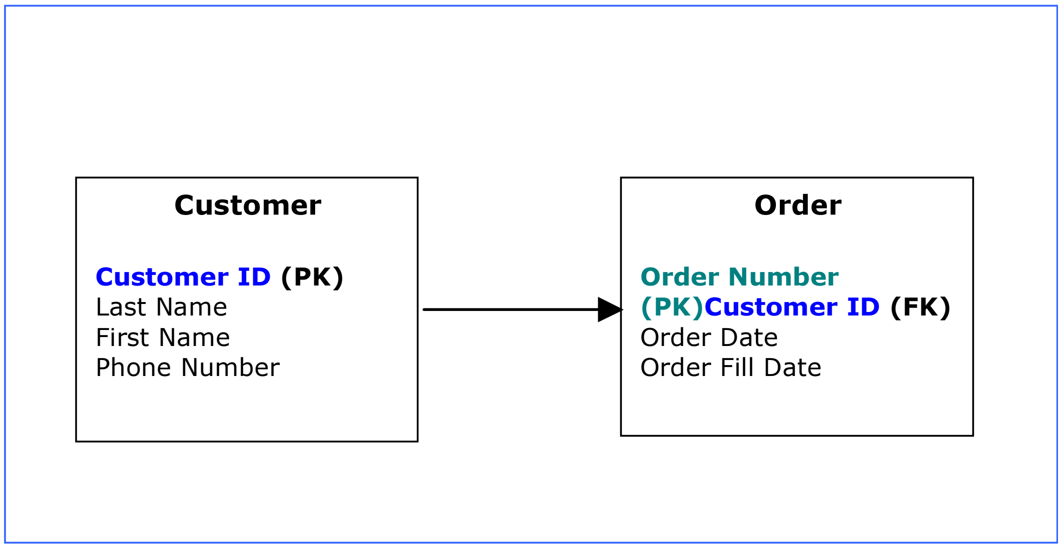


Figure 6.1 : Example of a Relational Database Management System (RDBMS)

Vendors	RDBMS
Computer Associates	INGRES
IBM	DB2
INFORMIX Software	INFORMIX
Oracle Corporation	Oracle
Microsoft Corporation	MS Access
Microsoft Corporation	SQL Server
MySQL AB	MySQL
NCR Teradata	
PostgreSQL Dvlp Grp	PostgreSQL
Sybase	Sybase 11

Table 6.1

In table 6.1 show many different vendors that currently produce relational database management systems (RDBMS). Relational databases vary significantly in their capabilities and in costs. Some products are proprietary while others are open source. The leading vendors of RDBMS are listed below:

Self-Learning Exercise :

1. What is difference between DBMS and RDBMS?

6.2 Integrity Constraints

Integrity constraints are use to prevent invalid data entry into the base tables of the database. We can define integrity constraints to enforce the business rules you want to associate with the information in a database. If any of the results of a DML statement execution violate an integrity constraint, then database systems back the statement and return an error.

For example, assume that you define an integrity constraint for the salary column of the employee's table. This integrity constraint enforces the rule that no row in this table can contain a numeric value greater than 10,000 in this column. If an INSERT or UPDATE statement attempts to violate this integrity constraint, then Oracle rolls back the statement and returns an information error message.

The integrity constraints implemented in Oracle fully comply with ANSI X3.135-1989 and ISO 9075-1989 standards.

Use of integrity constraints to enforce the business rules associated with your database and prevents the entry of invalid information into tables.

It is important that data adhere to a predefined set of rules, as determined by the database administrator or application developer. As an example of data integrity, consider the tables employees and departments and the business rules for the information in each of the tables, as illustrated in Figure 6.2.

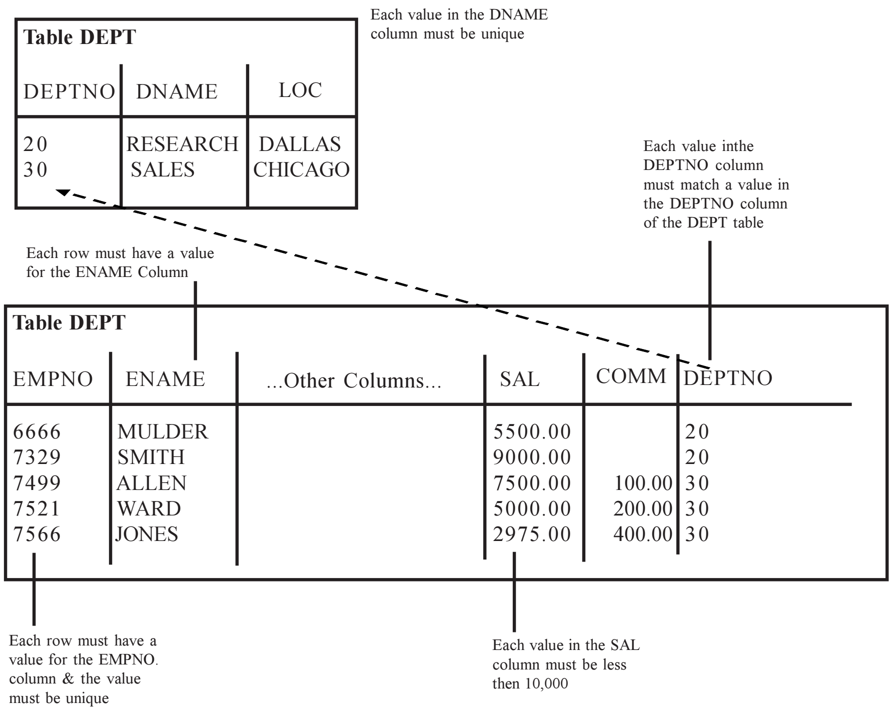

Each value in the DNAME column must be unique
Each value inthe DEPTNO column must match a value in the DEPTNO column of the DEPT table
Each row must have a value for the ENAME Column
Each row must have a value for the EMPNO. column & the value must be unique
Each value in the SAL column must be less then 10,000

Table DEPT

DEPTNO	DNAME	LOC
20	RESEARCH	DALLAS
30	SALES	CHICAGO

Table DEPT

EMPNO	ENAME	...Other Columns...	SAL	COMM	DEPTNO
6666	MULDER		5500.00		20
7329	SMITH		9000.00		20
7499	ALLEN		7500.00	100.00	30
7521	WARD		5000.00	200.00	30
7566	JONES		2975.00	400.00	30

Figure 6.2 : Examples of Data Integrity

Note that some columns in each table have specific rules that constrain the data contained within them.

Data Integrity :

The following categories of the data integrity exist with each RDBMS:

- **Entity Integrity :** There are no duplicate rows in a table.
- **Domain Integrity :** Enforces valid entries for a given column by restricting the type, the format, or the range of values.
- **Referential integrity :** Rows cannot be deleted, which are used by other records.
- **User-Defined Integrity :** Enforces some specific business rules that do not fall into entity, domain, or referential integrity.

Specifying Semantic Integrity Constraints SQL DDL provides three constructs for specifying semantic integrity constraints: *Checks, Assertions, and Triggers.* Check constraint specify where condition is full fill or not. assertions and triggers, is implemented in all DBMSs.

A constraint is a predicate that must be satisfied, and it can be expressed as a condition over multiple tables similarly to the way in which the WHERE-clause of a query using EXIST and NOT EXIST is expressed.

6.2.1 Check Integrity Constraints

A Check integrity constraint on a column or set of columns requires that a specified condition be true or unknown for every row of the table. If a DML statement results in the condition of the CHECK constraint evaluating to false, then the statement is rolled back.

The Check Condition CHECK constraints enable you to enforce very specific integrity rules by specifying a check condition.

SQL Check Constraint :

This constraint defines a business rule on a column. All the rows must satisfy this rule. The constraint can be applied for a single column or a group of columns.

Syntax to Define a Check Constraint :

```
[CONSTRAINT constraint_name] CHECK (condition)
```

For Example: In the employee table to select the gender of a person, the query would be like:-

Check Constraint at Column Level:

```
CREATE TABLE employee
( id number(5) ,
name char(20),
dept char(10),
age number(2),
gender char(1) CHECK (gender in ('M','F')),
salary number(10),
location char(10)
);
```

Check Constraint at Table Level :

```
CREATE TABLE employee
( id number(5) ,
name char(20),
dept char(10),
age number(2),
gender char(1),
salary number(10),
location char(10),
CONSTRAINT gender_ck CHECK (gender in ('M','F'))
);
```

6.2.2 Assertions

An assertion prohibits an action that would violate a specified constraint. However, It isevaluated every time a table involved in the constraint is modified. Assertions as global constraints.

An assertion is specified using the CREATE ASSERTION command. As an example, that the budget in SECTION cannot be less than the sum of the salaries of the librarians who work in that section.

The same constraint expressed as an ASSERTION is given below.

Syntax:

```
CREATE ASSERTION  < assertion_name>
SQL statement ;
```

Example :

```
CREATE ASSERTION  ass_budget
CHECK (NOT EXISTS (SELECT * FROM SECTION
WHERE budget < ( SELECT SUM (Salary) FROM LIBRARIAN)));
```

Here ass_budget is the name of assertion .

When an assertion is not needed any longer, it can be removed, using the DROP ASSERTION command:

```
DROP ASSERTION  < assertion_name>;
```

Example :

```
DROP ASSERTION  ass_budget;
```

6.2.3 Trigger

Although triggers are supported by several DBMSs, SQL2 has not fully defined them. Triggers are expected to be defined fully in SQL3. A trigger is a block that is associated with a database table or a view. It is executed when automatically fired by a DML statement.

A trigger is specified using the DEFINE TRIGGER command. Or CREATE TRIGGER, and it consists of two parts: a *condition* and an *action*. The action is executed when the condition becomes true. The condition specifies the *time*, and the *events* that will trigger the action. The time can specify *before* or *after* a particular event. An event could be one of the modification operations: INSERT, DELETE, or UPDATE.

The general form of DEFINE Trigger is

```
DEFINE TRIGGER
        <trigger-name>
        < time events>
ON      <list-of-tables>
WHEN <Predicate>
        <action-name>
```

An action could be, for example, a ROLLBACK, a DELETE, a transaction with multiple updates, a stored procedure (as in our example below), and so on.

Let us express the same constraint we used above as a trigger: the salary of a librarian must not be greater than the salary of his or her head librarian.

```
DEFINE TRIGGER  librarian_salary_
after UPDATE of Salary
ON LIBRARIAN
WHEN (EXISTS (SELECT * FROM LIBRARIAN L, LIBRARIAN H, SECTION S
WHERE  L.Salary > H.Salary AND L.Section = S.SectNo
AND S.HeadSSN = H.SSN and L.SSN <> H.SSN))
inform_director (L.SSN,HeadSSN);
```

We assumed that inform_director() is a procedure stored in the database and can be called by the DBMS. Its arguments are the SSNs of the librarian and the head librarian and it sends an email to the director of the library. When a trigger is not needed any longer, it can be removed using the DROP TRIGGER command:

```
DROP TRIGGER librarian_salary ;
```

In conclusion, note the difference between assert and trigger in the condition. A trigger allows an action that may violate a constraint and hence it tests for the presence of the violation. An assertion does not permit a violation and hence it typically tests for its absence.

Self-Learning Exercise:

2. Assume that table EMPLOYEE contains column SALARY. Create a trigger, SAL_ADJ, that prevents an update to an employee's salary that exceeds 20% and signals such an error.
3. Difference between Store Procedure and Trigger?

   So We can say that integrity constraints are the rules that can be applied to table columns to enforce different types of data integrity. Other integrity constraints are as follows-

6.2.4 Not Null

A null is a rule defined on a single column that allows or disallows inserts or updates of rows containing a null (the absence of a value) in that column. By default, all columns in a table allow nulls. **Null** means the absence of a value. A NOT NULL constraint requires a column of a table contain no null values. For example, you can define a NOT NULL constraint to require that a value be input in the last_name column for every row of the employees table. Figure show a NOT NULL integrity constraint.

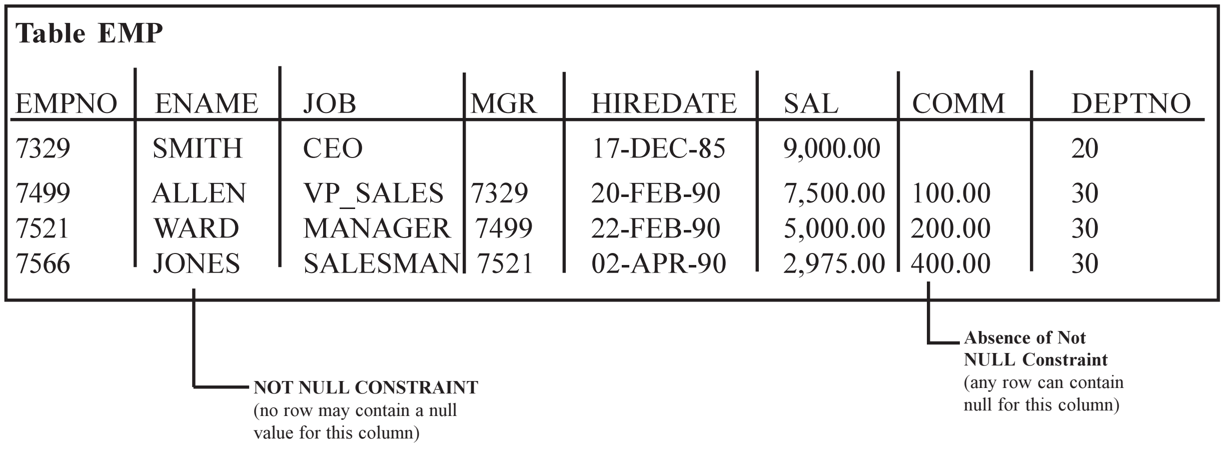


Table EMP

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7329	SMITH	CEO		17-DEC-85	9,000.00		20
7499	ALLEN	VP_SALES	7329	20-FEB-90	7,500.00	100.00	30
7521	WARD	MANAGER	7499	22-FEB-90	5,000.00	200.00	30
7566	JONES	SALESMAN	7521	02-APR-90	2,975.00	400.00	30

NOT NULL CONSTRAINT
(no row may contain a null value for this column)
Absence of Not NULL Constraint
(any row can contain null for this column)

Figure 6.3 : Not Null Integrity Constraints

SQL Statement for Not Null Constraint :

This constraint ensures all rows in the table contain a definite value for the column which specified as not null. Which means a null value is not allowed.

Constraints can be defined in two ways

(1) The constraints can be specified immediately after the column definition. This is called column level definition.

(2) The constraints can be specified after all the columns are defined. This is called table-level definition.

Syntax to Define a Not Null Constraint :

```
CREATE TABLE  table name
( column1 datatype (size) CONSTRAINT id_c NOT NULL,
column2 datatype (size),
….
);
```

For Example : To create a employee table with Null value, the query would be like

```
CREATE TABLE employee
```

```
( id number(5) CONSTRAINT id_c NOT NULL,
name char(20) CONSTRAINT nm_nn NOT NULL,
dept char(10),
age number(2),
salary number(10),
location char(10)
);
```

Here we define two constraints on two different coloum we define id attribute to not null and give constraint name id_c and second constraint is nm_nn which is define name attribute not null. That is it does not accept null value we must enter some value in these attribute.

6.2.5 Unique Key

A UNIQUE key integrity constraint requires that every value in a column or set of columns (key) be unique—that is, no two rows of a table have duplicate values in a specified column or set of columns.

The columns included in the definition of the UNIQUE key constraint are called the **unique key**. **Unique key** is often incorrectly used as a synonym for the terms **UNIQUE key constraint** or **UNIQUE index**. However, note that **key** refers only to the column or set of columns used in the definition of the integrity constraint.

If the UNIQUE key consists of more than one column, that group of columns is said to be a **composite unique key**. For example, in Figure the customer table has a UNIQUE key constraint defined on the composite unique key: the area and phone columns.

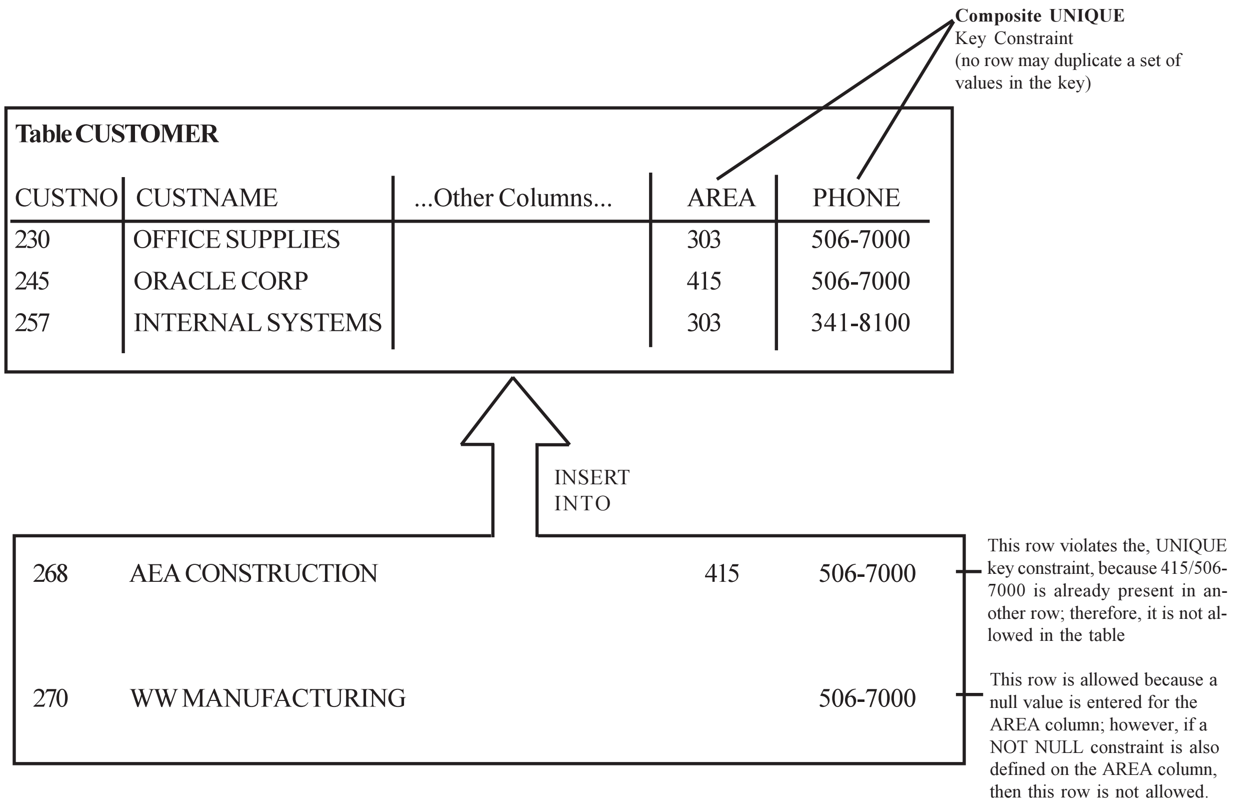

Composite UNIQUE Key Constraint (no row may duplicate a set of values in the key)
Table CUSTOMER
CUSTNO
CUSTNAME
...Other Columns...
AREA
PHONE
230
OFFICE SUPPLIES
303
506-7000
245
ORACLE CORP
415
506-7000
257
INTERNAL SYSTEMS
303
341-8100
INSERT INTO
268
AEA CONSTRUCTION
415
506-7000
270
WW MANUFACTURING
506-7000
This row violates the, UNIQUE key constraint, because 415/506-7000 is already present in another row; therefore, it is not allowed in the table
This row is allowed because a null value is entered for the AREA column; however, if a NOT NULL constraint is also defined on the AREA column, then this row is not allowed.

Figure 6.4 : A Composite Unique Key Constraint

This Unique key constraint lets you enter an area code and telephone number any number of times, but the combination of a given area code and given telephone number cannot be duplicated in the table. This eliminates unintentional duplication of a telephone number.

Combine Unique Key and Not Null Integrity Constraints :

In Figure 6.4, Unique key constraints allow the input of nulls unless you also define Not Null constraints for the same columns. In fact, any number of rows can include nulls for columns without Not Null constraints because nulls are not considered equal to anything. A null in a column (or in all columns of a composite Unique key) always satisfies a Unique key constraint. Columns with both unique keys and Not Null integrity constraints are common. This combination forces the user to enter values in the unique key and also eliminates the possibility that any new row's data will ever conflict with an existing row's data.

SQL statement for Unique Key :

This constraint ensures that a column or a group of columns in each row have a distinct value.A column(s) can have a null value but the values cannot be duplicated.

Syntax to Define a Unique Key at Column Level :

```
[CONSTRAINT constraint_name] UNIQUE
```

Syntax to Define a Unique Key at Table Level :

```
[CONSTRAINT constraint_name] UNIQUE(column_name)
```

For Example : To create an employee table with Unique key, the query would be like,

Unique Key at Column Level :

```
Synatx:
CREATE TABLE  tablename
(( column1 datatype (size) UNIQUE,
column2 datatype (size),
....
);
```

Example :

```
CREATE TABLE employee
( id number(5) ,
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10) UNIQUE
);
```

Note : We can define more than one column as a Unique.

or

```
CREATE TABLE employee
( id number(5) ,
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10) CONSTRAINT loc_un UNIQUE
);
```

Unique Key at Table Level :

Likewise, unique keys can be defined as part of the CREATE TABLE SQL statement.

```
CREATE TABLE TABLE_NAME (
id_col   NUMBER(2),
col2     VARCHAR(20),
...
CONSTRAINT key_unique UNIQUE(key_col),
...
);
CREATE TABLE employee
( id number(5) ,
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10),
CONSTRAINT loc_un UNIQUE(location)
);
```

Defining Unique Keys in Already Created Table :

The definition of unique keys is as follows .

```
ALTER TABLE <TABLE identifier>
ADD [ CONSTRAINT <CONSTRAINT identifier> ]
UNIQUE ( <COLUMN expression> {, <COLUMN expression>}... )
```

Example :

```
ALTER TABLE  employee
ADD CONSTRAINT loc_un UNIQUE (location);
```

6.2.6 Primary Key

Each table in the database can have at most one PRIMARY KEY constraint. The values in the group of one or more columns subject to this constraint constitute the unique identifier of the row. In effect, each row is named by its primary key values.

The Oracle implementation of the PRIMARY KEY integrity constraint guarantees that following are true:

- No two rows of a table have duplicate values in the specified column or set of columns.
- The primary key columns do not allow nulls. That is, a value must exist for the primary key columns in each row.

Primary Keys

The columns included in the definition of a table's PRIMARY KEY integrity constraint are called the *primary key*. Although it is not required, every table should have a primary key so that:

Each row in the table can be uniquely identified .No duplicate rows exist in the table

Figure 6.5 illustrates a Primary Key constraint in the departments table and examples of rows that violate the constraint.

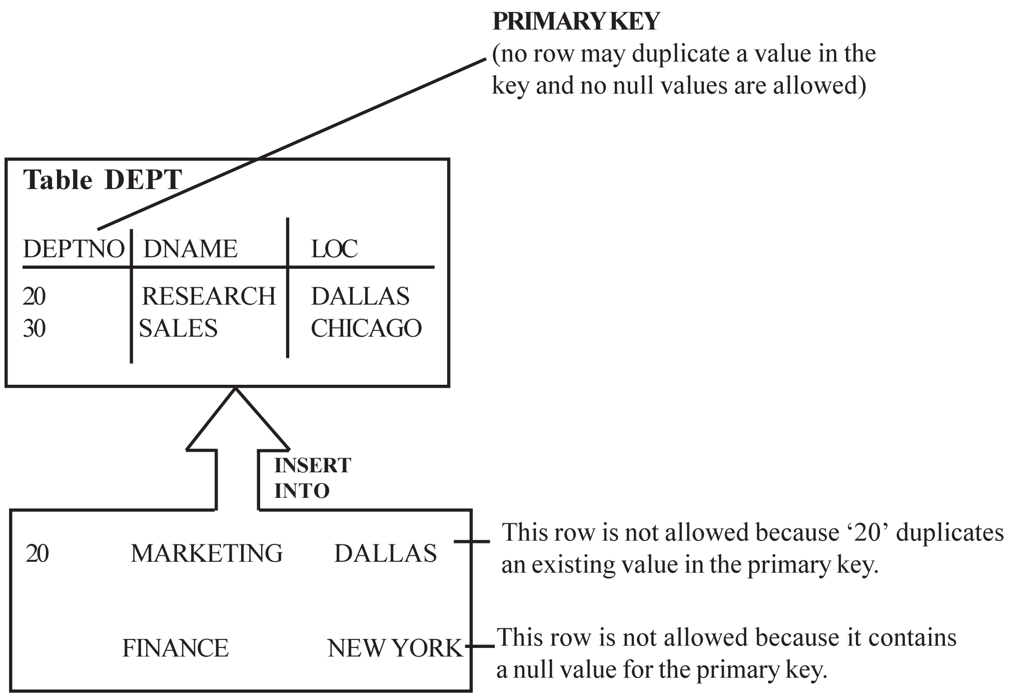

PRIMARY KEY
(no row may duplicate a value in the key and no null values are allowed)
Table DEPT
DEPTNO DNAME LOC
20 RESEARCH DALLAS
30 SALES CHICAGO
INSERT INTO
20 MARKETING DALLAS
FINANCE NEW YORK
This row is not allowed because '20' duplicates an existing value in the primary key.
This row is not allowed because it contains a null value for the primary key.

Figure 6.5 : A Primary Key Constraint

SQL Primary Key :

This constraint defines a column or combination of columns which uniquely identifies each row in the table.

Syntax to Define a Primary Key at Column Level :

column name datatype [CONSTRAINT constraint_name] PRIMARY KEY

Syntax to Define a Primary Key at Table Level :

[CONSTRAINT constraint_name] PRIMARY KEY

(column_name1,column_name2,..)

column_name1, column_name2 are the names of the columns which define the primary Key.

The syntax within the bracket i.e. [CONSTRAINT constraint_name] is optional.

For Example : To create an employee table with Primary Key constraint, the query would be like.

Primary Key at Column Level :

If the primary key consists only of a single column, the column can be marked as such using the following syntax:

```
CREATE TABLE TABLE_NAME (
id_col  NUMBER(2)  PRIMARY KEY,
col2   VARCHAR(20),
...
);
```

Example :

```
CREATE TABLE employee
( id number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10)
);
```

or

```
CREATE TABLE employee
( id number(5) CONSTRAINT emp_id_pk PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10)
);
```

Primary Key at Table Level :

The primary key can also be specified directly during table creation. In the SQL Standard, primary keys may consist of one or multiple columns. Each column participating in the primary key is implicitly defined as Not Null. Note that some DBMS require explicitly marking primary-key columns as Not Null.

Syntax for Defining Primary Key at Table Level :

```
CREATE TABLE TABLE_NAME (
id_col  NUMBER(2),
col2    CHAR(20),
...
CONSTRAINT tab_pk PRIMARY KEY(id_col),
...
);
```

Example :

```
CREATE TABLE employee
( id number(5),
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10),
CONSTRAINT emp_id_pk PRIMARY KEY (id)
);
```

The syntax to add such a constraint to an existing table is defined in SQL:2003 like this:

```
ALTER TABLE <TABLE identifier>
ADD [ CONSTRAINT <CONSTRAINT identifier> ]
PRIMARY KEY ( <COLUMN expression> {, <COLUMN expression>}... )
```

Example :

```
ALTER TABLE <TABLE identifier>
ADD  CONSTRAINT emp_id_pk
PRIMARY KEY (id);
```

Or

```
ALTER TABLE <TABLE identifier>
ADD   PRIMARY KEY (id);  //without defining as a constraint
```

Self-Learning Exercise :

4. What's the difference between a primary key and a unique key?

6.2.7 Referential Integrity

Common columns can relate different tables in a relational database, and the rules that govern the relationship of the columns must be maintained. Referential integrity rules guarantee that these relationships are preserved.

The following terms are associated with referential integrity constraints.

Term Definition :

Foreign Key : The column or set of columns included in the definition of the referential integrity constraint that reference a referenced key.

Referenced Key : The Unique Key or primary key of the same or different table that is referenced by a foreign key.

Dependent or Child Table :

The table that includes the foreign key. Therefore, it is the table that is dependent on the values present in the referenced unique or primary key.

Referenced or Parent Table :

The table that is referenced by the child table's foreign key. It is this table's referenced key that determines whether specific inserts or updates are allowed in the child table.

A referential integrity constraint requires that for each row of a table, the value in the foreign key matches a value in a parent key.

Figure 6.6 shows a foreign key defined on the deptno column of the employees table. It guarantees that every value in this column must match a value in the primary key of the departments table (also the deptno column). Therefore, no erroneous department numbers can exist in the dept no column of the employees table.

Foreign keys can be defined as multiple columns. However, a composite foreign key must

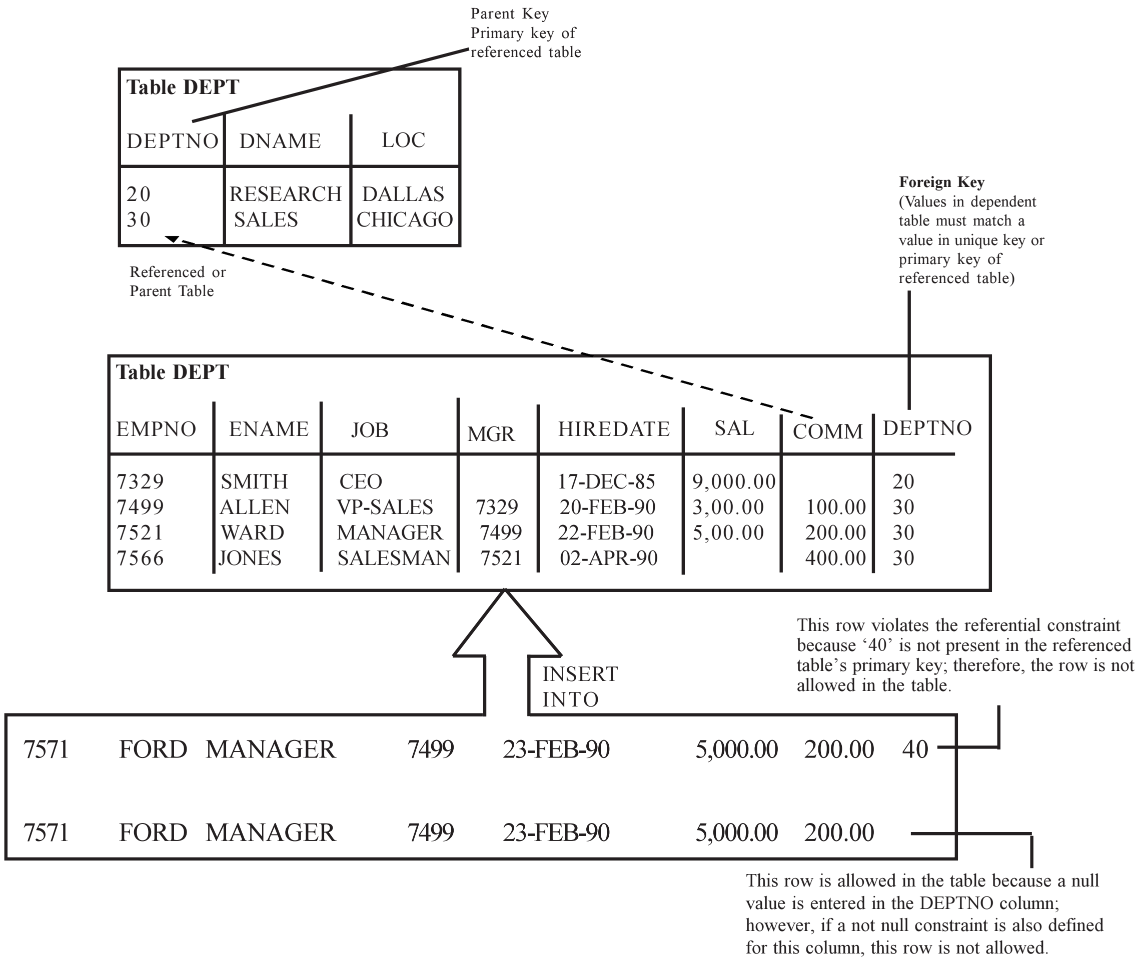

Parent Key
Primary key of
referenced table
Referenced or
Parent Table
Foreign Key
(Values in dependent
table must match a
value in unique key or
primary key of
referenced table)
INSERT
INTO
This row violates the referential constraint because '40' is not present in the referenced table's primary key; therefore, the row is not allowed in the table.
This row is allowed in the table because a null value is entered in the DEPTNO column; however, if a not null constraint is also defined for this column, this row is not allowed.

Table DEPT

DEPTNO	DNAME	LOC
20	RESEARCH	DALLAS
30	SALES	CHICAGO

Table DEPT

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7329	SMITH	CEO		17-DEC-85	9,000.00		20
7499	ALLEN	VP-SALES	7329	20-FEB-90	3,00.00	100.00	30
7521	WARD	MANAGER	7499	22-FEB-90	5,00.00	200.00	30
7566	JONES	SALESMAN	7521	02-APR-90		400.00	30

7571	FORD	MANAGER	7499	23-FEB-90	5,000.00	200.00	40
7571	FORD	MANAGER	7499	23-FEB-90	5,000.00	200.00	

Figure 6.6 Referential Integrity Constraints

Self-Referential Integrity Constraints :

Another type of referential integrity constraint, shown in Figure –6.7, is called a self-referential integrity constraint. This type of foreign key references a parent key in the same table.

In Figure-6.7, the referential integrity constraint ensures that every value in the mgr column of the employees table corresponds to a value that currently exists in the empno column of the same table, but not necessarily in the same row, because every manager must also be an employee. This integrity constraint eliminates the possibility of erroneous employee numbers in the mgr column.

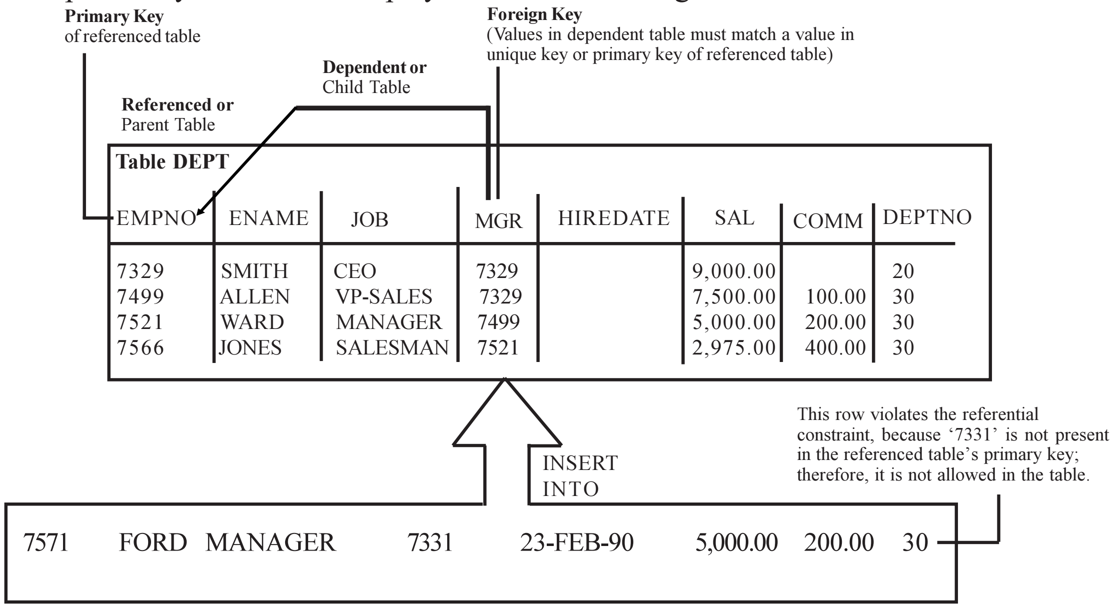

Primary Key
of referenced table
Foreign Key
(Values in dependent table must match a value in unique key or primary key of referenced table)
Dependent or
Child Table
Referenced or
Parent Table
INSERT
INTO
This row violates the referential constraint, because '7331' is not present in the referenced table's primary key; therefore, it is not allowed in the table.

Table DEPT

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7329	SMITH	CEO	7329		9,000.00		20
7499	ALLEN	VP-SALES	7329		7,500.00	100.00	30
7521	WARD	MANAGER	7499		5,000.00	200.00	30
7566	JONES	SALESMAN	7521		2,975.00	400.00	30

7571	FORD	MANAGER	7331	23-FEB-90	5,000.00	200.00	30

Figure 6.7 Single Table Referential Constraints

SQL Foreign Key or Referential Integrity :

This constraint identifies any column referencing the Primary Key in another table. It establishes a relationship between two columns in the same table or between different tables. For a column to be defined as a Foreign Key, it should be a defined as a Primary Key in the table which it is referring. One or more columns can be defined as Foreign key.

Syntax to Define a Foreign Key at Column Level :

```
[CONSTRAINTconstraint_name]REFERENCES
        Referenced_Table_name(column_name)
```

Syntax to Define a Foreign Key at Table Level :

```
[CONSTRAINT constraint_name] FOREIGN KEY(column_name)
        REFERENCES referenced_table_name(column_name);
```

For Example:

1. Lets use the "Dept" table and "Emp".

Foreign Key at Column Level :

```
CREATE TABLE Dept
( deptno number(5) CONSTRAINT pd_id_pk PRIMARY KEY,
dname char(20),
locchar(20));
CREATE TABLE Emp
( empno  number(5) CONSTRAINT od_id_pk PRIMARY KEY,
ename char(20),
job char(20),
mgr number(5),
hiredate date,
sal number(8,2),comm number (5),
comm number (5,2),
deptno number(5) CONSTRAINT pd_id_fk REFERENCES dept(deptno)
);
```

Foreign Key at Table Level :

```
( empno  number(5),
ename char(20),
job char(20),
mgr  number(5),
hiredate date,
sal number(8,2),comm number (5),
comm number (5,2),
deptno number(5)
CONSTRAINT od_id_pk PRIMARY KEY(empno),
CONSTRAINT pd_id_fk FOREIGN KEY (deptno) REFERENCES dept(deptno)
);
```

2. If the employee table has a 'mgr_id' i.e, manager id as a foreign key which references primary key 'id' within the same table, the query would be like,

```
CREATE TABLE Emp
( empno number(5)  PRIMARY KEY,
ename char(20),
job char(20),
mgr number(5) ) REFERENCES Emp(empno),
hiredate date,
sal number(8,2),comm number (5),
comm number (5,2),
deptno number(5)
);
```

Nulls and Foreign Keys :

The relational model permits the value of foreign keys either to match the referenced primary or unique key value, or be null. If any column of a composite foreign key is null, then the non-null portions of the key do not have to match any corresponding portion of a parent key.

Actions Defined by Referential Integrity Constraints :

Referential integrity constraints can specify particular actions to be performed on the dependent rows in a child table if a referenced parent key value is modified. The referential actions supported by the FOREIGN KEY integrity constraints of Oracle are UPDATE and DELETE NO ACTION, and DELETE CASCADE.

Update and Delete no Action :

The No Action (default) option specifies that referenced key values cannot be updated or deleted if the resulting data would violate a referential integrity constraint. For example, if a primary key value is referenced by a value in the foreign key, then the referenced primary key value cannot be deleted because of the dependent data.

Delete Cascade :

A **delete cascades** when rows containing referenced key values are deleted, causing all rows in child tables with dependent foreign key values to also be deleted. For example, if a row in a parent table is deleted, and this row's primary key value is referenced by one or more foreign key values in a child table, then the rows in the child table that reference the primary key value are also deleted from the child table.

Syntax :

```
Column datatype (size) REFERENCES tablename [(column name)] [ON DELETE CASCADE]
```

Example :

```
CREATE TABLE employee
( id number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
mgr_id number(5) REFERENCES employee(id) ON DELETE CASCADE,
salary number(10),
location char(10)
);
```

If ON DELETE CASCADE option is set, a DELETE operation in the master table will trigger the DELETE operation for the correcponding records in the detail table .

After a referential constraint has been defined, it can be dropped or altered by an ALTER TABLE statement. To drop a foreign or parent key after a referential constraint has been defined, you must first drop the constraint and then alter the table.

Update Cascade :

A **update cascades** when rows containing referenced key values are updated , causing all rows in child tables with dependent foreign key values to also be updated. For example, if a row in a parent table is updated , and this row's primary key value is referenced by one or more foreign key values in a child table, then the rows in the child table that reference the primary key value are also update from the child table.

Syntax :

Column datatype (size) REFERENCES tablename [(column name)] [ON UPDATE CASCADE]

Example :

```
CREATE TABLE employee
( id number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
mgr_id number(5) REFERENCES employee(id) ON UPDATE CASCADE,
salary number(10),
location char(10)
);
```

If ON UPDATE CASCADE option is set, a UPDATE operation in the master table will trigger the UPDATE operation for the correcponding records in the detail table .

After a referential constraint has been defined, it can be dropped or altered by an ALTER TABLE statement. To drop a foreign or parent key after a referential constraint has been defined, you must first drop the constraint and then alter the table.

To DROP the Foreign Key We Use Following Syntax :

ALTER TABLE table name DROP FOREIGN KEY ;

To DROP the foreign key we use define it by using constraint following syntax:

ALTER TABLE table name DROP CONSTRAINT constraints name;

Self-Learning Exercise :

5. What are the advantages and disadvantages of primary key and foreign key in SQL?

6.3 Summary

RDBMS stands for **R**elational **D**atabase **M**anagement **S**ystem. RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by E. F. Codd. The data in RDBMS is stored in database objects called **tables**. The table is a collection of related data entries and it consists of columns and rows. Every table is broken up into smaller entities called fields. The fields in the EMPLOYEE table consist of ID, NAME, AGE, ADDRESS and SALARY. A field is a column in a table that is designed to maintain specific information about every record in the table. A **record,** also called a row of data, is each individual entry that exists in a table.A record is a horizontal entity in a table.A column is a vertical entity in a table that contains all information associated with a specific field in a table.

A NULL value in a table is a value in a field that appears to be blank which means A field with a NULL value is a field with no value. It is very important to understand that a NULL value is different than a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation.

Constraints are the rules enforced on data columns on table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

Constraints could be column level or table level. Commonly used constraints are-

Not Null Constraint : Ensures that a column cannot have Null value.

Default Constraint : Provides a default value for a column when none is specified.

Unique Constraint : Ensures that all values in a column are different.

Primary Key : Uniquely identified each rows/records in a database table.

Foreign Key : Uniquely identified a rows/records in any another database table.

Check Constraint : The CHECK constraint ensures that all values in a column satisfy certain conditions.

6.4 Answers to Self-Learning Exercises

1. The main difference of DBMS & RDBMS is RDBMS have Normalization. Normalization means to refining the redundant information and the DBMS hasn't normalization concept.

2.
   ```
   CREATE TRIGGER SAL_ADJ
   AFTER UPDATE OF SALARY ON EMPLOYEE
   REFERENCING OLD AS OLD_EMP
                      NEW AS NEW_EMP
   FOR EACH ROW MODE DB2SQL
   WHEN (NEW_EMP.SALARY > (OLD_EMP.SALARY *1.20))
   BEGIN ATOMIC
   SIGNAL SQLSTATE '75001'('Invalid Salary Increase - Exceeds 20%');
   END
   ```

3. We can call stored procedure explicitly, but trigger is automatically invoked when the action defined in trigger is done. Stored procedure can't be inactive but trigger can be Inactive. Triggers are used to initiate a particular activity after fulfilling certain condition. Ttrigger can be define and can be enable and disable according to need.

4. Both primary key and unique enforce uniqueness of the column on which they are defined. But we can define more than one column as a unique but a relation have only one primary key. Another major difference is that, primary key doesn't allow NULLs, but unique key allows one Null only .

5. Primary Key

Advantages

(1) It is a unique key on which all the other candidate keys are functionally dependent

Disadvantage

(1) There can be more than one keys on which all the other attributes are dependent on.

Foreign Key :

It allows referencing another table using the primary key for the other table

6.5 Self Assesment Questions

1. Explain different type of integrity constraints with example.
2. What do you meant by database integrity?
3. What are the advantages and limitation of triggers?
4. How are the nulls represented in database system?
5. What are the situations where you can use nulls?
6. What is a primary key? Give an example.
7. What is an alternate key? Give an example.
8. What do you mean by foreign key?
9. What are domain constraints?
10. What is entity integrity?
11. What is a referential integrity?

′ ′ ′

Unit - 7 : Normalization

Structure of the Unit

7.0 Objective

Topics covered within the unit are as follows,

- Idea about Database Design;
- Define a Relation;
- Know the purpose of Normalization;
- Understand the steps involved for normalization process.

7.1 Introduction

Normalization is a technique that is more applicable to record-based data models for e.g. a relational database model. Each of the processes can be carried out independently to arrive at normalized tables (depending on how detailed the decompositions is). If ER Modeling is done in detail, normalization may not be required at all. However, some people use the ER diagram as an input to normalization i.e. if the tables derived from ER diagrams are in the first normal form.

In this unit we will concentrate on normalization which is an important step in database design, particularly for relation DBMSs. The relational data model is based on a relation. When structuring data that is to be stored, the analyst must anticipate the need to access the data to meet unexpected requirements and to reduce redundancy. These can be achieved through the techniques of 'Normalization' that provides a systematic way of boiling data structures down to their simplest possible forms.

7.2 Database Design

Database design involves designing the conceptual model of the databases. This model is independent of the physical representation of data. Before actually implementing the database, the conceptual model is designed using various techniques.

The requirements of all the users are taken into account to decide the actual data that needs to be stored in the system. Once the conceptual model is designed, it can then be mapped to the DBMS/ RDBMS that is actually being used. Two of the widely used approaches are Entity-Relationship (ER) Modeling and Normalization.

7.2.1 Meaning of 'Relation'

A 'Relation' is a two-dimensional table. It consists of 'rows' which represent records and 'columns' which show the attributes of the entity. A relation is also called a file, it consists of a number of records which are also called as tuples. Record consists of a number of attributes which are also known as fields or domains.

In order for a relational structure to be useful and manageable, the relation tables must first be 'normalized'.

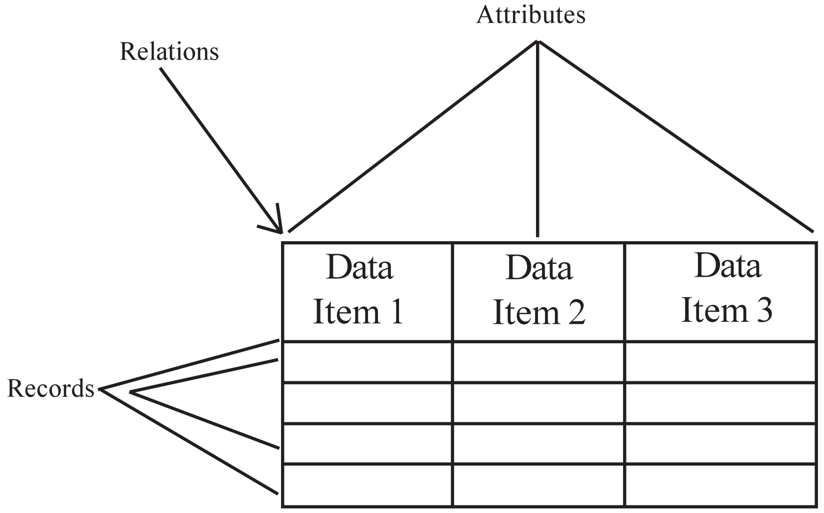
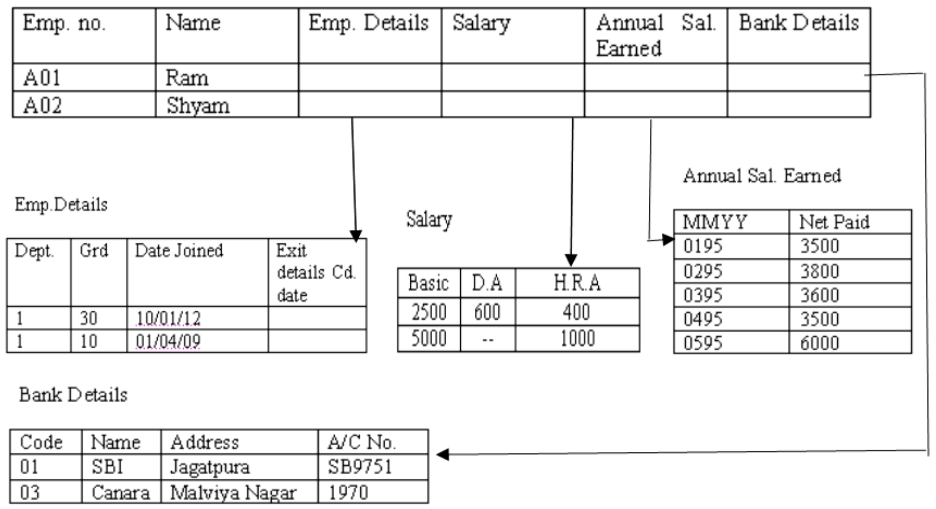


Figure 7.1: Components of a 'Relation'

Some of the properties of a relation are,

- **No duplication:** In the sense that no two records are identical.
- **Unique key:** Each relation has a unique key by which it can be accessed.
- **Order:** There is no significant order of data in the table



Emp. no.	Name	Emp. Details	Salary	Annual Sal. Earned	Bank Details
A01	Ram				
A02	Shyam				

Emp.Details

Dept.	Grd	Date Joined	Exit details Cd. date
1	30	10/01/12	
1	10	01/04/09	

Salary

Basic	D.A	H.R.A
2500	600	400
5000	--	1000

Annual Sal. Earned

MMYY	Net Paid
0195	3500
0295	3800
0395	3600
0495	3500
0595	6000

Bank Details

Code	Name	Address	A/C No.
01	SBI	Jagatpura	SB9751
03	Canara	Malviya Nagar	1970

Figure 7.2 : Un-normalized Employee Record

Figure 7.2 shows a relation (un-normalized form) of the employee entity. As per the figure an un-normalized form of an employee record consists of the following,

Employee no., employee name, employee details (department code, grade, date of joining, exit code and exit date), annual salary earned (MMYY, net paid), bank details (bank code, bank name, address, employees A/C no).

In case we want the names of all the employees whose grade is 20, we can scan the employee relation, noting the grade. Here the unique key is the employee number.

Here it is clearly seen that the employee's annual salary earned details which are Month and Year paid, net paid, are being repeated. Therefore this relation is not following the normalization process. Now, the question arises 'what is normalization'?

7.2.2 Introduction to Normalization

Normalization is a process of simplifying the relationship between data elements in a record. It is the transformation of complex data stores to a set of smaller, stable data structures.

Normalized data structures are simpler, more stable and are easier to maintain. Normalization can therefore be defined as a process of simplifying the relationship between data elements in a record.

7.2.3 Purpose of Normalization

Normalization is carried out for the following four reasons,

- To structure the data so that there is no repetition of data, this helps in saving space.
- To permit simple retrieval of data in response to query and report requests.
- To simplify the maintenance of the data through updates, insertions and deletions.
- To reduce the need to restructure or reorganize data when new application requirements arise.

7.2.4 Steps of Normalization

Normalization process starting with a data store developed for a data dictionary and analyst normalizes a data structure in three steps. Each step involves a re to simplify the data structure.

It consists of basic three steps:

1. First Normal Form which decomposes all data groups into two-dimensional records.
2. Second Normal form which eliminates any relationship in which data elements do not fully depend on the primary key of the record.
3. Third Normal Form which eliminates any ntain transitive dependencies.

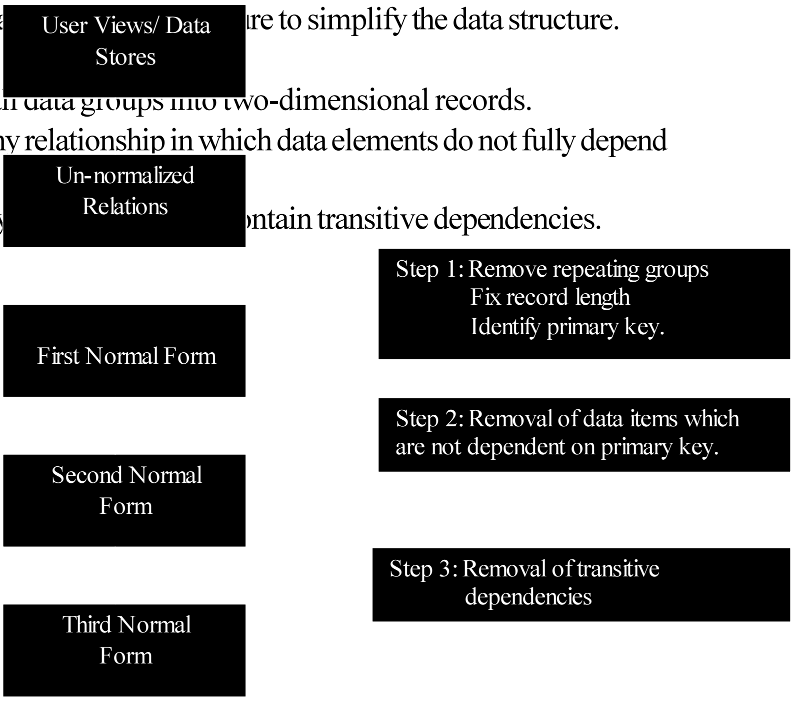

User Views/ Data Stores
Un-normalized Relations
First Normal Form
Second Normal Form
Third Normal Form
Step 1: Remove repeating groups Fix record length Identify primary key.
Step 2: Removal of data items which are not dependent on primary key.
Step 3: Removal of transitive dependencies

Figure 7.3 Steps of Normalization

7.2.5 Functional Dependency

So, before we start to learn about 'Normalization and its steps', we have to understand the term 'Functional dependency'.

Let X and Y be two attributes of a relation R. Given the value of X, if there is only one value of Y corresponding to it, then y is said to be functionally dependent on X.

X Y

So, Y is functionally dependent on X.

Consider an example, given the value of item code, there is only one value of item name for it. Thus item name is functionally dependent on item code.

$$\text{Item Code} \longrightarrow \text{Item Name}$$

Functional dependency may also be based on a composite attributes. Suppose, if we write,

$$X, Z \longrightarrow Y$$

It means that there is only one value of Y corresponding to given values of X,Z. So, Y is functionally dependent on the composite X, Z. In figure 7.3, for example, Order no., and Item code together determine Qty. and Price.

$$\text{Order no., Item code} \longrightarrow \text{Qty., Price}$$

Order no.	Order date	Item code	Quantity	Price/unit
1456	260289	3687	52	50.40
1456	260289	4627	38	60.20
1456	260289	3214	20	17.50
1886	040389	4629	45	20.25
1886	040389	4627	30	60.20
1788	040489	4627	40	60.20

Figure 7.4 : Normalized form of the Relation

There are mainly three types of functional dependencies,

(a) Full functional dependency

(b) Partial functional dependency

(c) Transitive dependency

(a) Full functional dependency :

Consider the following Relation

REPORT1(S#, SName, C#, CTitle, Iname, Room#, Marks, Grade)

S#	→	Student Number
SName	→	Student Name
C#	→	Course Number
CTitle	→	Course Title
Iname	→	Name of the Instructor who delivered the course
Room#	→	Room number which is assigned to respective instructor

Marks	$\rightarrow$	Scored in Course COURSE# by the student STUDENT#
Grade	$\rightarrow$	Obtained by student STUDENT# in Course C#

Functional dependencies in the given example are,

S#	$\rightarrow$	SName
C#	$\rightarrow$	CTitle
C#	$\rightarrow$	Iname (Assuming one course is thought by only one instructor)
IName	$\rightarrow$	Room# (Assuming each instructor has his won non-shared room)
S#.C#	$\rightarrow$	Marks
Marks	$\rightarrow$	Grade
S#.C#	$\rightarrow$	Grade

Dependency Diagram :

REPORT1(S#, C#, CTitle, SName, Iname, Room#, Marks, Grade)

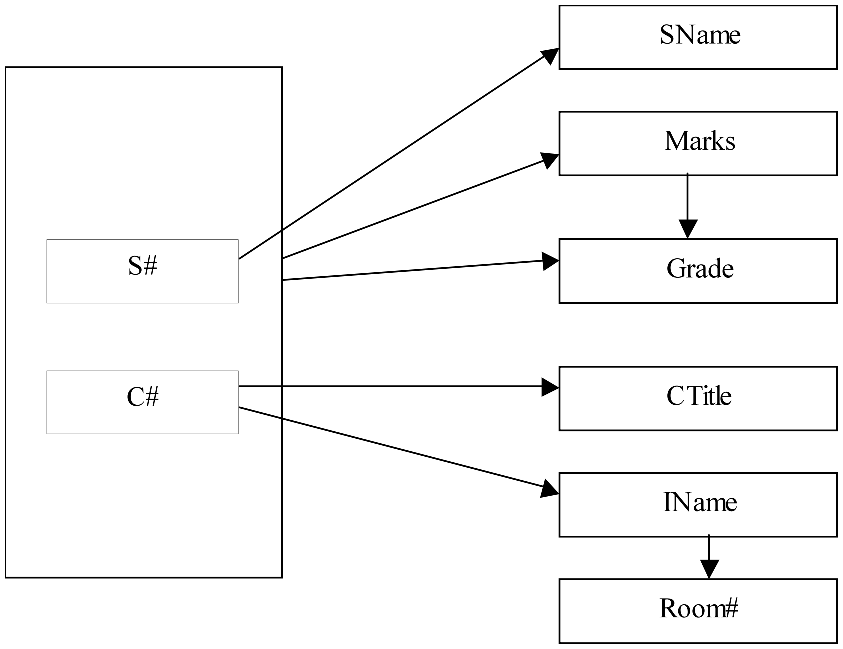


Figure 7.5 : Dependency Diagram

In above example, Marks is fully functionally dependent on S#, C# and not on subset of S#, C#. This means, marks cannot be determined either by S# or C# alone. It can be determined only using S# and C# together. Hence Marks is fully functionally dependent on S# and C#.

(b) Partial functional dependency :

If in a relation R, X and Y are attributes. Attribute Y is partially dependent on the attribute X only if it is dependent on a sub-set of attribute X.\

In the above relationship (figure 7.4) CTitle, Iname are partially dependent on composite attributes S#, C# because C# alone defines the CTitle, Iname.

(c) Transitive dependency :

In a give relation R, there are three attributes X, Y and Z. Y is functionally dependent on X and Z is functionally dependent on Y. Therefore there is an indirect dependency between X and Z. This is called 'Transitive (indirect) Dependency'.

$$X \rightarrow Y$$
$$Y \rightarrow Z$$
$$X \rightarrow Z$$

In the above example, Room# depends on Iname and in turn Iname depends on C#. Hence Room# transitively depends on C#.

Similarly, Grade depends on Marks, in turn Marks depends on S#, C# hence Grade depends transitively on S#, C#.

Check Your Progress - 7.1 :

1. A _________ is a two-dimensional table. It consists of 'rows' which represent records and 'columns' which show the attributes of the entity.

(a)	Relation	(b)	Software	
(c)	System	(d)	None of above	( )

2. A relation is also called a _________.

(a)	File	(b)	Data	
(c)	Software	(d)	None of above	( )

3. A Relation consists of a number of records which are called as__________.

(a)	Columns	(b)	Systems	
(c)	Tuples	(d)	All of above	( )

4. Record consists of a number of attributes which are also known as _______.

(a)	Fields	(b)	Package	
(c)	Programs	(d)	None of above	( )

5. _________reduces redundancy.

(a)	Normalization	(b)	System	
(c)	Software	(d)	Table	( )

7.3 Normal Forms

Normalization reduces redundancy. Redundancy is the unnecessary repetition of data. It can cause problems with storage and retrieval of data. Redundancy can lead to

- **Inconsistencies:** Errors are more likely to occur when facts are repeated.
- **Update Anomalies**
  - Inserting, modifying and deleting data may cause inconsistencies.
  - There is a high likelihood of updating or deleting data in one relation, while omitting to make corresponding changes in other relations.

7.3.1 First Normal Form (1NF)

A relation is said to be in first normal form if all attributes defined on domains containing atomic values. Consider the following relation; this relation is un-normalized because each row contains multiple values.

The relational model does not permit tables that the un-normalized. In the relational mode, every relation is in first normal form. Hence, the tables arrived at form the entity-relationship diagram should be at least in first normal form.

ECODE	DEPT	PROJCODE	HOURS
EN101	SYSTEMS	P27 P51 920	90 101 60
EN305	SALES	P27	109
EN508	ADMIN	P51 P27	Null 72

Figure 7.6 : Un-normalized Relation

Steps for above converting a database to 1NF:

- Put all double values in separate lines.

ECODE	DEPT	PROJCODE	HOURS
EN101	SYSTEMS	P27	90
EN101	SYSTEMS	P51	101
EN101	SYSTEMS	920	60
EN305	SALES	P27	109
EN508	ADMIN	P51	NULL
EN508	ADMIN	P27	72

Figure 7.7 : A relation in First Normal Form (1NF)

7.3.2 Second Normal Form (2NF)

Data in the tables in 1NF may be redundant. Consider the table in figure 7.6. This table stores the following attributes:

ECODE	:	employee code.
DEPT	:	department to which the employee belongs.
PROJCODE	:	code of project on which employee is working.
HOURS	:	number of hours worked on project.

The primary key here is composite (ECODE + PROJCODE). Few attributes in this table depends upon only part of the primary key as given below,

- PROJCODE + ECODE functionally determine HOURS.
- ECODE functionally determines DEPT. Attributes DEPT has no dependency on PROJCODE.

This situation could lead to the following problems,

(a) **Insertion :** The department of a particular employee cannot be recorded until the employee is assigned a project.

(b) **Updation :** For a given employee, the employee code and department is repeated several times. Hence, if an employee is transferred to another department, this change will have to be recorded in every instance or record of the employee. Any omissions will lead to inconsistencies.

(c) **Deletion :** If an employee completes work on a project, his/her record will be deleted. The information regarding the department the employee belongs to will also be lost.

The table in figure 7.6 should, therefore, be decomposed without any loss of information as shown under,

ECODE
EN101
EN305
EN508

ECODE	PROJCODE	HOURS
EN101	P27	90
EN101	P51	101
EN101	920	60
EN305	P27	109
EN508	P51	NULL
EN508	P27	72

Figure 7.8 : No-loss decomposition in Second Normal Form (2NF)

Notice that the original table can be reconstructed with a join.

A table is said to be in Second Normal Form when it is in First Normal Form, and every attribute in the record is functionally dependent upon the whole key, and not just a part of the key.

The steps for converting a database to Second Normal Form (2NF):

- Identify the functional dependencies in the relation.
- Identify attributes that are dependent only to a part of key (partial dependency).
- Remove partial dependencies by placing them in new relations together with a copy of determinant.

7.3.3 Third Normal Form (3NF)

A table is said to be in 3NF when it is in 2NF and every non-key attribute is functionally dependent on just the primary key. For example, consider the figure 7.8. The primary key is ECODE. The attribute DEPT is dependent on ECODE. The attribute DEPT-HEAD is dependent on DEPT-HEAD is the code of the department head. Notice that there is a transitive dependence between ECODE and DEPT-HEAD attributes.

ECODE	DEPT	DEPT-HEAD
*E*101	*SYSTEMS*	*E*901
*E*305	*SALES*	*E*906
*E*402	*SALES*	*E*906
*E*508	*ADMIN*	*E*908
*E*607	*FINANCE*	*E*909
*E*608	*FINANCE*	*E*909

Figure 7.8 : Table with Transitive Dependency

The problems with transitive dependency are:

- **Insertion :**

  The department head of a new department that does not have any employees as yet cannot be entered. This is because the primary key is unknown.

- **Updation :**

  For a given department, the department head's code is repeated several times. Hence, if a department head is moved to another department, the change will have to be made consistently across.

- **Deletion :**

  If a particular employee record is deleted, the information regarding the head of the department will also be deleted. Hence, there will be a loss of information.

  The relation is therefore, reduced to the following two relations:

ECODE	**DEPT**
*E*101	*SYSTEMS*
*E*305	*SALES*
*E*402	*SALES*
*E*508	*ADMIN*
*E*607	*FINANCE*
*E*608	*FINANCE*

DEPT	**DEPT-HEAD**
SYSTEMS	*E*901
SALES	*E*906
ADMIN	*E*908
FINANCE	*E*909

Figure 7.9 : Removing Transitive Dependency

Each non-key attribute depends of the key, the whole key and nothing but the key.

7.3.4 Boyce-Code Normal Form (BCNF)

The original definition of 3NF was inadequate in some situations. It was not satisfactory for relation that:

- Had multiple candidate keys, where
  - ⇒ Those candidate keys were composite.
  - ⇒ The candidate keys overlapped (had at least one attribute in common).

Hence, a new normal form – the Boyce-Codd Normal Formal (BCNF) was introduced. We must understand that in relations where the above three conditions do not apply, we can stop at the third normal form. In such cases, 3NF is the same as BCNF.

Let us examine BCNF in detail. Consider the table in figure 7.10. The relation PROJECT holds details on the number of hours spent by each employee working on each project.

ECODE	**EMAILID**	**PROJCODE**	**HOURS**
*E*1	*bk@rediffmail.com*	*P*2	48
*E*1	*bk@rediffmail.com*	*P*5	100
*E*1	*bk@rediffmail.com*	*P*6	15
*E*4	*rahul@rediffmail.com*	*P*5	250
*E*5	*rahul@rediffmail.com*	*P*5	75

Figure 7.10 : Project Table

Notice the following dependencies in this relation :

- The attributes ECODE and PROJCODE functionally determine HOURS.
- The attribute EMAILID and PROJCODE also functionally determine determines HOURS.
- ECODE functionally determine EMAILID.
- EMAILID functionally determine ECODE.

Notice that this relation has :

- Multiple candidate keys.
- The candidate keys are composite.
- The candidate keys overlap – PROJCODE is common.

This is a case for the Boyce-Codd Normal Form. This relation is in 3NF. The only non-key item is HOURS, and it is dependent on the whole key and only the key, i.e. PROJCODE+ECODE or PROJCODE+EMAILID. However, this relation has redundancies. If the emailed of and employee is changed, the change will have to be made in every tuple of the relation, otherwise inconsistencies will creep in. This table needs to be further decomposed to eliminate dependence between the candidate key columns. Figure 7.10 illustrates the non-loss decomposition of the table in figure 7.11.

ECODE	PROJCODE	HOURS
*E*1	*P*2	48
*E*1	*P*5	100
*E*1	*P*6	15
*E*4	*P*5	250
*E*4	*P*5	75

ECODE	EMAILID
*E*1	*bk@rediffmail.com*
*E*4	*rahul@rediffmail.com*

Figure 7.11 : Non-loss decomposition of Project table in BCNF.

Check Your Progress 7.2 :

1. Normalization is the ________________of logical database design.
   (a) Top-down approach (b) bottom-up approach
   (c) Database systems (d) None of above ( )
2. ________ is a step-by-step decomposition of complex records into simple records.
   (a) Normalization (b) System
   (c) Database (d) None of above ( )
3. A relation is said to be in _____________if all attributes defined on domains containing atomic values.
   (a) First Normal Form (b) Second Normal Form
   (c) Third Normal Form (d) None of above ( )
4. A table is said to be in _____________when it is in First Normal Form, and every attribute in the record is functionally dependent upon the whole key, and not just a part of the key.
   (a) First Normal Form (b) Second Normal Form
   (c) Third Normal Form (d) None of above ( )
5. A table is said to be in ______________ when it is in 2NF and every non-key attribute is functionally dependent on just the primary key.
   (a) First Normal Form (b) Second Normal Form
   (c) Third Normal Form (d) None of above ( )

7.4 Summary

A 'Relation' is a two-dimensional table. It consists of 'rows' which represent records and 'columns' which show the attributes of the entity. A relation is also called a file, it consists of a number of records which are also called as tuples. Record consists of a number of attributes which are also known as fields or domains. Normalization is a technique that is more applicable to record-based data models for e.g. a relational database model. Each of the processes can be carried out independently to arrive at normalized tables (depending on how detailed the decompositions is). Normalization is the bottom-up approach of logical database design. It is a step-by-step decomposition of complex records into simple records. Normalization reduces redundancy. Redundancy can lead to inconsistencies as well as Insertion, Updation and Deletion anomalies. The different stages of normalization are known as 'normal forms'. The most important and widely used of these are: 1NF, 2NF, 3NF, BCNF, 4NF etc. A relation is said to be in first normal form if all attributes defined on domains containing atomic values.

7.5 Answer to the Self -Learning Exercises

Answer to Check Your Progress - 7.1

1.	(a)	2.	(a)	3.	(c)
4.	(a)	5.	(a)		

Answer to Check your progress 7.2

1.	(b)	2.	(a)	3.	(a)
4.	(b)	5.	(c)		

7.6 Self Assessment Questions

1. What is Normalization? Discuss its need.
2. Define Boyce-codd normal form. How does it differ from 3NF? Why it is considered as a stronger form of 3NF?
3. What are the design goals of a good relational database design? Is it always possible to achieve these goals? If some of these goals are not achievable, what alternate goals should you aim for and why?
4. What do you understand by 'Functional Dependency'? Discuss various types of dependencies with suitable examples.
5. Explain 1NF, 2NF, 3NF and BCNF with suitable example.

′ ′ ′

Unit - 8 : Introduction to Popular RDBMS Packages

Structure of the Unit

8.0 Objective

A **database** is a collection of data that is related to a particular topic or purpose. As an example, employee records in a filing cabinet, a collection of sales leads in a notebook, are examples of collections of data or databases. A **Database Management System** (DBMS) is a system that stores and retrieves information in a database. It is used to help you organize your data according to a subject, so that it is easy to track and verify your data, and you can store information about how different subjects are related, so that it makes it easy to bring related data together.

A DBMS stores data in a table where the entries are filed under a specific category and are properly indexed. This allowed programmers to have a lot more structure when saving or retrieving data. A relational database contains data in more than one table. Each table contains a database that is then linked to other tables with respect to their relationships.

This unit provides a brief description about various popular RDBMS packages. Some of them are used for commercial purpose while others are available as an open source packages.

8.1 Introduction

Data is the most important aspect in computing. Any program, whether big or small, needs data in order to process and produce its output; which often is some sort of data. Storing data has evolved a lot over the last few years. The first method of storing data before was in text files but this way very inefficient and difficult to deal with large amounts of data.

DBMS provides search functionalities in order to find a certain database entry. Once it is found, you can then pull out any other related information from that entry. DBMS is a very competent system for keeping track of data, but it doesn't scale very well. Dealing with huge databases, although possible, becomes a huge task in DBMS.

Relational database contains data in more than one table. Each table contains a database that is then linked to other tables with respect to their relationships. RDBMS is an improvement over the older DBMS. It provides the mechanism to overcome the restrictions that DBMS faces.

8.2 Various Types of RDBMS Packages

A DBMS that is based on relational model is called as RDBMS. Relation model is most successful mode of all three models. Designed by E.F.Codd, relational model is based on the theory of sets and relations of mathematics.

Relational model represents data in the form a table. A table is a two dimensional array containing rows and columns. Each row contains data related to an entity such as a student. Each column contains the data related to a single attribute of the entity such as student name. One of the reasons behind the success of relational model is its simplicity. It is easy to understand the data and easy to manipulate.

Another important advantage with relational model, compared with remaining two models is, it doesn't bind data with relationship between data item. Instead it allows you to have dynamic relationship between entities using the values of the columns.

Almost all Database systems that are sold in the market, now- a-days, have either complete or partial implementation of relational model.

8.2.1 Commercial RDBMS V/s. Open Source RDBMS

Basically Commercial (such as Oracle, MSSQL Server etc) will cost money, Non Commercial (MySQL, PostgreSQL) are 'free' or at least open source. They are developed in different ways and come with different levels of support.

Check Your Progress 8.1 :

1. A __________ is a system that stores and retrieves information in a database.
   (a) Operating System (b) Software System
   (c) Design System (d) Database Management System ( )
2. A ________ stores data in a table where the entries are filed under a specific category and are properly indexed.
   (a) DBMS (b) Network System
   (c) Operating System (d) None of above ( )
3. Relational database contains data in __________ table.
   (a) One (b) Two
   (c) More than one (d) None of above ( )
4. Relational Model of DBMS was designed by _________.
   (a) Thomas Cook (b) Saint Louis
   (c) E.F.Codd (d) None of above ( )
5. The relational model is based on the theory of____________.
   (a) Analytical Systems (b) Gene Analysis
   (c) Object Analysis Systems (d) Sets & Relations of Mathematics ( )

8.3 SQL Server

Microsoft SQL Server is a relational database server which is developed by Microsoft. MS SQL Server is a software product whose primary function is to store and retrieve data as requested by other software applications from networks or other systems. There are at least a dozen different editions of Microsoft SQL Server aimed at different audiences and for different workloads, some of them are,

Various versions of SQL Server		
Version	*Year*	*Release Name*
1.0 (OS/2)	*1989*	*SQL Server 1.0 (16bit)*
1.1 (OS/2)	*1991*	*SQL Server 1.1 (16bit)*
4.21 (WinNT)	*1993*	*SQL Server 4.21*
6.0	*1995*	*SQL Server 6.0*
6.5	*1996*	*SQL Server 6.5*
7.0	*1998*	*SQL Server 7.0*
7.0	*1999*	*SQL Server 7.0 (with OLAP Tools)*
8.0	*2000*	*SQL Server 2000*
8.0	*2003*	*SQL Server 2000 (64-bit Edition)*
9.0	*2005*	*SQL Server 2005*
10.0	*2008*	*SQL Server 2008*
10.25	*2010*	*SQL Azure*
10.5	*2010*	*SQL Server 2008 R2*
11.0		*SQL Server 2012*

Table 8.1: Various Versions of SQL Server

8.3.1 Characteristics

Microsoft SQL Server is an application used to create computer databases for the Microsoft Windows family of server operating systems. Microsoft SQL Server provides an environment used to generate databases that can be accessed from workstations, the Internet, or other media such as a personal digital assistant (PDA).

Microsoft SQL Server 2000 is a full-featured relational database management system (RDBMS) that offers a variety of administrative tools to ease the burdens of database development, maintenance and administration. Six of the more frequently used tools are: Enterprise Manager, Query Analyzer, SQL Profiler, Service Manager, Data Transformation Services and Books Online. Let's take a brief look at each:

- **Enterprise Manager** is the main administrative console for SQL Server installations. It provides you with a graphical 'birds-eye' view of all of the SQL Server installations on your network. You can perform high-level administrative functions that affect one or more servers, schedule common maintenance tasks or create and modify the structure of individual databases.
- **Query Analyzer** offers a quick method for performing queries against any of your SQL Server databases. It's a great way to quickly pull information out of a database in response to a user request, test queries before implementing them in other applications, create/ modify stored procedures and execute administrative tasks.
- **SQL Profiler** provides a window into the inner workings of your database. You can monitor many different event types and observe database performance in real time. SQL Profiler allows you to capture and replay system 'traces' that log various activities. It's a great tool for optimizing databases with performance issues or troubleshooting particular problems.

- **Service Manager** is used to control the MSSQLServer (the main SQL Server process), MSDTC (Microsoft Distributed Transaction Coordinator) and SQLServerAgent processes. An icon for this service normally resides in the system tray of machines running SQL Server. You can use Service Manager to start, stop or pause any one of these services.
- **Data Transformation Services (DTS)** provide an extremely flexible method for importing and exporting data between a Microsoft SQL Server installation and a large variety of other formats. The most commonly used DTS application is the 'Import and Export Data' wizard found in the SQL Server program group.
- **Books Online** is an often overlooked resource provided with SQL Server that contains answers to a variety of administrative, development and installation issues. It's a great resource to consult before turning to the Internet or technical support.

8.3.2 Strength

SQL Server includes an assortment of add-on services which are the strength of this application. While these are not essential for the operation of the database system, they provide value added services on top of the core database management system. Some of them are given as under,

✓ **Service Broker :** Used inside an instance, it is used to provide an asynchronous programming environment. For cross instance applications, Service Broker communicates over TCP/IP and allows the different components to be synchronized together, via exchange of messages. The Service Broker, which runs as a part of the database engine, provides a reli able messaging and message queuing platform for SQL Server applications.

✓ **Replication Services :** SQL Server Replication Services are used by SQL Server to replicate and synchronize database objects, either in entirety or a subset of the objects present, across replication agents, which might be other database servers across the network, or database caches on the client side. Replication follows a publisher/subscriber model i.e., the changes are sent out by one database server ('publisher') and are received by others ('subscribers'). SQL Server supports three different types of replications:

- o **Transaction Replication :** Each transaction made to the publisher database (master database) is synced out to subscribers, who update their databases with the transaction. Transactional replication synchronizes databases in near real time.
- o **Merge replication:** Changes made at both the publisher and subscriber databases are tracked, and periodically the changes are synchronized bi-directionally between the publisher and the subscribers. If the same data has been modified differently in both the publisher and the subscriber databases, synchronization will result in a conflict which has to be resolved - either manually or by using pre-defined policies.
- o **Snapshot replication:** Snapshot replication published a copy of the entire database (the then-snapshot of the data) and replicates out to the subscribers. Further changes to the snapshot are not tracked.

✓ **Analysis Services :** SQL Server Analysis Services adds OLAP and data mining capabilities for SQL Server databases. The OLAP engine supports MOLAP, ROLAP and HOLAP storage modes for data. Analysis Services supports the XML for Analysis standard as the underlying communication protocol. The cube data can be accessed using MDX and LINQ queries. Data mining specific functionality is exposed via the DMX query language. Analysis Services includes various algorithms - Decision trees, clustering algorithm, Naive Bayes algorithm, time series analysis, sequence clustering algorithm, linear and logistic regression analysis, and neural networks - for use in data mining.

✓ **Reporting Services:** SQL Server Reporting Services is a report generation environment for data gathered from SQL Server databases. It is administered via a web interface. Reporting services features a web services interface to support the development of custom reporting applications. Reports are created as RDL files. Reports can be designed using recent versions of Microsoft Visual Studio (Visual Studio.NET 2003, 2005, and 2008) with Business Intelligence Development Studio, installed or with the included Report Builder. Once created, RDL files can be rendered in a variety of formats including Excel, PDF, CSV, XML, TIFF (and other image formats), and HTML Web Archive.

- ✓ **Notification Services:** Originally introduced as a post-release add-on for SQL Server 2000, Notification Services was bundled as part of the Microsoft SQL Server platform for the first and only time with SQL Server 2005. SQL Server Notification Services is a mechanism for generating data-driven notifications, which are sent to Notification Services subscribers. A subscriber registers for a specific event or transaction (which is registered on the database server as a trigger); when the event occurs, Notification Services can use one of three methods to send a message to the subscriber informing about the occurrence of the event. These methods include SMTP, SOAP, or by writing to a file in the filesystem. Notification Services was discontinued by Microsoft with the release of SQL Server 2008 in August 2008, and is no longer an officially supported component of the SQL Server database platform.
- ✓ **Integration Services :** SQL Server Integration Services is used to integrate data from different data sources. It is used for the ETL capabilities for SQL Server for data warehousing needs. Integration Services includes GUI tools to build data extraction workflows integration various functionality such as extracting data from various sources, querying data, transforming data including aggregating, duplication and merging data, and then loading the transformed data onto other sources, or sending e-mails detailing the status of the operation as defined by the user.

8.3.3 Limitations

Although SQL Server has several strong features for Database creation, security, monitoring etc. but there are a few limitations with SQL Server 2005 as it introduced Database Mirroring, but it was not fully supported until the first Service Pack release (SP1). In the initial release (RTM) of SQL Server 2005, database mirroring was available, but unsupported. In order to implement database mirroring in the RTM version, one had to apply trace flag 1400 at startup. Database mirroring is a high availability option that provides redundancy and failover capabilities at the database level.

8.4 Oracle

Oracle databases are used with many applications, for large and small organizations operating in a variety of sectors. Powerful, secure and reliable database applications can be built with Oracle, as the platform has a range of features for handling both the data itself and user access to it. Oracle also provides tools for maintenance and developer functions. Oracle is a proprietary database system, and therefore has to be purchased with a commercial license before it can be used.

8.4.1 Characteristics

An **Oracle** database is a collection of data treated as a unit. The purpose of a database is to store and retrieve related information. A database server is the key to solving the problems of information management. In general, a server reliably manages a large amount of data in a multiuser environment so that many users can concurrently access the same data. All this is accomplished while delivering high performance. A database server also prevents unauthorized access and provides efficient solutions for failure recovery.

Oracle Database is the first database designed for enterprise grid computing, the most flexible and cost effective way to manage information and applications. Enterprise grid computing creates large pools of industry-standard, modular storage and servers. With this architecture, each new system can be rapidly provisioned from the pool of components. There is no need for peak workloads, because capacity can be easily added or reallocated from the resource pools as needed. The database has logical structures and physical structures. Because the physical and logical structures are separate, the physical storage of data can be managed without affecting the access to logical storage structures.

Oracle is an ORDBMS (Object Relational Database Management System) product. This means that Oracle databases model data in terms of objects and their relationships. Data modeled within an Oracle system is divided into conceptual entities, each of which has a set of attributes. ORDBMS modeling also has some of the characteristics of Object Oriented development, in which application responsibilities are divided between logical objects, each of which has a well-defined set of characteristics and behaviors.

Oracle database architecture models the data for a system as a single unit stored on a database server. This data is managed as a unit, but is designed to allow access by multiple users at any one time, while still maintaining data integrity. This means that even where multiple users have the ability to view the same data items, to update, insert and even remove data, an Oracle system is able to prevent corruption of the data. This is known as transaction management and is a key feature of Oracle systems.

Oracle databases are developed with various features to enhance the efficiency, reliability and security of systems in different organizational environments. Concurrency features in Oracle prevent the same item of data from being changed by more than one user at the same time. Consistency is also a key feature, meaning that when a user accesses a data item, they can be sure it is accurate at the time of access. Management of user accounts within an Oracle database is also an important feature, as are the many administration tools, which facilitate any necessary maintenance work by database administrators (DBAs).

There are two aspects of Oracle database development: development of the database itself and development of the user interface application for accessing it. Development of a database is carried out through SQL programming, where databases can be created in the form of tables with columns. The columns are for specific data items and have set data types. Oracle databases can also be created through a user interface. The set of database queries required by an application will also be constructed in SQL code.

A database application consists of the database itself, and the application that provides an interface for user access. These interfaces can be developed using many different languages and technologies. Oracle databases provide APIs (Application Programming Interfaces) to assist developers in some of the more common languages used. Oracle applications can be built in languages such as Java, PHP and PL/SQL among others.

8.4.2 Strength

Oracle includes several software mechanisms to fulfill the following important requirements of an information management system:

- Data concurrency of a multiuser system must be maximized. A primary concern of a multiuser database management system is how to control **concurrency**, which is the simultaneous access of the same data by many users. Without adequate concurrency controls, data could be updated or changed improperly, compromising data integrity. One way to manage data concurrency is to make each user wait for a turn. The goal of a database management system is to reduce that wait so it is either nonexistent or negligible to each user. All data manipulation language statements should proceed with as little interference as possible, and destructive interactions between concurrent transactions must be prevented. Destructive interaction is any interaction that incorrectly updates data or incorrectly alters underlying data structures. Neither performance nor data integrity can be sacrificed. Oracle resolves such issues by using various types of locks and a multiversion consistency model. These features are based on the concept of a transaction. It is the application designer's responsibility to ensure that transactions fully exploit these concurrency and consistency features.
- Data must be read and modified in a consistent fashion. The data a user is viewing or changing is not changed (by other users) until the user is finished with the data. Read consistency, as

supported by Oracle, does the following:

- ✓ Guarantees that the set of data seen by a statement is consistent with respect to a single point in time and does not change during statement execution (statement-level read consistency).
- ✓ Ensures that readers of database data do not wait for writers or other readers of the same data.
- ✓ Ensures that writers of database data do not wait for readers of the same data.
- ✓ Ensures that writers only wait for other writers if they attempt to update identical rows in concurrent transactions.

- High performance is required for maximum productivity from the many users of the database system.
- To manage the multiversion consistency model, Oracle must create a read-consistent set of data when a table is queried (read) and simultaneously updated (written). When an update occurs, the original data values changed by the update are recorded in the database undo records. As long as this update remains part of an uncommitted transaction, any user that later queries the modified data views the original data values. Oracle uses current information in the system global area and information in the undo records to construct a **read-consistent view** of a table's data for a query. Only when a transaction is committed are the changes of the transaction made permanent. Statements that start *after* the user's transaction is committed only see the changes made by the committed transaction. The transaction is key to Oracle's strategy for providing read consistency. This unit of committed (or uncommitted) SQL statements:
  - ✓ Dictates the start point for read-consistent views generated on behalf of readers
  - ✓ Controls when modified data can be seen by other transactions of the database for reading or updating
- By default, Oracle guarantees statement-level read consistency. The set of data returned by a single query is consistent with respect to a single point in time. However, in some situations, you might also require transaction-level read consistency. This is the ability to run multiple queries within a single transaction, all of which are read-consistent with respect to the same point in time, so that queries in this transaction do not see the effects of intervening committed transactions. If you want to run a number of queries against multiple tables and if you are not doing any updating, you prefer a **read-only transaction**.
- Oracle also uses **locks** to control concurrent access to data. When updating information, the data server holds that information with a lock until the update is submitted or committed. Until that happens, no one else can make changes to the locked information. This ensures the data integrity of the system. Oracle provides unique non-escalating row-level locking. Oracle always locks only the row of information being updated. Because Oracle includes the locking information with the actual rows themselves, Oracle can lock an unlimited number of rows so users can work concurrently without unnecessary delays.
- Oracle locking is performed automatically and requires no user action. Implicit locking occurs for SQL statements as necessary, depending on the action requested. Oracle's lock manager automatically locks table data at the row level. By locking table data at the row level, contention for the same data is minimized. Oracle's lock manager maintains several different types of row locks, depending on what type of operation established the lock. The two general types of locks are **exclusive locks** and **share locks**. Only one exclusive lock can be placed on a resource (such as a row or a table); however, many share locks can be placed on a single resource. Both exclusive and share locks always allow queries on the locked resource but prohibit other activity on the resource (such as updates and deletes).

- Under some circumstances, a user might want to override default locking. Oracle allows manual override of automatic locking features at both the row level (by first querying for the rows that will be updated in a subsequent statement) and the table level.
- Oracle Database provides a high degree of self-management - automating routine DBA tasks and reducing complexity of space, memory, and resource administration. Oracle self-managing database features include the following: automatic undo management, dynamic memory management, Oracle-managed files, meantime to recover, free space management, multiple block sizes, and Recovery Manager (RMAN).
- Enterprise Manager is a system management tool that provides an integrated solution for centrally managing your heterogeneous environment. Combining a graphical console, Oracle Management Servers, Oracle Intelligent Agents, common services, and administrative tools, Enterprise Manager provides a comprehensive systems management platform for managing Oracle products.
- Automatic Storage Management automates and simplifies the layout of data files, control files, and log files. Database files are automatically distributed across all available disks, and database storage is rebalanced whenever the storage configuration changes. It provides redundancy through the mirroring of database files, and it improves performance by automatically distributing database files across all available disks. Rebalancing of the database's storage automatically occurs whenever the storage configuration changes.
- Traditionally, the operating systems regulated resource management among the various applications running on a system, including Oracle databases. The Database Resource Manager controls the distribution of resources among various sessions by controlling the execution schedule inside the database. By controlling which sessions run and for how long, the Database Resource Manager can ensure that resource distribution matches the plan directive and hence, the business objectives.
- Oracle provides for complete media recovery from all possible types of hardware failures, including disk failures. Options are provided so that a database can be completely recovered or partially recovered to a specific point in time. If some data files are damaged in a disk failure but most of the database is intact and operational, the database can remain open while the required table spaces are individually recovered. Therefore, undamaged portions of a database are available for normal use while damaged portions are being recovered.
- A data warehouse is a relational database designed for query and analysis rather than for transaction processing. It usually contains historical data derived from transaction data, but it can include data from other sources. It separates analysis workload from transaction workload and enables an organization to consolidate data from several sources. In addition to a relational database, a data warehouse environment includes an extraction, transportation, transformation, and loading (ETL) solution, an online analytical processing (OLAP) engine, client analysis tools, and other applications that manage the process of gathering data and delivering it to business users.
- Oracle has many SQL operations for performing analytic operations in the database. These include ranking, moving averages, cumulative sums, ratio-to-reports, and period-over-period comparisons.
- Oracle includes datatypes to handle all the types of rich Internet content such as relational data, object-relational data, XML, text, audio, video, image, and spatial. These datatypes appear as native types in the database. They can all be queried using SQL. A single SQL statement can include data belonging to any or all of these datatypes.
- Oracle includes built-in spatial features that let you store, index, and manage location content (assets, buildings, roads, land parcels, sales regions, and so on.) and query location relationships using the power of the database. The Oracle Spatial Option adds advanced spatial features such as linear reference support and coordinate systems.

- The Oracle database provides **discretionary access control**, which is a means of restricting access to information based on privileges. The appropriate privilege must be assigned to a user in order for that user to access a schema object. Appropriately privileged users can grant other users privileges at their discretion. Oracle manages database security using several different facilities:
  - ✓ Authentication to validate the identity of the entities using your networks, databases, and applications
  - ✓ Authorization processes to limit access and actions, limits that are linked to user's identities and roles.
  - ✓ Access restrictions on objects, like tables or rows.
  - ✓ Security policies
  - ✓ Database auditing

Other characteristics are given as under,

- ✓ Oracle is very simple to install.
- ✓ There is no need to install agents when you are going to install Oracle.

8.4.3 Limitations

- Oracle not provides any kind of GUI.
- There is no direct memory access for fine grained monitoring.
- There is no graphical tool like Excel.
- Users still need indepth knowledge of the Oracle database.
- Users still need a method for performance tuning & analyses.
- Analysis of performance puts extra load on the database.

Check Your Progress 8.2 :

1. ___________is an application used to create computer databases for the Microsoft Windows family of server operating systems.
   (a) Operating System (b) Microsoft SQL Server
   (c) Firewall systems (d) None of above ( )
2. ________provide an extremely flexible method for importing and exporting data between a Microsoft SQL Server installation and a large variety of other formats.
   (a) Data Transformation Services (DTS) (b) Graphics System
   (c) Computer Screen (d) None of above ( )
3. SQL Server Analysis Services adds ________and data mining capabilities for SQL Server databases.
   (a) OLAP (b) Cryptography
   (c) Secret Key Cryptography (d) None of above ( )
4. Oracle is an ___________ product.
   (a) ORDBMS (b) OLAP
   (c) Database (d) None of above ( )
5. Oracle uses ________ to control concurrent access to data.
   (a) Network cards (b) Memory
   (c) Locks (d) Output devices ( )

8.5 Summary

A **database** is a collection of data that is related to a particular topic or purpose. A **database management system** (DBMS) is a system that stores and retrieves information in a database. A DBMS stores data in a table where the entries are filed under a specific category and are properly indexed. A DBMS that is based on relational model is called as RDBMS. Relational model is based on the theory of sets and relations of mathematics. **Microsoft SQL Server** is a relational database server which is developed by Microsoft. MS SQL Server is a software product whose primary function is to store and retrieve data as requested by other software applications from networks or other systems. Microsoft SQL Server 2000 is a full-featured relational database management system (RDBMS) that offers a variety of administrative tools to ease the burdens of database development, maintenance and administration. An **Oracle** database is a collection of data treated as a unit. The purpose of a database is to store and retrieve related information. Oracle Database is the first database designed for enterprise grid computing, the most flexible and cost effective way to manage information and applications. Oracle is an ORDBMS (Object Relational Database Management System) product. This means that Oracle databases model data in terms of objects and their relationships. Data modeled within an Oracle system is divided into conceptual entities, each of which has a set of attributes.

8.6 Answer to Self Learning Exercises

Answer to Check your Progress - 8.1 :

1.	(d)	2.	(a)	3.	(c)
4.	(c)	5.	(d)		

Answer to Check Your Progress - 8.2 :

1.	(b)	2.	(a)	3.	(a)
4.	(a)	5.	(c)		

8.7 Self Assesment Questions

1. What do you understand by 'Relational Database Management System'? How it is differ from DBMS?
2. Write a short note on Microsoft SQL Server with discussion of its strengths.
3. How Oracle is differing from other Relational Database Designs? Discuss some features of Oracle.
4. List and draw a table indicating different versions of SQL Server.
5. What are the basic limitations of Oracle?

Unit - 9 : Introduction to SQL

Structure of the Unit

9.0 Objective

At the end of this unit, you should be able to -

- Describe the various types of SQL
- Describe the different data types and use of SQL
- Describe the DML, DML, DQL and DCL commands of SQL
- Describe the data administrations and TCL commands of SQL
- Describe the various queries of SQL to manipulate data

9.1 Introduction

SQL is a relational database data sublanguage. It is not a complete programming language, but depends on the I/O and control facilities of a host language. It is both a dejure and a de facto standard. ANSI (American Nationa Standards Institute) has published three generations of SQL, as has ISO (International Organization for Standardization). X/Open, a consortium, has also published an SQL specification. Chamberlin and Boyce (1974) published the first paper on what became SQL, based on Codd's mathematical foundation for logical representation and manipulation of data (Codd 1974). In 1978, ANSI began to standardize a data definition language for the network database language then being designed by CODASYL; Technical Committee X3H2 was formed for this project, which soon evolved to encompass the entire network database language, published in 1986 as Databae Language NDL. X3H2 recognized the importance of the relational model and intitated a project based on Chamberlin's work. In cooperation with the corresponding ISO group, the SQL specification was developed and published in 1986. SQL-86 omitted support for referential integrity, but SQL-89 added basic referential integrity. SQL-86 and SQL-89 were rightly criticiezed as inadequate for real applications. In 1922, a major new version, SQL-92, was published, containing features that allowed signifcant applications without vendor extensions (ANSI 1992 ; ISO 1992).

SQL has proved key in the success of relational database management systems and is central to many areas, ragning from traditional MIS applications to scientific reserach. The fourth generation of SQL is currently being prepared. SQL 3 adds significant new facilities, including support for object tehcnology, and is partioned into several parts that can progress independently. See Melton and Simon (1993) for a comprehensive introduction to the SQL language.

9.2 What is SQL?

Structured Query Language (SQL) is the standard language used to communicate with a relational database. The prototype was originally developed by IBM using Dr. E.F. Codd's paper ("A Relational Model of Data for Large Shared Data Banks") as a model. In 1979, not long after IBM's prototype, the first SQL product, ORACLE, was released by Relational Software, Incorporated (which was later renamed Oracle Corporation). Today it is one of the distinguished leaders in relational database technologies.

The American National Standards Institute (ANSI) is an organization that approves certain standards in many different industries. SQL has been deemed the standard language in relational database communication, originally approved in 1986 based on IBM's implementation. In 1987, the ANSI SQL standard was accepted as the international standard by the International Standards Organization (ISO). The standard was revised again in 1992 (SQL-92) and once again in 1999 (SQL-99). The newest standard is now called SQL-2008, which was officially adopted in July of 2008.

Characteristics of SQL

SQL databases tend to be mysterious when it comes to the information that is stored within them. The SQL Database Engine is the core service for storing, processing, and extracting data. The SQL Database Engine provides access and rapid transaction processing to meet the requirements of applications.

The SQL Database Engine can be used to create relational databases for online transaction processing or online analytical processing data. It is likely that you have a number of applications that put information into your various SQL database engines. From this information set, you need to be able to retrieve meaningful information and that is where this course comes in.

9.3 Types of SQL

9.3.1 The Relational Database

A relational database is a database divided into logical units called tables, where tables are related to one another within the database. A relational database allows data to be broken down into logical, smaller, manageable units, enabling easier maintenance and providing more optimal database performance according to the level of organization. In Figure 1, you can see that tables are related to one another through a common key (data value) in a relational database.

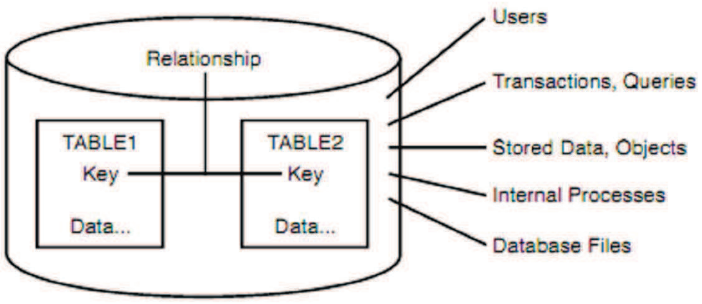


Figure 1. The relational database

Again, tables are related in a relational database, allowing adequate data to be retrieved in a single query (although the desired data may exist in more than one table). By having common keys, or fields, among relational database tables, data from multiple tables can be joined to form one large set of data. As you venture deeper into this book, you see more of a rela- tional database's advantages, including overall performance and easy data access.

9.3.2 Client/Server Technology

In the past, the computer industry was predominately ruled by mainframe computers—large, powerful systems capable of high storage capacity and high data processing capabilities. Users communicated with the mainframe through dumb terminals—terminals that did not think on their own but relied solely on the mainframe's CPU, storage, and memory. Each terminal had a data line attached to the mainframe. The mainframe environment definitely served its purpose and does today in many businesses, but a greater technology was soon to be introduced: the client/server model.

In the client/server system, the main computer, called the server, is accessible from a network—typically a local area network (LAN) or a wide area network (WAN). The server is normally accessed by personal computers (PCs) or by other servers, instead of dumb terminals. Each PC, called a client, is provided access to the network, allowing communication between the client and the server, thus explaining the name client/server. The main difference between client/server and mainframe environments is that the user's PC in a client/server environment is capable of thinking on its own, capable of running its own processes using its own CPU and memory, but readily accessible to a server computer through a network. In most cases, a client/server system is much more flexible for today's overall business needs and is much preferred.

Modern database systems reside on various types of computer systems with various operating systems. The most common types of operating systems are Windows-based systems, Linux, and command-line systems such as UNIX. Databases reside mainly in client/server and web environments. A lack of training and experience is the main reason for failed implementations of database systems. Nevertheless, an understanding of the client/server model and web-based systems, which will be explained in the next section, is imperative with the rising (and sometimes unreasonable) demands placed on today's businesses as well as the development of Internet technologies and network computing. Figure 2 illustrates the concept of client/server technology.

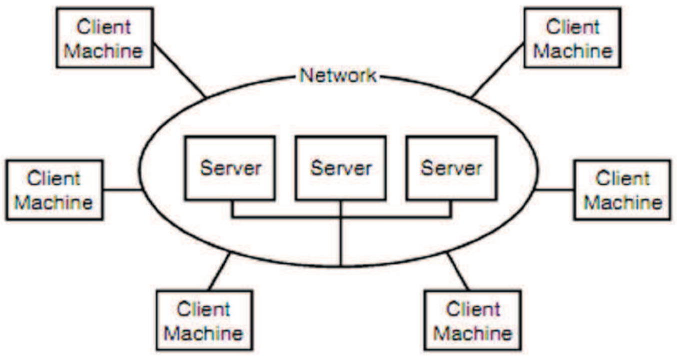


Figure 2. The client/server model.

9.3.3 Web-Based Database System

Business information systems are moving toward web integration. Databases are now accessible through the Internet, meaning that customers' access to an organization's information is enabled through an Internet browser such as Internet Explorer or Firefox. Customers (users of data) are able to order merchandise, check on inventories, check on the status of orders, make administrative changes to accounts, transfer money from one account to another, and so forth. A customer simply invokes an Internet browser, goes to the organization's website, logs in (if required by the organization), and uses an application built into the organization's web page to access data. Most organizations require users to register with them and issue a login and password to the customer. Of course, many things occur behind the scenes when a database is being accessed via a web browser. SQL, for instance, can be executed by the web application. This executed SQL is used to access the organization's database, return data to the web server, and then return that data to the customer's Internet browser.

The basic structure of a web-based database system is similar to that of a client-server system from a user's standpoint (refer to Figure 1.3). Each user has a client machine, which has a connection to the Internet and contains a web browser. The network in Figure 1.3 (in the case of a web-based database) just happens to be the Internet, as opposed to a local network. For the most part, a client is still accessing a server for information. It doesn't matter that the server might exist in another state or even another country. The main point of web-based database systems is to expand the potential customer base of a database system that knows no physical location bounds, thus increasing data availability and an organization's customer base.

9.4 Advantages of SQL

Programming using static SQL requires less effort than using embedded dynamic SQL. Static SQL statements are simply embedded into the host language source file, and the precompiler handles the necessary conversion to database manager run-time services API calls that the host language compiler can process.

Because the authorization of the person binding the application is used, the end user does not require direct privileges to execute the statements in the package. For example, an application could allow a user to update parts of a table without granting an update privilege on the entire table. This can be achieved by restricting the static SQL statements to allow updates only to certain columns or to a range of values.

Static SQL statements are persistent, meaning that the statements last for as long as the package exists.

Dynamic SQL statem ents are cached until they are either invalidated, freed for space management reasons, or the database is shut down. If required, the dynamic SQL statements are recompiled implicitly by the DB2$^{(R)}$ SQL compiler whenever a cached statement becomes invalid.

The key advantage of static SQL, with respect to persistence, is that the static statements exist after a particular database is shut down, whereas dynamic SQL statements cease to exist when this occurs. In addition, static SQL does not have to be compiled by the DB2 SQL compiler at run time, while dynamic SQL must be explicitly compiled at run time (for example, by using the PREPARE statement). Because DB2 caches dynamic SQL statements, the statements do not need to be compiled often by DB2, but they must be compiled at least once when you execute the application.

There can be performance advantages to static SQL. For simple, short-running SQL programs, a static SQL statement executes faster than the same statement processed dynamically because the overhead of preparing an executable form of the statement is done at precompile time instead of at run time.

9.5 SQL Data Types and Literals

The following sections discuss the basic data types supported by ANSI SQL. Data types are characteristics of the data itself, whose attributes are placed on fields within a table. For example, you can specify that a field must contain numeric values, disallowing the entering of alphanumeric strings. After all, you would not want to enter alphabetic characters in a field for a dollar amount. Defining each field in the database with a data type eliminates much of the incorrect data found in a database due to data entry errors. Field definition (data type definition) is a form of data validation that controls the type of data that may be entered into each given field. Depending on your implementation of relational database management sys- tem (RDBMS), certain data types can be converted automatically to other data types depending upon their format. This type of conversion in known as an implicit conversion, which means that the database handles the con- version for you. An example of this is taking a numeric value of 1000.92 from a numeric field and inputting it into a string field. Other data types cannot be converted implicitly by the host RDBMS and therefore must undergo an explicit conversion. This usually involves the use of an SQL function, such as CAST or CONVERT. For example

SELECT CAST('12/27/1974' AS DATETIME) AS MYDATE

The very basic data types, as with most other languages, are

- ✓ String types
- ✓ Numeric types
- ✓ Date and time types

Fixed-Length Strings :

Constant characters, those strings that always have the same length, are stored using a fixed-length data type. The following is the standard for an SQL fixed-length character:

Character (n) :

n represents a number identifying the allocated or maximum length of the particular field with this definition.

Some implementations of SQL use the CHAR data type to store fixed-length data. You can store alphanumeric data in this data type. An example of a constant length data type would be for a state abbreviation because all state abbreviations are two characters.

Spaces are normally used to fill extra spots when using a fixed-length data type; if a field's length was set to 10 and data entered filled only 5 places, the remaining 5 spaces would be recorded as spaces. The padding of spaces ensures that each value in a field is a fixed length.

Varying-Length Strings :

SQL supports the use of varying-length strings, strings whose length is not constant for all data. The following is the standard for an SQL varying- length character:

Character Varying (n) :

n represents a number identifying the allocated or maximum length of the particular field with this definition.

Common data types for variable-length character values are the VARCHAR, VARBINARY, and VARCHAR2 data types. VARCHAR is the ANSI standard, which Microsoft SQL Server and MySQL use; Oracle uses both VARCHAR and VARCHAR2. The data stored in a character-defined column can be alphanumeric, which means that the data value may contain numeric characters. VARBINARY is similar to VARCHAR and VARCHAR2 except that it contains a variable length of bytes. Normally, you would use a type such as this to store some kind of digital data such as possibly an image file.

Remember that fixed-length data types typically pad spaces to fill in allocated places not used by the field. The varying-length data type does not work this way. For instance, if the allocated length of a varying-length field is 10, and a string of 5 characters is entered, the total length of that particular value would be only 5. Spaces are not used to fill unused places in a column.

9.6 Large Object Types

Some variable-length data types need to hold longer lengths of data than what is traditionally reserved for a VARCHAR field. The BLOB and TEXT data types are two examples of such data types in modern database implementations. These data types are specifically made to hold large sets of data. The BLOB is a binary large object, so its data is treated as a large binary string (a byte string). A BLOB is especially useful in an implementation that needs to store binary media files in the database, such as images or MP3s. The TEXT data type is a large character string data type that can be treated as a large VARCHAR field. It is often used when an implementation needs to store large sets of character data in the database. An example of this would be storing HTML input from the entries of a blog site. Storing this type of data in the database enables the site to be dynamically updated.

9.6.1 Numeric Types

Numeric values are stored in fields that are defined as some type of number, typically referred to as NUMBER, INTEGER, REAL, DECIMAL, and so on. The following are the standards for SQL numeric values:

- BIT(n)
- BIT VARYING(n)
- DECIMAL(p,s)
- INTEGER
- SMALLINT
- BIGINT
- FLOAT(p,s)
- DOUBLE PRECISION(p,s)
- REAL(s)

p represents a number identifying the allocated or maximum length of the particular field for each appropriate definition.

s is a number to the right of the decimal point, such as 34.ss.

A common numeric data type in SQL implementations is NUMERIC, which accommodates the direction for numeric values provided by ANSI. Numeric values can be stored as zero, positive, negative, fixed, and floating-point numbers. The following is an example using NUMERIC:

NUMERIC(5)

This example restricts the maximum value entered in a particular field to 99999. Note that all the database implementations that we use for the examples support the NUMERIC type but implement it as a DECIMAL.

9.6.2 Decimal Types

Decimal values are numeric values that include the use of a decimal point. The standard for a decimal in SQL follows, where p is the precision and s is the decimal's scale:

DECIMAL(p,s)

The precision is the total length of the numeric value. In a numeric defined DECIMAL(4,2), the precision is 4, which is the total length allocated for a numeric value. The scale is the number of digits to the right of the decimal point. The scale is 2 in the previous DECIMAL(4,2) example. If a value has more places to the right side of the decimal point than the scale allows, the value is rounded; for instance, 34.33 inserted into a DECIMAL(3,1) is typically rounded to 34.3. If a numeric value was defined as the following data type, the maximum value allowed would be 99.99:

DECIMAL(4,2)

The precision is 4, which represents the total length allocated for an associated value. The scale is 2, which represents the number of places, or bytes, reserved to the right side of the decimal point. The decimal point does not count as a character.

Allowed values for a column defined as DECIMAL(4,2) include the following:

- 12
- 12.4
- 2.44
- 12.449

The last numeric value, 12.449, is rounded off to 12.45 upon input into the column. In this case, any numbers between 12.445 and 12.449 would be rounded to 12.45.

9.6.3 Integers

An integer is a numeric value that does not contain a decimal, only whole numbers (both positive and negative).

Valid integers include the following:

- 1
- 0
- –1
- 99
- –99
- 199

9.6.4 Floating-Point Decimals

Floating-point decimals are decimal values whose precision and scale are variable lengths and virtually without limit. Any precision and scale is acceptable. The REAL data type designates a column with single-precision, floating-point numbers. The DOUBLE PRECISION data type designates a column that contains double-precision, floating-point numbers. To be considered a single-precision floating point, the precision must be between 1 and 21 inclusive. To be considered a double-precision floating point, the precision must be between 22 and 53 inclusive. The following are examples of the FLOAT data type:

- FLOAT
- FLOAT(15)
- FLOAT(50)

9.6.5 Date and Time Types

Date and time data types are quite obviously used to keep track of information concerning dates and time. Standard SQL supports what are called DATETIME data types, which include the following specific data types:

- DATE
- TIME
- DATETIME
- TIMESTAMP

The elements of a DATETIME data type consist of the following:

- YEAR
- MONTH
- DAY
- HOUR
- MINUTE
- SECOND

9.6.6 NULL Data Types

As you should know from Hour 1, a NULL value is a missing value or a column in a row of data that has not been assigned a value. NULL values are used in nearly all parts of SQL, including the creation of tables, search conditions for queries, and even in literal strings.

The following are two methods for referencing a NULL value:

- **NULL (the keyword NULL itself)**

The following does not represent a NULL value, but a literal string containing the characters N-U-L-L:

'NULL' When using the NULL data type, it is important to realize that data is not required in a particular field. If data is always required for a given field, always use NOT NULL with a data type. If there is a chance that there might not always be data for a field, it is better to use NULL.

9.6.7 User-Defined Types

A user-defined type is a data type that the user defines. User-defined types allow users to customize their own data types to meet data storage needs and are based on existing data types. User-defined data types can assist the developer by providing greater flexibility during database application development because they maximize the number of possibilities for data storage. The CREATE TYPE statement is used to create a user-defined type.

For example, you can create a type as follows in both MySQL and Oracle:

```
CREATE TYPE PERSON AS OBJECT
(NAME      VARCHAR (30),
 SSN       VARCHAR (9));
```

You can reference your user-defined type as follows:

```
CREATE TABLE EMP_PAY
(EMPLOYEE   PERSON,
 SALARY     DECIMAL(10,2),
 HIRE_DATE  DATE);
```

Notice that the data type referenced for the first column EMPLOYEE is PERSON. PERSON is the user-defined type you created in the first example.

9.7 Types of SQL Commands

The following sections discuss the basic categories of commands used in SQL to perform various functions. These functions include building database objects, manipulating objects, populating database tables with data, updating existing data in tables, deleting data, performing database queries, controlling database access, and overall database administration.

The main categories are

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Query Language (DQL)
- Data Control Language (DCL)
- Data administration commands
- Transactional control commands

9.7.1 Data Definition Language (DDL)

Data Definition Language (DDL) is the part of SQL that enables a database user to create and restructure database objects, such as the creation or the deletion of a table.

Some of the most fundamental DDL commands discussed during the following hours include

- CREATE TABLE
- ALTER TABLE
- DROP TABLE
- CREATE INDEX
- ALTER INDEX
- DROP INDEX
- CREATE VIEW
- DROP VIEW

1. **<u>Create Table:</u>** This command is used to create a new table in a database.

The SQL syntax for CREATE TABLE is

```
CREATE TABLE "table_name"
("column 1" "data_type_for_column_1",
"column 2" "data_type_for_column_2",
... )
```

So, if we are to create the customer table specified as above, we would type in

```
CREATE TABLE customer
(First_Name char(50),
Last_Name char(50),
Address char(50),
City char(50),
Country char(25),
Birth_Date date)
```

2. **<u>Alter table:</u>**

The ALTER TABLE statement is used to add or drop columns in an existing table.

```
ALTER TABLE table_name
ADD column_name datatype
```

```
ALTER TABLE table_name
DROP COLUMN column_name
```

Person:

LastName	**FirstName**	**Address**
Pettersen	Kari	Storgt 20

Example

To add a column named "City" in the "Person" table:

```
ALTER TABLE Person ADD City varchar(30)
```

Result:

LastName	**FirstName**	**Address**	**City**
Pettersen	Kari	Storgt 20	

3. **Drop Table:** This command is used to drop a table from a database.

The syntax for **drop table** is

DROP TABLE "table_name"

4. **Create Index:**

Indices are created in an existing table to locate rows more quickly and efficiently. It is possible to create an index on one or more columns of a table, and each index is given a name. The users cannot see the indexes, they are just used to speed up queries.

Note: Updating a table containing indexes takes more time than updating a table without, this is because the indexes also need an update. So, it is a good idea to create indexes only on columns that are often used for a search.

A Unique Index

Creates a unique index on a table. A unique index means that two rows cannot have the same index value.

```
CREATE UNIQUE INDEX index_name
ON table_name (column_name)
```

The "column_name" specifies the column you want indexed.

A Simple Index

Creates a simple index on a table. When the UNIQUE keyword is omitted, duplicate values are allowed.

```
CREATE INDEX index_name
ON table_name (column_name)
```

The "column_name" specifies the column you want indexed.

Example

This example creates a simple index, named "PersonIndex", on the LastName field of the Person table:

```
CREATE INDEX PersonIndex
ON Person (LastName)
```

If you want to index the values in a column in descending order, you can add the reserved word DESC after the column name:

```
CREATE INDEX PersonIndex
ON Person (LastName DESC)
```

If you want to index more than one column you can list the column names within the parentheses, separated by commas:

```
CREATE INDEX PersonIndex
ON Person (LastName, FirstName)
```

5. Drop Index

You can delete an existing index in a table with the DROP INDEX statement.

Syntax for Microsoft SQLJet (and Microsoft Access):

```
DROP INDEX index_name ON table_name
```

Syntax for MS SQL Server:

```
DROP INDEX table_name.index_name
```

Syntax for IBM DB2 and Oracle:

```
DROP INDEX index_name
```

Syntax for MySQL:

```
ALTER TABLE table_name DROP INDEX index_name
```

6. Create View

In SQL, a VIEW is a virtual table based on the result-set of a SELECT statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database. You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from a single table.

Note: The database design and structure will NOT be affected by the functions, where, or join statements in a view.

Syntax

```
CREATE VIEW view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition
```

Note: The database does not store the view data. The database engine recreates the data, using the view's SELECT statement, every time a user queries a view.

Using Views

A view could be used from inside a query, a stored procedure, or from inside another view. By adding functions, joins, etc., to a view, it allows you to present exactly the data you want to the user. The sample database Northwind has some views installed by default. The view "Current Product List" lists all active products (products that are not discontinued) from the Products table. The view is created with the following SQL:

```
CREATE VIEW [Current Product List] AS
SELECT ProductID,ProductName
FROM Products
WHERE Discontinued=No
```

We can query the view above as follows:

```
SELECT * FROM [Current Product List]
```

Another view from the Northwind sample database selects every product in the Products table that has a unit price that is higher than the average unit price:

```
        CREATE VIEW [Products Above Average Price] AS
        SELECT ProductName,UnitPrice
        FROM Products
WHERE UnitPrice>(SELECT AVG(UnitPrice) FROM Products)
```

We can query the view above as follows:

```
        SELECT * FROM [Products Above Average Price]
```

Another example view from the Northwind database calculates the total sale for each category in 1997. Note that this view select its data from another view called "Product Sales for 1997":

```
        CREATE VIEW [Category Sales For 1997] AS
        SELECT DISTINCT CategoryName,Sum(ProductSales) AS CategorySales
        FROM [Product Sales for 1997]
        GROUP BY CategoryName
```

We can query the view above as follows:

```
        SELECT * FROM [Category Sales For 1997]
```

We can also add a condition to the query. Now we want to see the total sale only for the category "Beverages":

```
        SELECT * FROM [Category Sales For 1997]
WHERE CategoryName='Beverages'
```

9.7.2 Data Manipulation Language (DML)

Data Manipulation Language (DML) is the part of SQL used to manipulate data within objects of a relational database.

The three basic DML commands are

- INSERT
- UPDATE
- DELETE

1. INSERT Statement

The INSERT Statement adds one or more rows to a table. It has two formats:

INSERT INTO table-1 [(column-list)] VALUES (value-list)

and,

INSERT INTO table-1 [(column-list)] (query-specification)

The first form inserts a single row into table-1 and explicitly specifies the column values for the row. The second form uses the result of query-specification to insert one or more rows into table-1. The result rows from the query are the rows added to the insert table.

Both forms have an optional column-list specification. Only the columns listed will be assigned values. Unlisted columns are set to null, so unlisted columns must allow nulls. The values from the VALUES Clause (first form) or the columns from the query-specification rows (second form) are assigned to the corresponding column in column-list in order.

If the optional column-list is missing, the default column list is substituted. The default column list contains all columns in table-1 in the order they were declared in CREATE TABLE, or CREATE VIEW.

VALUES Clause

The VALUES Clause in the INSERT Statement provides a set of values to place in the columns of a new row. It has the following general format:

VALUES (value-1 [, value-2] ...)

value-1 and value-2 are Literal Values or Scalar Expressions involving literals. They can also specify NULL.

The values list in the VALUES clause must match the explicit or implicit column list for INSERT in degree (number of items). They must also match the data type of corresponding column or be convertible to that data type.

INSERT Examples

INSERT INTO p (pno, color) VALUES ('P4', 'Brown')

Before				**After**		
pno	**descr**	**color**		**pno**	**descr**	**color**
P1	Widget	Blue		P1	Widget	Blue
P2	Widget	Red	=>	P2	Widget	Red
P3	Dongle	Green		P3	Dongle	Green
				P4	NULL	Brown

INSERT INTO sp

SELECT s.sno, p.pno, 500

FROM s, p

WHERE p.color='Green' AND s.city='London'

Before				**After**		
sno	**pno**	**qty**		**sno**	**pno**	**qty**
S1	P1	NULL		S1	P1	NULL
S2	P1	200	=>	S2	P1	200
S3	P1	1000		S3	P1	1000
S3	P2	200		S3	P2	200
				S2	P3	500

2. UPDATE Statement

The UPDATE statement modifies columns in selected table rows. It has the following general format:

UPDATE table-1 SET set-list [WHERE predicate]

The optional WHERE Clause has the same format as in the SELECT Statement. See WHERE Clause. The WHERE clause chooses which table rows to update. If it is missing, all rows are in table-1 are updated.

The set-list contains assignments of new values for selected columns.

The SET Clause expressions and WHERE Clause predicate can contain subqueries, but the subqueries cannot reference table-1. This prevents situations where results are dependent on the order of processing.

SET Clause

The SET Clause in the UPDATE Statement updates (assigns new value to) columns in the selected table rows. It has the following general format:

SET column-1 = value-1 [, column-2 = value-2] ...

column-1 and column-2 are columns in the Update table. value-1 and value-2 are expressions that can reference columns from the update table. They also can be the keyword — NULL, to set the column to null.

Since the assignment expressions can reference columns from the current row, the expressions are evaluated first. After the values of all Set expressions have been computed, they are then assigned to the referenced columns. This avoids results dependent on the order of processing.

UPDATE Examples

UPDATE sp SET qty = qty + 20

Before				After		
sno.	**pno**	**qty**		**sno.**	**pno**	**qty**
S1	P1	NULL		S1	P1	NULL
S2	P1	200	=>	S2	P1	220
S3	P1	1000		S3	P1	1020
S3	P2	200		S3	P2	220

UPDATE s

SET name = 'Tony', city = 'Milan'

WHERE sno = 'S3'

Before				After		
sno	**name**	**city**		**sno**	**name**	**city**
S1	Pierre	Paris		S1	Pierre	Paris
S2	John	London	=>	S2	John	London
S3	Mario	Rome		S3	Tony	Milan

3. <u>**DELETE Statement**</u>

The DELETE Statement removes selected rows from a table. It has the following general format:

DELETE FROM table-1 [WHERE predicate]

The optional WHERE Clause has the same format as in the SELECT Statement. See WHERE Clause. The WHERE clause chooses which table rows to delete. If it is missing, all rows are in table-1 are removed.

The WHERE Clause predicate can contain subqueries, but the subqueries cannot reference table-1. This prevents situations where results are dependent on the order of processing.

DELETE Examples

DELETE FROM sp WHERE pno = 'P1'

Before				After		
sno	**pno**	**qty**		**sno**	**pno**	**qty**
S1	P1	NULL		S3	P2	200
S2	P1	200	=>			
S3	P1	1000				
S3	P2	200				

DELETE FROM p WHERE pno NOT IN (SELECT pno FROM sp)

Before				After		
pno	**descr**	**color**		**pno**	**descr**	**color**
P1	Widget	Blue		P1	Widget	Blue
P2	Widget	Red	=>	P2	Widget	Red
P3	Dongle	Green				

9.7.3 Data Query Language (DQL)

Though comprised of only one command, Data Query Language (DQL) is the most concentrated focus of SQL for modern relational database users. The base command is SELECT.

This command, accompanied by many options and clauses, is used to compose queries against a relational database. A query is an inquiry to the database for information. A query is usually issued to the database through an application interface or via a command-line prompt. You can easily create queries, from simple to complex, from vague to specific.

The SELECT statement is used to select data from a table. The tabular result is stored in a result table (called the result-set).

Syntax

```
        SELECT column_name(s)
FROM table_name
```

Note: SQL statements are not case sensitive. SELECT is the same as select.

SQL SELECT Example

To select the content of columns named "LastName" and "FirstName", from the database table called "Persons", use a SELECT statement like this:

```
        SELECT LastName,FirstName FROM Persons
```

The database table "Persons":

LastName	**FirstName**	**Address**	**City**
Hansen Ola	Timoteivn 10	Sandnes	
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

The result

LastName	**FirstName**
Hansen Ola	
Svendson	Tove
Pettersen	Kari

Select All Columns

To select all columns from the "Persons" table, use a * symbol instead of column names, like this:

```
        SELECT * FROM Persons
```

Result

LastName	**FirstName**	**Address**	**City**
Hansen Ola	Timoteivn 10	Sandnes	
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

9.7.4 Data Control Commands

Data control commands in SQL enable you to control access to data within the database. These Data Control Language (DCL) commands are normally used to create objects related to user access and also control the distribution of privileges among users. Some data control commands are as follows:

- ALTER PASSWORD
- GRANT

- REVOKE
- CREATE SYNONYM

You will find that these commands are often grouped with other commands and might appear in a number of lessons throughout this book. Data Administration Commands Data administration commands enable the user to perform audits and perform analyses on operations within the database. They can also be used to help analyze system performance. Two general data administration commands are as follows:

- START AUDIT
- STOP AUDIT

Do not get data administration confused with database administration. Database administration is the overall administration of a database, which envelops the use of all levels of commands. Data administration is much more specific to each SQL implementation than are those core commands of the SQL language.

SQL-Transaction Statements

SQL-Transaction Statements control transactions in database access. This subset of SQL is also called the Data Control Language for SQL (SQL DCL).

There are 2 SQL-Transaction Statements:

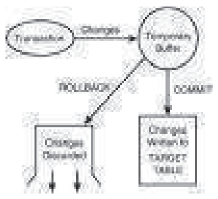
- COMMIT Statement — commit (make persistent) all changes for the current transaction
- ROLLBACK Statement — roll back (rescind) all changes for the current transaction

9.7.5 Transactional Control Commands

Transactional control is the ability to manage various transactions that may occur within a relational database management system. When a transaction is executed and completes successfully, the target table is not immediately changed, although it may appear so according to the output. When a transaction successfully completes, there are transactional control commands that are used to finalize the transaction, either saving the changes made by the transaction to the database or reversing the changes made by the transaction.

There are three commands used to control transactions:

- COMMIT
- ROLLBACK
- SAVEPOINT
- The SET TRANSACTION Command
- Transactional control commands are only used with the DML commands INSERT, UPDATE, and DELETE. For example, you do not issue a COMMIT statement after creating a table. When the table is created, it is automatically committed to the database. Likewise, you cannot issue a ROLLBACK to replenish a table that was just dropped.
- When a transaction has completed, the transactional information is stored either in an allocated area or in a temporary rollback area in the database. All changes are held in this temporary rollback area until a transactional control command is issued. When a transactional control command is issued, changes are either made to the database or discarded; then, the temporary rollback area is emptied. Figure 6.1 illustrates how changes are applied to a relational database.



The COMMIT Command

The COMMIT command is the transactional command used to save changes invoked by a transaction to the database. The COMMITcommand saves all transactions to the database since the last COMMIT or ROLLBACK command.

COMMIT [WORK];

The keyword COMMIT is the only mandatory part of the syntax, along with the character or command used to terminate a statement according to each implementation. WORK is a keyword that is completely optional; its only purpose is to make the command more user-friendly.

The ROLLBACK Command

The ROLLBACK command is the transactional control command used to undo transactions that have not already been saved to the database. The ROLLBACK command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.

The syntax for the ROLLBACK command is as follows:

rollback [work];

Once again, the COMMIT statement, the WORK keyword is an optional part of the ROLLBACK syntax.

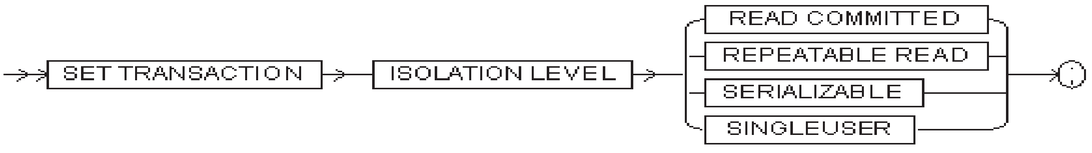
The SAVEPOINT Command

A SAVEPOINT is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction.

The syntax for the SAVEPOINT command is

This command serves only in the creation of a SAVEPOINT among transactional statements. The ROLLBACK command is used to undo a group of transactions. The SAVEPOINT is a way of managing transactions by breaking large numbers of transactions into smaller, more manageable groups.

The SET TRANSACTION Command



The SET TRANSACTION Command Establishes the isolation level of the current transaction. If you use a SET TRANSACTION statement, it must be the first statement in your transaction. However, a transaction need not have a SET TRANSACTION statement.

9.8 Summary

SQL (Structured Query Language) is a database sublanguage for querying and modifying relational databases. It was developed by IBM Research in the mid 70's and standardized by ANSI in 1986. SQL is a version of Relational Calculus. The basic structure in SQL in the statement. Semicolons separate multiple SQL statements.

9.9 Self-Assessment Questions

1. How DDL commands are different from DML? Explain with the help of suitable example.
2. Explain the different data types of SQL with the help of example.
3. What do you understand by sub queries? Give example and explain.
4. Explain the use of join in RDBMS with the help of example.
5. Discuss the advantages of SQL over old database software packages.

□□□

Unit - 10 : More on SQL

Structure of the Unit

10.0 Objective

At the end of this unit, you should be able to -

- Describe the various aggregate functions of SQL
- Describe the join in SQL
- Describe the set operations in SQL

10.1 Introduction

In previous section we have only focused on queries that refer to exactly one table. Furthermore, conditions in a where were restricted to simple comparisons. A major feature of relational databases, however, is to combine (join) tuples stored in different tables in order to display more meaningful and complete information.

10.2 Aggregate Functions

SQL has a lot of built-in functions for counting and calculations.

Function Syntax

The syntax for built-in SQL functions is:

SELECT function(column) FROM table

Aggregate functions operate against a collection of values, but return a single value.

Note : If used among many other expressions in the item list of a SELECT statement, the SELECT must have a GROUP BY clause!!

Five important aggregate functions: SUM, AVG, MAX, MIN, and COUNT.

They are calledaggregate functions because they summarize the results of a query, rather than listing all of the rows.

- SUM () gives the total of all the rows, satisfying any conditions, of the given column, where the givencolumn is numeric.
- AVG () gives the average of the given column.
- MAX () gives the largest figure in the given column.
- MIN () gives the smallest figure in the given column.
- COUNT(*) gives the number of rows satisfying the conditions.

Examples 1 :

```
SELECT SUM(SALARY), AVG(SALARY)
FROM EMPLOYEESTABLE;
```

This query shows the total of all salaries in the table, and the average salary of all of the entries in the table.

```
SELECT MIN(BENEFITS)
FROM EMPLOYEESTABLE
WHERE POSITION = 'Manager';
```

This query gives the smallest figure of the Benefits column, of the employees who are Managers, which is 12500.

```
SELECT COUNT(*)
FROM EMPLOYEESTABLE
WHERE POSITION = 'Staff';
```

This query tells you how many employees have Staff status.

Aggregate functions (like SUM) often need an added GROUP BY functionality.

10.3 GROUP BY Clause

GROUP BY Clause is added to SQL because aggregate functions (like SUM) return the aggregate of all column values every time they are called, and without the GROUP BY function it was impossible to find the sum for each individual group of column values.

The syntax for the GROUP BY function is:

```
SELECT column,SUM(column) FROM table GROUP BY column
```

GROUP BY Example

This "Sales" Table :

Company	Amount
TVS	5500
IBM	4500
TVS	7100

And This SQL :

```
SELECT Company, SUM(Amount) FROM Sales
```

Returns this result :

Company	SUM(Amount)
TVS	17100
IBM	17100
TVS	17100

The above code is invalid because the column returned is not part of an aggregate. A GROUP BY clause will solve this problem:

SELECT Company,SUM(Amount) FROM Sales

GROUP BY Company

Returns this result :

Company	SUM(Amount)
TVS	12600
IBM	4500

10.4 HAVING Clause

Having clause can also be added to SQL because the WHERE keyword could not be used against aggregate functions (like SUM), and without HAVING, it would be impossible to test for result conditions.

The syntax for the HAVING function is:

SELECT column,SUM(column) FROM table

GROUP BY column

HAVING SUM(column) condition value

This "Sales" Table :

Company	Amount
TVS	5500
IBM	4500
TVS	7100

This SQL :

SELECT Company,SUM(Amount) FROM Sales

GROUP BY Company

HAVING SUM(Amount)>10000

Returns this result :

Company	SUM(Amount)
TVS	12600

10.5 ORDER BY Clause

The ORDER BY clause is optional. If used, it must be the last clause in the SELECT statement. The ORDER BY clause requests sorting for the results of a query.

When the ORDER BY clause is missing, the result rows from a query have no defined order (they are *unordered*). The ORDER BY clause defines the ordering of rows based on columns from the SELECT clause. The ORDER BY clause has the following general format:

ORDER BY column-1 [ASC|DESC] [column-2 [ASC|DESC]] ...

column-1, *column-2*, ... are column names specified (or implied) in the select list. If a select column is renamed (given a new name in the select entry), the new name is used in the ORDER BY list. ASC and DESC request ascending or descending sort for a column. ASC is the default.

ORDER BY sorts rows using the ordering columns in left-to-right, major-to-minor order. The rows are sorted first on the first column name in the list. If there are any duplicate values for the first column, the duplicates are sorted on the second column (within the first column sort) in the Order By list, and so on. There is no defined inner ordering for rows that have duplicate values for all Order By columns.

Database *nulls* require special processing in ORDER BY. A *null* column sorts higher than all regular values; this is reversed for DESC.

In sorting, *nulls* are considered duplicates of each other for ORDER BY. Sorting on *hidden* information makes no sense in utilizing the results of a query. This is also why SQL only allows select list columns in ORDER BY.

For convenience when using expressions in the select list, select items can be specified by number (starting with 1). Names and numbers can be intermixed.

Example queries :

SELECT * FROM sp ORDER BY 3 DESC

sno	**pno**	**qty**
S1	P1	NULL
S3	P1	1000
S3	P2	200
S2	P1	200

SELECT name, city FROM s ORDER BY name

name	**city**
John	London
Mario	Rome
Pierre	Paris

SELECT * FROM sp ORDER BY qty DESC, sno

sno	pno	qty
S1	P1	NULL
S3	P1	1000
S2	P1	200
S3	P2	200

Orders table :

Company	OrderNumber
Sega	3412
ABC Shop	5678
TVS	2312
TVS	6798

Example :

To display the company names in alphabetical order:

SELECT Company, OrderNumber FROM Orders

ORDER BY Company ASC (asending)

Result:

Company	OrderNumber
ABC Shop	5678
Sega	3412
TVS	6798
TVS	2312

Example :

To display the company names in alphabetical order AND the OrderNumber in numerical order :

SELECT Company, OrderNumber FROM Orders

ORDER BY Company, OrderNumber

Result:

Company	OrderNumber
ABC Shop	5678
Sega	3412
TVS	2312
TVS	6798

Example :

To display the company names in reverse alphabetical order:

SELECT Company, OrderNumber FROM Orders

ORDER BY Company DESC

Result:

Company	OrderNumber
TVS	6798
TVS	2312
Sega	3412
ABC Shop	5678

Example :

To display the company names in reverse alphabetical order AND the OrderNumber in numerical order:

SELECT Company, OrderNumber FROM Orders

ORDER BY Company DESC, OrderNumber ASC

Result:

Company	OrderNumber
TVS	2312
TVS	6798
Sega	3412
ABC Shop	5678

Notice that there are two equal company names (TVS) in the result above. The only time you will see the second column in ASC order would be when there are duplicated values in the first sort column, or a handful of nulls.The ORDER BY keyword is used to sort the result.

Sort the Rows :

The ORDER BY clause is used to sort the rows.

Orders :

Company	OrderNumber
Sega	3412
ABC Shop	5678
TVS	2312
TVS	6798

Example :

To display the company names in alphabetical order:

SELECT Company, OrderNumber FROM Orders

ORDER BY Company

Result :

Company	OrderNumber
ABC Shop	5678
Sega	3412
TVS	6798
TVS	2312

Example :

To display the company names in alphabetical order AND the OrderNumber in numerical order:

SELECT Company, OrderNumber FROM Orders

ORDER BY Company, OrderNumber

Result :

Company	OrderNumber
ABC Shop	5678
Sega	3412
TVS	2312
TVS	6798

Example :

To display the company names in reverse alphabetical order:

SELECT Company, OrderNumber FROM Orders

ORDER BY Company DESC

Result :

Company	OrderNumber
TVS	6798
TVS	2312
Sega	3412
ABC Shop	5678

Example :

To display the company names in reverse alphabetical order AND the OrderNumber in numerical order:

SELECT Company, OrderNumber FROM Orders

ORDER BY Company DESC, OrderNumber ASC

Result :

Company	OrderNumber
TVS	2312
TVS	6798
Sega	3412
ABC Shop	5678

Notice that there are two equal company names (TVS) in the result above. The only time you will see the second column in ASC order would be when there are duplicated values in the first sort column, or a handful of nulls.

10.6 Join

The FROM clause allows more than 1 table in its list, however simply listing more than one table will very rarely produce the expected results. The rows from one table must be correlated with the rows of the others. This correlation is known as joining.

s Table

sno	name	city
S1	Pierre	Paris
S2	John	London
S3	Mario	Rome

sp Table

sno	pno	qty
S1	P1	NULL
S2	P1	200
S3	P1	1000
S3	P2	200

An example can best illustrate the rationale behind joins. The following query:

SELECT * FROM sp, p

Produces :

sno	pno	qty	pno	descr	color
S1	P1	NULL	P1	Widget	Blue
S1	P1	NULL	P2	Widget	Red
S1	P1	NULL	P3	Dongle	Green
S2	P1	200	P1	Widget	Blue
S2	P1	200	P2	Widget	Red
S2	P1	200	P3	Dongle	Green
S3	P1	1000	P1	Widget	Blue
S3	P1	1000	P2	Widget	Red
S3	P1	1000	P3	Dongle	Green
S3	P2	200	P1	Widget	Blue
S3	P2	200	P2	Widget	Red
S3	P2	200	P3	Dongle	Green

Each row in sp is arbitrarily combined with each row in p, giving 12 result rows (4 rows in sp X 3 rows in p.) This is known as a cartesian product.

A more usable query would correlate the rows from sp with rows from p, for instance matching on the common column — pno:

```
SELECT *
FROM sp, p
WHERE sp.pno = p.pno
```

This Produces :

sno	pno	qty	pno	descr	color
S1	P1	NULL	P1	Widget	Blue
S2	P1	200	P1	Widget	Blue
S3	P1	1000	P1	Widget	Blue
S3	P2	200	P2	Widget	Red

Rows for each part in p are combined with rows in sp for the same part by matching on part number (pno). In this query, the WHERE Clause provides the join predicate, matching pno from p with pno from sp.

The join in this example is known as an innerequi-join. equi meaning that the join predicate uses = (equals) to match the join columns. Other types of joins use different comparison operators. For example, a query might use a greater-than join.

The term inner means only rows that match are included. Rows in the first table that have no matching rows in the second table are excluded and vice versa (in the above join, the row in p with pno P3 is not included in the result.) An outer join includes unmatched rows in the result.

More than 2 tables can participate in a join. This is basically just an extension of a 2 table join. 3 tables — a, b, c, might be joined in various ways:

- a joins b which joins c
- a joins b and the join of a and b joins c
- a joins b and a joins c

Plus several other variations. With inner joins, this structure is not explicit. It is implicit in the nature of the join predicates. With outer joins, it is explicit;

This query performs a 3 table join:

```
SELECT name, qty, descr, color
FROM s, sp, p
WHERE s.sno = sp.sno
AND sp.pno = p.pno
```

It joins *s* to *sp* and *sp*to *p*, producing :

name	qty	descr	color
Pierre	NULL	Widget	Blue
John	200	Widget	Blue
Mario	1000	Widget	Blue
Mario	200	Widget	Red

Note that the order of tables listed in the FROM clause should have no significance, nor does the order of join predicates in the WHERE clause.

10.6.1 Inner Join

An inner join is a join in which the values in the columns being joined are compared using a comparison operator.

Inner joins can be specified in either the FROM or WHERE clause.. Inner joins specified in the WHERE clause are known as old-style inner joins..

SQL INNER JOIN Syntax

```
SELECT column_name(s)
FROM table_name1
INNER JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```

The INNER JOIN keyword return rows when there is at least one match in both tables Let's assume that we have the following two tables,

Table Store_Information

store_name	**Sales**	**Date**
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
Los Angeles	$300	Jan-08-1999
Boston	$700	Jan-08-1999

Table Geography

region_name	**store_name**
East	Boston
East	New York
West	Los Angeles
West	San Diego

We want to find out sales by store, and we only want to see stores with sales listed in the report. To do this, we can use the following SQL statement using INNER JOIN:

```
SELECT A1.store_name STORE, SUM(A2.Sales) SALES
FROM Geography A1
INNER JOIN Store_Information A2
ON A1.store_name = A2.store_name
GROUP BY A1.store_name
```

Result:

STORE	**SALES**
Los Angeles	$1800
San Diego	$250
Boston	$700

By using INNER JOIN, the result shows 3 stores, even though we are selecting from theGeography table, which has 4 rows. The row "New York" is not selected because it is not present in the Store_Information table.

10.6.2 Outer Join

An inner join excludes rows from either table that don't have a matching row in the other table. An outer join provides the ability to include unmatched rows in the query results. The outer join combines the unmatched row in one of the tables with an artificial row for the other table. This artificial row has all columns set to null.

The outer join is specified in the FROM clause and has the following general format:

table-1 { LEFT | RIGHT | FULL } OUTER JOIN table-2 ON predicate-1

Predicate -1 : is a join predicate for the outer join. It can only reference columns from the joined tables. The LEFT, RIGHT or FULL specifiers give the type of join:

- LEFT — only unmatched rows from the left side table (table-1) are retained
- RIGHT — only unmatched rows from the right side table (table-2) are retained
- FULL — unmatched rows from both tables (table-1 and table-2) are retained

Outer join example:

SELECT pno, descr, color, sno, qty

FROM p LEFT OUTER JOIN sp ON p.pno = sp.pno

pno	descr	color	sno	qty
P1	Widget	Blue	S1	NULL
P1	Widget	Blue	S2	200
P1	Widget	Blue	S3	1000
P2	Widget	Red	S3	200
P3	Dongle	Green	NULL	NULL

10.6.3 Self Join

A query can join a table to itself. Self joins have a number of real world uses. For example, a self join can determine which parts have more than one supplier :

SELECT DISTINCT a.pno

FROM sp a, sp b

WHERE a.pno = b.pno

AND a.sno<>b.sno

As illustrated in the above example, self joins use correlation names to distinguish columns in the select list and where predicate. In this case, the references to the same table are renamed - a and b. Self joins are often used in sub queries.

10.7 Set Operations

Set Operators : Set operators combines results of two queries into a single result. Set operations are generally performed on two Lists obtained from distinct tables.

10.7.1 Union

The UNION command is used to select related information from two tables, much like the JOIN command. However, when using the UNION command all selected columns need to be of the same data type.

Note : With UNION, only distinct values are selected.

SQL Statement 1

UNION

SQL Statement 2

Employees_Norway :

E_ID	E_Name
01	Hansen, Ola
02	Svendson, Tove
03	Svendson, Stephen
04	Pettersen, Kari

Employees_USA :

E_ID	E_Name
01	Turner, Sally
02	Kent, Clark
03	Svendson, Stephen
04	Scott, Stephen

Using the UNION Command

Example :

List all different employee names in Norway and USA:

SELECT E_Name FROM Employees_Norway

UNION

SELECT E_Name FROM Employees_USA

Result :

E_Name
Hansen, Ola
Svendson, Tove
Svendson, Stephen
Pettersen, Kari
Turner, Sally
Kent, Clark
Scott, Stephen

Note : This command cannot be used to list all employees in Norway and USA. In the example above we have two employees with equal names, and only one of them is listed. The UNION command only selects distinct values.

10.7.2 Union All

The UNION ALL command is equal to the UNION command, except that UNION ALL selects all values.

```
SQL Statement 1
UNION ALL
SQL Statement 2
```

Using the UNION ALL Command

Example :

List all employees in Norway and USA:

```
SELECT E_Name FROM Employees_Norway
UNION ALL
SELECT E_Name FROM Employees_USA
```

Result :

E_Name
Hansen, Ola
Svendson, Tove
Svendson, Stephen
Pettersen, Kari
Turner, Sally
Kent, Clark
Svendson, Stephen
Scott, Stephen

10.7.3 Intersection

The Intersect operator is used to return the rows returned by both queries. The following command displays the rows that are common in the results of first and second queries.

```
SELECT              member_id
FROM                members
WHERE               category ='F'
INTERSECT
SELECT DISTINCT     member_id
FROM bookissue
ORDER BY            member_id;
```

MEMBER_ID -----------------
2
4
5
7

10.7.4 Minus

The Minus operator is used to return rows from the result of the first query that are not available in the result of the second query.

```
SELECT            member_id
FROM              members
WHERE             category ='F'
MINUS
SELECT DISTINCT   member_id
FROM              bookissue
ORDER BY          member_id;
```

pno
P1

MEMBER_I D --------------- ---
1
6

- Set Operator combine the result of two queries into one
- All set operators have equal precedence.
- When multiple set operators are present in the same query they are evaluated from left to right.
- The datatype of resulting columns should match in both queries.
- The resultant column name would be the column name of first query.

10.8 Summary

Using different aggregate functions, joins and different set operations we can perform more complex queries on DBMS and get desired outputs in different formats.

10.9 Self Assessment Questions

1. What do you understand by aggregate functions in SQL? Explain.
2. Explain the basic difference between where and having clause of select statement.
3. Explain the different set operations of SQL with the help of suitable example.
4. Explain the use of order by clause of select statement.

Unit - 11 : Queries and Subqueries

Structure of the Unit

11.0 Objective

In general, a *query* is a form of questioning, in a line of inquiry. This unit covers the in depth of SQL query that run on Oracle 10.

- Queries
- Joins
- Sub queries with exists. any, some and all operators.

11.1 Introduction

Tables used in queries :

EMP	Contains information about the employees of the sample company
DEPT	Contains information about the departments in the company

Structure of EMP Table:

EMPNO	NUMBER(4)	Employee number
ENAME	VARCHAR(20)	Employee name
JOB	CHAR (10)	Designation
MGR	NUMBER (4)	Respective manager's EMPNO
HIREDATE	DATE	Date of joining
SAL	NUMBER (9,2)	Basic salary
COMM	NUMBER (7,2)	Commission
DEPTNO	NUMBER (2)	Department number

Structure of DEPT Table :

Column names	**Types**	**Description**
DEPTNO	NUMBER(2)	Department number
DNAME	VARCHAR2 (20)	Name of the department
LOC	VARCHAR2 (10)	Location of the department

11. 2 Queries with join

- Joins are used to combine columns from different tables
- The connection between tables is established through the WHERE clause
- Types of joins: Equi Joins, Cartesian Joins, Outer Joins, Self Joins

One of the most important features of SQL is the ability to define relationships between multiple tables and draw information from them in terms of these relationship, all within a single command. With joins, the information from any number of tables can be related.

To join two tables, the retrieval criteria will typically specify the condition that a column in the first table (which is defined as foreign key) is equal to a column in the second table (which is the primary key referenced by the foreign key) A join's where clause may contain additional conditions. In a join, the table names are listed in the FROM clause, separated by commas.

```
SELECT    < select - list >
FROM      < table1 > < table 2 >,...............< tableN >
WHERE     < table1. column1> = < table2. column 2 and
< table2 column 3 =  < tableN. columnN >
....................................
additional - conditions
```

The variables are defined as follows:-

<select-list> is the set of columns and expressions from **<table1>** through **<tableN>**

<table1> through **<tableN>** are the tables from which column values are retrieved.

<column1> through **<columnN>** are the columns in **<table>** through **<tableN>**

Additional conditions are optional query criteria.:-

We introduce another table, **INCR**, which holds the information about the salary increments of the employee. The structure of the INCR table is:

EMPNO	**NUMBER (4)**
AMT	**NUMBER (7,2)**
DATEINCR	**DATE**

11.2.1 Equi Join

When two tables are joined together using equality of values in one or more columns, they make an **Equi Join.** Table prefixes are utilized to prevent ambiguity and the WHERE clause specifies the columns being joined.

Examples:-

List the employee numbers, names, department numbers and the department name:

```
SELECT empno, ename, emp. deptno, dname FROM emp, dept
WHERE emp. deptno = dept. deptno;
```

Here the deptno column exists in both the tables. To avoid ambiguity, the column name should be qualified with the table name (or with an alias).

Both the table names need to be specifed (emp and dept.) The WHERE clause defines the joining condition i.e., joining the deptno of emp table to the deptno of dept table. Here it checks for the equality values in these columns.

```
EMPNO.          ENAME           EMP. DEPTNO      DNAME
..............  ................ .......................  .......................
7369            SMITH           20               RESEARCH
7499            ALLEN           30               SALES
7521            WARD            30               SALES
7566            JONES           20               RESEARCH
7654            MARTIN          30               SALES
7698            BLAKE           30               SALES
7782            CLARK           10               ACCOUNTING
7788            SCOTT           20               RESEARCH
7839            KING            10               ACCOUNTING
7844            TURNER          30               SALES
7876            ADAMS           20               RESEARCH
7900            JAMES           30               SALES
7902            FORD            20               RESEARCH
7934            MILLER          10               ACCOUNTING
7945            ALLEN           20               RESEARCH
7526            MARTIN          20               RESEARCH
7985            SCOTT           30               SALES
```

14 rows selected.

Using Table Aliases:-

It can be very tedious to type table mames repeatedly. Temporary labels (or aliases) can be used in the FROM clause. These temporary names are valid only for the current select statement.Table aliases should also specified in the select clause. Table aliases can be up to 30 character in length but the shorter they are the better.

Note: The advantages of using table aliases in that it effectively speeds up the query.

SELECT e. empno, e.ename, e.deptno, d.dname FROM emp e, dept d

WHERE e. deptno = d. deptno;

Self Learning Exercise :-

1. Why do we need to specify the table prefix when using equi-join ?

11.2.2 Cartesian Join

When no WHERE clause is specified, each row of one table matches every row of the other table. This results in a Cartesian product.

SELECT empno, ename, dname, loc FROM emp, dept;

If the umber of rows are 14 and 4 in emp and dept tables respectively, then the total number of rows produced is 56.

Cartesian product is finding out all the possible combination of columns from different tables.

Example:-

Consider the following tables and the data present:

Tab1	: Holds the principal amount.
Tab2	: Holds year and rate of interest.

Tab1	Tab2	
PRINCIPAL	YEAR	RATE
....	....	....
1000	1	10
2000	2	11
3000	3	11.5
	4	12

Finding the possible combinations of calculation of amount, a Cartesian join of the Tab1 and Tab2 is required. The formula for calulation of Amount is Princopal* (1+(rate/100)^{year}

SELECT PRINCIPAL, YAER, RATE, PRINCIPAL* POWER (1+RATE/100), YEAR) FROM TAB1, TAB2;

Will produce the following output

```
PRINCIPAL          YEAR          RATE   PRINCIPAL* POWER (1+RATE/100), YEAR)
......................  ............  ..........  .............................................................................
1000               1             10                                  1100
2000               1             10                                  2200
3000               1             10                                  3300
1000               2             11                                  1232.1
2000               2             11                                  2464.2
3000               2             11                                  3696.3
1000               3             11.5                                2772.1959
2000               3             11.5                                2772.3918
3000               3             11.5                                4158.5876
1000               4             12                                  1573.5194
2000               4             12                                  3147.0387
3000               4             12                                  4720.5581
```

11.2.3 Outer Join

If there are any values in one table that do not have corresponding values (s) in the other, in an equi join that row will not be selected. Such rows can be forcefully selected by using the outer join symbol (+) The corresponding columns for that row will have NULLs.

Example:-

In the emp table, no record of the employees belonging to the department 40 is present. Therefore, in the example above for equi join, the row of department 40 from the dept table will not be displayed. Dispaly the list of the employees working in each department Display the department information even if no empployee belongs to that department:

SELECT empno, ename, emp. deptno, dnamo, loc FROM emp, dept

WHERE emp. deptno (+) = demp. deptno;

```
EMPNO           ENAME           EMP. DEPTNDNAME
................ ............... ........................ ..................
                -
7369            SMITH           20              RESEARCH
7499            ALLEN           30              SALES
7521            WARD            30              SALES
7566            JONES           20              RESEARCH
7654            MARTIN          30              SALES
7698            BLAKE           30              SALES
7782            CLARE           10              ACCOUNTING
7788            SCOTT           20              RESEARCH
7839            KING            10              ACCOUNTING
7844            TURNER          30              SALES
7876            ADAMS           20              RESEARCH
7900            JAMES           30              SALES
7902            FORD            20              RESEARCH
7934            MILLER          10              ACCOUNTING
7945            ALLEN           20              RESEARCH
7526            MARTIN          20              RESEARCH
7985            SCOTT           30              SALES
                                40              OPERATIONS
14 rows selected.
```

If the symbol (+) is placed on the other side of the equaticn then all the employee details with no corresponding department name and location, will be displayed with NULL values in DNAME and LOC column.

Rules to place (+) operator:-

- The outer join symbol (+) can not be on both the side.
- We can not "outer join" the some table to more than one other table in a single SELECT statement.
- A condition involving an outer join may not use the IN operator or be linked to another condition by the OR operator.

11.2.4 Self Join

To join a table to itself means taht each row of the table is combined with itself and with every other row of the table. The self join can be viewed as a join of two copies of the same table. The table is not actually copied, but SQL performs the command as though it were.

The syntax of the command for joining a table to itself is almost the same as that for joinning two different table. To distinguish the column names from one another, aliases for the actual the table name are used, since both the table have the same name. Table name aliases are defined in the FROM

clause of the query.

To define the alias, one space is left after the table name and the alias.

Example:-

*** EMP TABLE**

EMPNO	ENAME	MGR
7839	KING	
7566	JONES	7839
7876	ADAMS	7788
7934	MILLER	7782
....	....	....

Consider the emp table shown above. Primary key of the emp table is empno. Details of each employee's manager is just another row in the EMP table whose EMPNO is stored in MGR column of some other row. So every employee except manager has a Manager. Therefore MGR is a foreign key taht reference empno. To list out the names of the manager with the employee record one will have to join EMP with ltself.

SELECT WORKER. ename, MANAGER. ename 'Manager'

FROM emp WORKER, emp MANAGER

WHERE WORKER. mgr = MANAGER. empno;

Where WORKER and MANAGER are two aliases for the EMP table and as a virtual

EMP TABLE

EMPNO	ENAME	MGR
7839	KING	
7566	JONES	7839
7876	ADAMS	7788
7934	MILLER	7782
....	....	....

WORKER

EMPNO	ENAME	MGR
7839	KING	
7566	JONES	7839
7876	ADAMS	7788
7934	MILLER	7782
....	....	....

MANAGER

EMPNO	ENAME	MGR
7839	KING	
7788	SCOTT	7566
7782	CLARK	7839
7934	MILLER	7782
....	....	....

The output will be:

```
ENAME               MANAGER
........................      ..................
SCOTT               JONES
FORD                JONES
ALLEN               BLAKE
WARD                BLAKE
JAMES               BLAKE
TURNER              BLAKE
MARTIN              BLAKE
MILLER              CLARK
ADAMS               SCOTT
JONES               KING
CLARK               KING
BLAKE               KING
SMITH               FORD
13 rows selected.
```

Self Learning Exercise:-

2. In the previous query, only 13 rows (Not 14) have been retrieved. Why ?
3. List all employees who joined the company before their manager.

```
SELECT e. ename, e.hiredate, m.ename manager, m.hiredate
FROM emp e, emp m
WHERE e.mgr = m.empno
and e. hiredate < m.hiredate;
```

```
ename       hiredate     manager     hiredate
............ ................ ................ ...............
ALLEN       15-AUG-83    BLAKE       11-JUN-84
WARD        26-MAR-84    BLAKE       11-JUN-84
MARTIN      05-DEC-83    BLAKE       11-JUN-84
TURNER      04-JUN-84    BLAKE       11-JUN-84
MILLER      21-NOV-83    CLARK       14-MAY-84
JONES       31-OCT-83    KING        09-JUN-84
BLAKE       11-JUN-84    KING        09-JUL-84
CLARK       14-MAY-84    KING        09-JUL-84
SMITH       13-JUN-83    FORD        05-DEC-83
```

11.3 Set Operators

- SET Operator are used to combine information of similar type from one or more than one table.
- Datatype of correponding columns must be the same
- The types of SET operator in ORACLE are :

UNION	: Rows of first query plus rows of second query, less duplicate rows
INTERSECT	: Common rows from all the queries
MINUS	: Rows unique to the first query

SET operator combine two or more queries into one result.

Suppose we want following three details from dept table

- List of all the different designations in department 20 and 30
- List the jobs common to department 20 and 30
- List the jobs unique to department 20 :

To get these combination of information the SET operators UNION, INTERSECT and MINUS are used.

11.3.1 Union

The UNION clause merges the outputs of two of more queries into a single set of rows and columns.

The syntax of UNION operator is

```
select <stmt1>
union
select <stmt2>
{order-by-clause}
```

The variables are defined as follows:

select stmt1 and select stmt 2 are valid **SELECT** statement.

order-by-clause is optional and it references the columns by number rather than by name.

The queries are all executed independently, but their output is merged. Only the final query ends with a semicolon.

Examples:-

- Dispaly the different designations in department 20 and 30:

```
SELECT job FROM emp
WHERE deptno = 20
UNION
SELECT job FROM emp
WHERE deptno = 30;
```

The output will be:

```
JOB
......................
CLERK
SALESMAN
MANAGER
ANALYST
```

Points to be kept in mind while using UNION operator.

- The two select statement may not contain an ORDER BY clause; hawever, the final result of the entire UNION operation can be ordered.
- The number of columns retrieved by first select must be equal to number of columns retrieved by second select.
- The date types of columns retrieved by the select statements should be same.
- The optional order by clause differs from the usual ORDER BY clause an a SELECT statement because the columns used for ordering must be reference by a number rather than name. The reason that the columns must be referenced by number is the SQL does not require that the column name retrieved by first select be identical to theolumns names retrieved by second select.

Example:-

```
select empno, ename from emp
where deptno = 10
UNION
select empno, ename from emp
where deptno = 30
order by 1;
EMPNO             ENAME
..............    ..............
7499              ALLEN
7521              WARD
7654              MARTIN
7698              BLAKE
7839              KING
7844              TURNER
7900              JAMES
7934              MILLER
```

11.3.2 Intersect

The intersect operator returns the rows that are common between two sets of rows.

The syntax of INTERSECT operator is same as UNION operator. Only UNION key word is replaced by INTERSECT.

```
select stmt1
INTERSECT
select stmt2
{order-by clause}
```

Example:-

List the jobs common to department 20 and 30:

SELECT job FROM emp WHERE deptno = 20

```
INTERSECT
SELECT job FROM emp WHERE deptno = 30,
Job
...........
CLERK
MANAGER
```

11.3.3 Minus

Minus operator returns the rows unique to first query. The syntax using the MINUS operator resembles the syntax for the union operator:

```
select stmt1
MINUS
select stmt2
{order-by clause}
```

The requirements and considerations for using the MINUS operator are essentially the same as those for the INTERSECT and UNION operator. To illustrate the use of the MINUS operator, consider the following example.

List the jobs unique to department 20:

```
SELECT job FROM emp
WHERE deptno = 20
MINUS
SELECT job FROM emp
WHERE deptno = 10
MINUS
SELECT job FROM emp
WHERE deptno = 30;
    Job
    ...........
    ANALYST
```

Classroom Exercise:-

Can we rewrite the query to find jobs that are unique to department 20 as:

```
SELECT job FROM emp WHERE deptno = 20
MINUS
SELECT job FROM emp WHERE deptno IN (10, 30);
```

11.4 Sub Queries

- The result of inner query is dynamically substituted in the condition of outer query
- There is no practical limitation to the level of nesting of queries in Oracle 9
- When using relational operators, ensure that the sub query returns a single column output
- In some cases, the DISTINCT clause cab be used to ensure single valued output

SQL has an ability to nest queries within one another. A subquery is a SELECT statement that is nested within another SELECT statement and which returns intermediate results. SQL frist evaluates the inner query (or sub query) within the WHERE clause. The inner query generates values that are tested in the predict of the outer query, determining when it will be true. The return value of inner query is then substituted in the condition of the outer query.

Advantages of Nested queries

- Subqueries allows a developer to build powerful commands out of simple ones.
- The nested subquery is very useful when you need to select rows from a table with a condition that depends on the data in the table itself.

Example:-

List the employees belonging to the department of MILLER:

Here we do not know the department to which MILLER belongs. So, we have to determine the epartment of MILLER and use that department number to find out the other employees of that department.

```
SELECT deptno FROM emp
WHERE ename = 'MILLER';
DEPTNO
....................
10
SELECT ename FROM emp
WHERE deptno = 10;
        ENAME
        ...........
        KING
        CLARK
        MILLER
```

Combining the above two queries:

```
SELECT ename FROM emp
WHERE deptno = (SELECT deptno FROM emp WHERE ename = 'MILLER');
```

* list the names of the employee drawing the highest salary:

```
SELECT ename FROM emp
WHERE sal = (SELECT MAX (sal) FROM emp),;
```

Using Aggregate Functions In Subqueries

Aggregate function produces single value for any number of rows. We want to see all employee details whose salary is greater than avarege salary of employees whose hiredata is before'01-04-81' For this we need to use aggregate function in inner query.

```
SELECT * from emp
where sal >
        (select avg (sal) from emp
        where hiredata < '01-APR- 81');
```

Subqueries in Having

We can also use subqueries within the Having clause. These subqueries can use their own aggregate functions as long as do not produce multiple values or use GROUP BY or HAVING themselves.

List the employee number, name, total number of increments and total increments amount for the employee who has got maximum number of increments:

```
SELECT incr. empno, ename, COUNT (*), SUM (amt) FROM emp, incr
WHERE incr. empno = emp. empno
GROUP BY incr. empno, ename
HAVING COUNT (*) =          (SELECT MAX (COUNT (*) from incr
                            GROUP BY empno);
```

The output will be:

```
EMPNO        ENAME        COUNT(*)      SUM (AMT)
...............  ...............  ....................  ......................
7369         SMITH        3             500
```

List the job with highest average salary.

```
SELECT job, AVG (sal)
FROM emp
GROUP BY job
HAVING AVG (sal)    =       (SELECT MAX (AVG (sal)
                            FROM emp
                            GROUP BY job);
```

The output will be:

```
    JOB                AVG (SAL)
    ..........         ....................
    PRESIDENT          5000
```

The inner query first finds the average salary for each different job group, and the MAX function picks the highest average salary. That value (5000) is used in the HAVING clause.The GROUP BY clause in the main query is needed because the main query's SELECT list contains both an aggregate and non-aggregate column.

Distinct Clause with Subqueries

Distinct clause is used in some cases to force a subquery to generate a single value. Suppose we want to find the details of the department whose manager's empcode '7698'.The query for this is shown below.

```
    select * from dept
    where deptno = (select distinct deptno from emp where mgr = '7698');
```

The inner query will give deptno whose manager's empcode is '7698' Without distinct clause the inner query would have returned more than one row as there are more than one employee whose manager's empcode is '7698'.

Subqueries that return more than one row

When a query returns more than one row we need to use multirow comparision operator.

Example:-

List the names of the employees, who have got an increment:

```
SELECT ename FROM emp
WHERE empno IN (SELECT empno FROM incr);
```

Here, the inner query returns multiple values, hence the IN operator is used instead of a relational operator.

List the names of the employees, who earn lowest salary in each department:

```
SELECT ename, sal, deptno FROM emp
WHERE sal IN (SELECT MIN (sal) FROM emp GROUP BY deptno);
```

Here the inner query has a GROUP BY clause. This means it may return more than one value. In this case, the IN operator must be used because it expects a list of values.

The following points should be kept in mind while writting subqueries:

1. The inner query must be enclosed in parentheses.
2. The inner query must on the right hand side of the condition.
3. The subquery may not have an order by clause.
4. The ORDER BY clause appears at the end of the main select statement.
5. Subqueries are always executed from the most deeply nested to the least deeply nested, unless they are correlated subqueries.

Correlated Subquery

A correlated subquery is a nested subquery which is executed once for each 'candidate row' considered by the main query and which executed uses a value from a column in the outer query. In a correlated subquery, the column value used in inner sub query refers to the column value present in the outher query forming a subquery. The subquery is executed repeatedly once for each row of the main (outer) query table.

List the employee numbers and names, who have got more than 1 increments:

```
SELECT empno ename FROM emp
WHERE 1 <
        (SELECT COUNT (*) FROM incr
        WHERE empno = emp. empno);
EMPNO       ENAME
.............  ...............
7369        SMITH
7788        SCOTT
7900        JAMES
7934        MILLER
```

List employee details who earn salary greater than the average salary for their department.

```
SELECT empno, ename, sal, deptno
FROM emp e
WHERE  sal > (select AVG (sal) FROM emp WHERE deptno = e. deptno);
```

```
EMPNO        ENAME        SAL          DEPTNO
.............  ................  ..........  ..................
7839         KING         5000         10
7566         JONES        2975         20
7788         SCOTT        3000         20
7902         FORD         1600         30
7698         BLAKE        2850         30
5 rows selected.
```

Remember, a correlated subquery is signalled by a column name, a table name or table alias in the WHERE clause that refers to the value of a column in each candidate row ofthe outer select. Also the correlated subquery executes repeatedly for each candidate row in the main query. Correlated subquery is used to answer multipart questions whose answer depends on the value of each row of the parent query. The inner select is normaly executed once for each candidate row.

Self Learning Exercise:-

4. How are nested queries different from joined queries ?

Using Special Operators in Subqureies:-

Some Special operators used in subqueries are:

EXISTS

ANY

SOME

ALL Operators

EXISTS

This operator is used to check the existence of values

This operator produces a Boolean result

It takes a subquery as an argument and evaluates it to True, if it produces any output or

False, if it does not

ANY, SOME and ALL

Used along with the relational operators

Similar to IN operator, but only used in subqueries

The SOME and ANY operator can be used interchangeably

EXISTS operator

The EXISTS operator is frequenly used with correlated subqueries. It tests whether a value is there (NOT EXISTS ensure for nonexistence of values.) If the value exists it returns TRUE, if it does not exists it returns FALSE.

NOT EXISTS operator is more relible if the subquery returns any NULL values.

Examples:-

List all employee who have atleast one person reporting to them.

```
SELECT empno, ename, job, deptno
FROM emp e
```

```
          WHERE EXISTS        (SELECT empno from emp
                              WHERE emp. mgr = e. empno)
          ORDER BY empno;
empno         ename         job                   deptno
..........    ...........   ........              ............
7566          JONES         MANAGER               20
7698          BLAKE         MANAGER               30
7782          CLARK         MANAGER               10
7788          SCOTT         ANALYST               20
7839          KING          PRESISENT             10
7902          FORD          ANALYST               20
```

list the employee details if and only if more than 10 employees are present in department number 10:

```
SELECT * FROM  emp
WHERE DEPTNO = 10 AND EXISTS (SELECT COUNT (*) FROM emp
                              WHERE deptno = 10
                              GROUP BY deptno
                              HAVING COUNT (*) > 10);
```

list the name of employee from the employee table where the increment amount is greater than 1000 and the number of employees receiving the same increment is greater than 5:

```
SELECT  ename FROM emo
WHERE empno IN    (SELECT empno FROM incr
                  WHERE amt > 1000
                  AND EXISTS (SELECT COUNT (*)
                              FROM incr GROUP BY amt
                              HAVING count (*) > 5));
```

List all the employees datails who do not manage any one.

```
SELECT  ename, job from emp e
where not exists (select mgr frm emp where mgr = e. empno);
```

The output is

```
          ename         job
          ..........    ........
          SMITH         CLERK
          ADAMS         CLERK
          ALLEN         SALESMAN
          WARD          SALESMAN
          MARTIN        SELESMAN
          TURNER        SELESMAN
          JAMES         CLERK
          MILLER        CLERK
```

Self Learning Exercise

5. What will be the output if we use NOT IN operator instead of NOT EXISTS in the above query?

ANY operator

The ANY operator compares the lowest value from the set.

List the employee names whose salary is greater than the lowest of an employeebelonging to department number 20:

SELECT ename FROM emp

WHERE sal > ANY (SELECT sal FROM emp WHERE deptno = 21);

List the employee details of those employees whose salary is greater than any of the managers:

SELECT EMPNO, ENAME SAL FROM EMP WHERE SAL > ANY (SELECT SAL FROM EMP WHERE JOB = 'MANAGER');

ALL Operator

In case of All operator the predicate is true if every value selected by the subquery satisfies the condition in the predicate of the outer query.

Example:-

List the employee names whose salary is greater than the highest salary of all employee belonging to department number 20:

SELECT ename FROM emp

WHERE sal > ALL (SELECT sal FROM emp WHERE deptno = 20);

The inner query return salary of all employees who belong to department number 20. The outer query selects employee name of that employee whose salary is greater than all the employees' salary who belong to department number 20.

List the details of the employee earning more than the highest paid MANAGER:

SELECT empno, ename, sal FROM emp WHERE sal > ALL (SELECT sal FROM emp WHERE job = 'MANAGER');

11.5 Summary

- Joins are used to combine columns from different tables.
- The different types of joins are: equi joins, cartesian joins, outer joins, self joins and nonequi joins.
- Joing condition is specified in the WHERE clause of the SELECT statement.
- When two tables are joined together using equality of values in one or more columns, they make an **Equi Join.**

1. Without any joining condition the join becomes a cartesian join.
2. Join a table with itself is know is self join.
3. Self Join is possible by providing table name aliases for the table.
4. With joins, the names of all the table to be joined, need to be specified.
5. To select a row forcfully which cannot be selected using equi join outer join symbol (+). is used.
6. Set operator are used to combine result form different queries. the operator used are UNION, INTERSECT and MINUS.
7. The UNION clause merges the outputs of two or more queries into a single set of rows and columms.

8. The intersect operater returens tha rows thet are commen between two sets of rows.
9. Minus operator returns the rows unique to first query
10. Nested queries are used in a situation where the condition of the query is dependent on the outcome of an inner query. Subqueries can also used within tha HAVING clause.
11. A co-related subquery is a nested subquery which is executed once for each 'candidate row considered by the main query and which on execution uses a value from a column in the outer query
12. The special operators used with subqueries are :EXIST, SOME ANY and ALL.
13. The EXISTS operator is used to be test for the presence of the value. lf the value exists it returns TRUE; if it does not exist returns FALSE.
14. The ANY operator compares tha lowest value from the set.
15. ALL operator returns TRUE if every value selected by the subquery satisfise the condition in the predicate of the outer query

11.6 Self Assessment Questions

1. Fill in the blanks

1. In a join, the connection between tables is established using the--------------clause.
2. Joining a table to itself is called---------------------join.
3. To find the rows common to two different queries, the----------------operator is used.
4. The-----------------and------------------subquery operators are identical.
5. If inner query in subquery returns more than one row then------------------operator is used.

2. State true or false

1. The outer join symbol can appear on both sides of a relation operator.
2. All the table name need to be specified in the FROM claues of the SELECT stetement used for joins.
3. The UNION operator displays duplicate rows also.
4. When the presence of rows needs to be tested in a subquery, the EXIST operator is used.
5. ORDER BY clause can be used in subquery.

3. Hands on Exercise

1. List the item for which no trensaction was mad.
2. Display tha item number, name, rate, quentity recived and value of each item received.
3. List the differnt unit of measurement for item, in classes 'A' and 'B'.
4. List the commen units of measurement aveilable in classes 'c' and 'b'.
5. List the uniqe units of measuerment of item available in class 'A'
6. List all employees, their job and department number, who are having same job as that of any employee of department number 20.
7. List all employees, their salary and their increment, using 'emp' and 'incr' table.
8. Using self join, list all employees having salary greater than or equal to employee number 7788.

9. Consider emp table. List all employee who earn less tahn the average salary of all the employees.
10. List all employees name along with their manager's name. Also list the name of that who has no manager. (Employee KING has no Manager)
11. Dispaly the department that has no employee.
12. List the employee details who earn minimum salary for their job.
13. List the ename, salary, deptno for those employees who earn salary greater than average salary for their department. Sort the output in department number order.
14. List the employee details who earn highest salary for their job.
15. List the details of those employees who are among the five highest earners of the company.

Unit - 12 : Transaction Processing

Structure of the Unit

12.0 Objective

A transaction is a program unit whose execution may change the contents of the database. Transaction processing covers

- Transaction
- Concurrent execution
- Precedence Graph
- Serializability
- Recovery: Logs and Checkpoints

12.1 Introduction

A single DBMS operation as viewed by an user, for example, to update the grade of a student in the relation ENROL (Student_Name, Course, Grade), involves more than one task. Since the data resides on a secondary nonvolatile storage medium, it will have to be brought into the Volatile primary memory for manipulation. This requires that the data be transferred between secondary storage and primary storage. The transfer is usually performed in blocks of the implementation-specified size. The transfer task consists of locating the blocks of in the secondary storage device containing the required tuple (which may be preceded by searching an index), obtaining the necessary locks on the block or the tuple involved in the update, and reading in this block. This task is followed by making the update to the tuple in memory, which in turn is followed by another transfer task, written the tuple back to secondary device, and releasing the locks.

In order to reduce the number of accesses to disk, the blocks are read into blocks of main memory called buffers. We can thus assume that a program performs input/ output using, for example, the get and put operations, and the system transfers the required block from secondary memory to main memory using the read and write operations. The block read (Write) tasks need not be performed in case the system uses buffered input (output) and the required data (space) is already in the primary memory buffer. In such a case the get (put) operation of the program can input (output) the required data from (to) the appropriate buffer. If the required data is not in the buffer, the buffer manager does a read operation and obtains the required data, after which the data is input from the buffer to thte program executing the get statement. If there is no more space left in the buffer, the put operation causes the buffer to be writtern to the secondary storage (with a write) and then the put operation transfers the data from main memory to the space made available in the buffer.

The above DBMS operation of changing the grade of a student in a given course initiated by a user and appearing to her or him as a single operation actually requires a number of distinct tasks or steps to be performed by the DBMS. This is illustrated by the skeleton program given on the next page.

In this program the comment indicates the definition of the action update ENROL of the record for a given student in a given course; this action is being referenced later with the keywords commit and rollback. The statements defined for the update operation are assumed to modify a temporaty copy of the selected portion of the database (the main memory copy of the block of nonvolatile storage containing the tuple for the relation ENROL). Here we are using error to indicate whether there are any errors during the execution of the statements defined for the action update ENROL. If there were any errors, we want to undo any changes made to the database by the statements defined for the update action. This involves simply discarding the temporary copy of the affected portion of the database. The database itself is not changed if a temporary copy of the database is being used. If there were no errors, we want the changes made by the update operations to become permanent by being reflected in the actual database.

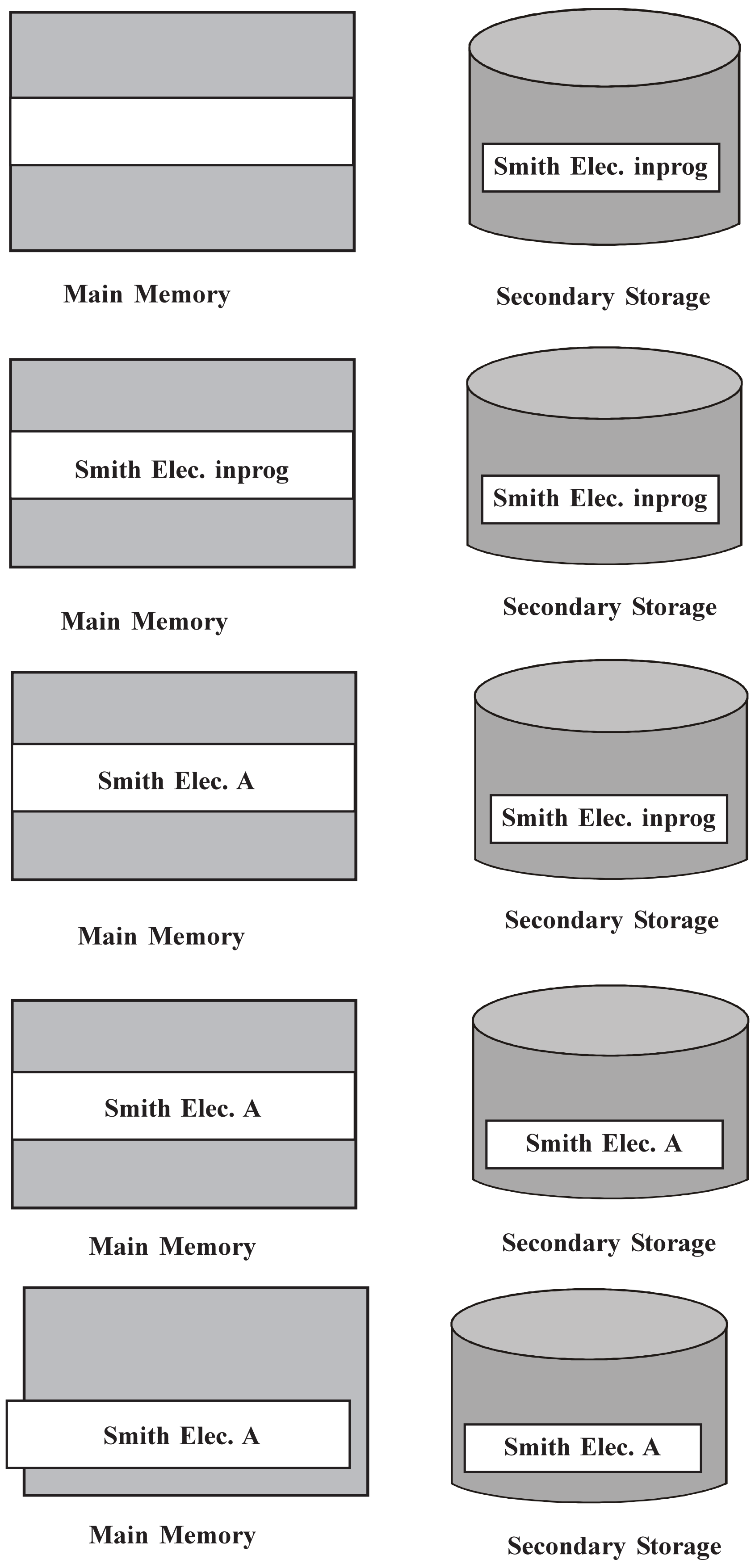
Figure 12.1 : Shows the successive states of the database system at different points of the execution of this program, with the change of student smith's grade in course Elec. from in program to A, as shown in part d of the figure. In case there are any errors by the program, it ignores any modifications and the record for jones rremains unchanged as shown in part e.

The Program unit change the grade given above consists of a number of statements, each of which is executed one at a time (each of the statements is compiled into a number of machine instructions, which are executed one at a time, sequentially). Such sequential execution can be interrupted due to errors. (Interrupts to execute the statements of other concurrent programs can also occur, but we will ignore this type of interruption for the time being). In case of errors, the program may be only partially executed. However, to preserve the consistency of the database we want to ensure that the program is executed as a single unit, the execution of which will not change the consistency of the database. Thus an interruption of a transaction following a system detected error will return the database to its state before the start of the transaction. Such a program unit, which operates on the database to perform a read operation or an update operation (which includes modification, insertion, and deletion), is called a transaction.

Transaction

A transaction is a program unit whose execution may change the contents of a database. If the database was in a consistent state before a transaction, then on the completion of the execution of the program unit corresponding to the transaction, the database will be in a consistent state. This requires that the transaction be consisdered atomic: it is executed successfully or in case of errors, the user can view the transaction as not having been executed at all.

The relationship between an application program and a transaction is shown in **Figure 12.2** the application program can be made up of a number of transactions, T_i, T_2,.................. T_n Each such transaction T_i starts at the time T_{istart}. It commits (or rolls back) at time $T_{icommit}$ ($T_{irollback}$) and terminates at time T_{iend}.



(a) (Initial state of the database before the execution of the Get Enrol statement)

(b) After the execution of the Get Enrol statement

(c) The temporary copy is modified

(d) After the commit statement the database is permanently changed

(e) In case of errors the rollback restores the database to the original state by ignoring the temporary copy

Figure : 12.1 : Database States

The commit and rollback operations included at the end of a transaction ensure that the user can view a transaction as an atomic operation, which preserves database consistency. The commit operation executed at the completion of the modifying phase of the transaction allows the modifications made on the temporary copy of the database items to be reflected in the permanent copy of the database. The rollback operation (which is also called the undo operation) is executed if there was an error of some type during the modification phase of the transaction. It indicates that any modifications made by the transaction are ignored; consequently, none of these modifications is allowed to change the contents of the database. If transaction T_i is rolled back, the logic of the application program is responsible for deciding whether or not to execute transaction T_j (for $i < j \leq n$). Once committed, a transaction cannot be rolled back.

From the definition of a transaction, we see that the status of a transaction and the observation of its actions must not be visible from outside the transaction until the transaction terminates. Any notification of what a transaction is doing must not be communicated, for instance via a message to a terminal, until the transacion commits. Once a transaction terminaters, the user may be notified of its success or failure.

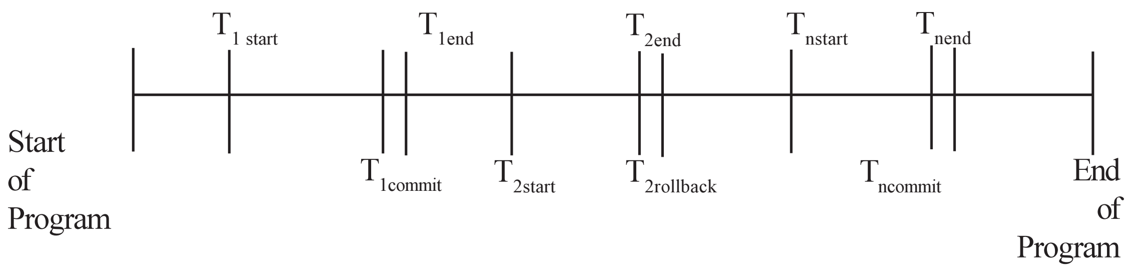


Figure 12.2 : Application Program and transactions

12.2 States of Transaction

A transaction can be considered to be an atomic operation by the user, in reality, however, it goes through a number of states during its lifetime. Figure 12.3 gives these states of the transaction, as well as the cause of a transaction between these states.

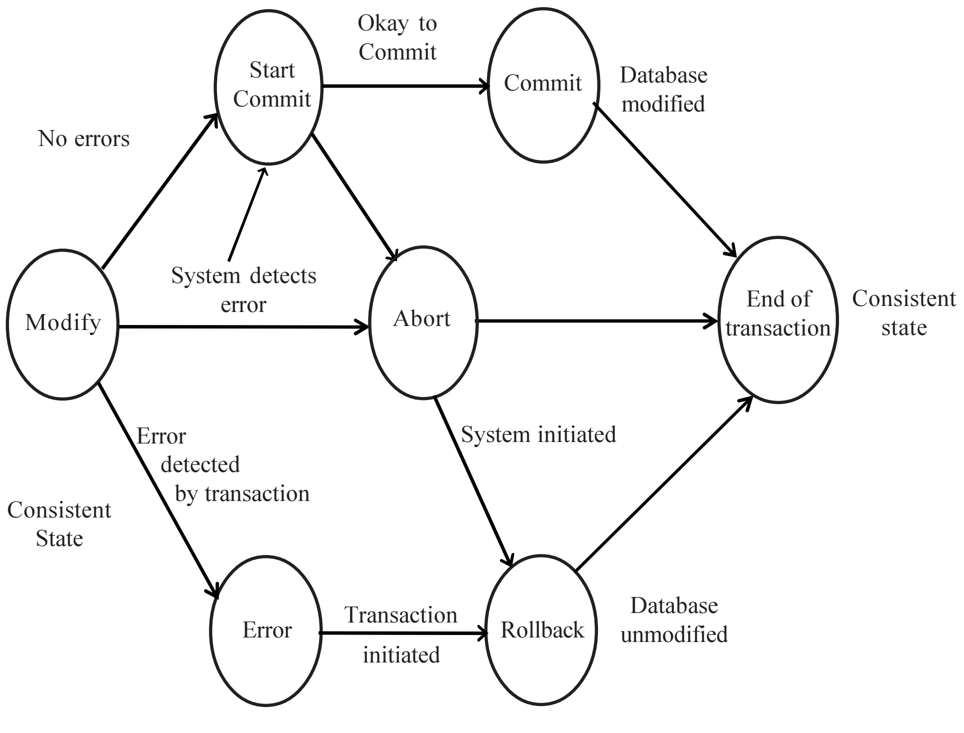


Figure 12.3 : Transaction States

A transaction can end in three possible ways. It can end after a commit operation (a successful termination). It can detect an error during its processing and decide to abort itself by performing a rollback operation (a suicidal termination). The DBMS or the operating system can force it to be aborted for one reason or another (a murderous termination)

We assume that the database is in a consistent state before a transaction starts. A transaction starts when the first statement of the transaction is executted; it becomes active and we assume that it is in the modify state, when it modifies the database. At the end of the modify state, there is a transaction into one of the following states start to commit, abort, or error. If the transaction completes the modification state satisfactorily, it enters the start-to-commit state where it instructs the DBMS to reflect the changes made by it into the database, the transaction is said to be in the commit state and from there the transaction is terminated, the database once again being in a consistent state. In the interval of time between the start-to-commit state and the commit state, some of the data changed by the transaction in the buffers may or may not have been propagated to the database on the nonvolatile storage.

There is a possibility that all the modifications made by the transaction cannot be propagated to the database due to conflicts or hardware failures. In this case the system forces the transaction to the abort state. The abort state could also be entered from the modify state if there are system errors for example, division by zero or an unrecoverable parity error. In case the transaction detects an error while in the modify state, it decides to terminate itself (suicide) and enters the error state and then, the rollback state. If the system aborts a transaction, it may have to initiate a rollback to undo partial changes made by the transaction. An aborted transaction that made no changes to the database is terminated without the need for a rollback, hence there are two paths in Figure 12.3 from the abort state to the end of the transaction. A transaction that, on the execution of its last statement, enters the start to commit state and from there the commit state is guaranteed that the modifications made by it are propagated to the database.

The transaction outcome can be either successful (if the transaction goes through the commit state), suicidal (if the transaction goes through the rollback state), or murdered (if the transaction goes through the abort state), as shown in Figure 12.3. In the last two cases, there is no trace of the transaction left in the database, and only the log indicates that the transaction was ever run.

Any messages given to the user by the transaction must be delayed till the end of the transaction, at which point the user can be notified as to the success or failure of the transaction and in the latter case, the reasons for the failure.

12.3 Properties of Transaction

From the definition of a transaction, we see the status of a transaction and the observation of its actions is not visible from outside until the transaction terminates. Once a transaction ends, the user may be notified of its success or failure and the changes made by the transaction are accessible. In order for a transaction to achieve these characteristics, it should have the properties of atomicity, consistency, isolation and durability (ACID).

The **atomicity** property of a transaction implies that it will run to completion as an indivisible unit, at the end of which either no changes have occurred to the database or the database has been changed in a consistent manner. At the end of a transaction the updates made by the transaction will be accessible to other transactions and processes outside the transaction.

The **consistency** property of a transaction implies that if the database was in a consistent state before the start of a transaction, then on termination of a transaction the database will also be in a consistent state.

The **isolation** property of a transaciton indicates that actions performed by a transaction will be isolated or hidden from outside the transaction untill the transaction terminates. This property gives the transaction a measure of relative independence.

The **durability property** of a transaction ensures that the commit action of a transaction, on its termination, will be reflected in the database. The permanence of the commit action of a transaction requires that any failures after the commit operation will not cause loss of the updates made by the transaction.

12.4 Concurrent Execution

Larger computer systems are typically used by many users in a multiprogramming mode; programs are executed concurrently.(multiple programs execute simultaneously); One reason for the use of multiprogramming is to exploit the different characteristics of the various program to maximize the utilization of the equipment; thus, while one program awaits the completion of an input/output operation, the processor can be used to do the computation of another program. Another reason for choosing multiprogramming is the need to share a resource by these different programms : a database is such a shared resource. The primary objective of the database system (at least on a large mainframe) is to allow many users and application programs to access data from the database in a concurrent manner.

The sharing of the database for read-only access does not cause any problem, but if one of the transactions runnning concurrently tries to modify some data-item, it could lead to inconsistencies. Furthermore, if more than one transaction is allowed to simultaneously modify a data-item in the database, it could lead to incorrect values for the data-item and an inconsistent database. Such would be the result even if each of the transactions were correct and a consistent database would remain so if each of agents access the airline reservations system simultaneously to see if a seat is available on a given flight; if both agents make a reservation against the last available seat on that flight, overbooking of the flight would result. This potential problem of leaving the database in an inconsistent state with concurrently running transactions would be able to access only disjoint data for modificatons.

One method of enforcing mutual exclusion is by some type of locking mechanism that locks a shared resource (for example a data-item) used by a transaction for the duration of its usage by the transaction. The locked data-item can only be used by the transaction that locked it. The other concurrent transactions are locked out and have to wait their turn at using the data-item. However, a locking scheme must be fair. This requires that the lock manager, which is the DBMS subsystem managing the locks, must not cause some concurrent transaction to be permanently blocked from using the shared restore. This is reformed to is avoiding the starvation or livelock situation. The other danger to be avoided is that of deadlock, wherein a number of transactions are waiting in a circular chain, each waiting for the release of resources held by the next transaction in the chain.

In other methods of concurrency control, some form of a priori ordering with a single or many versions of data is used. These methods are called timestamp ordering and multiversion schemes. The optimistic approach, on the other hand, assumes that the data-items used by concurrent transactions are most likely be disjoint.

Concurrency and Possible Problems :

In the last chapter we stressed that a correct transaction, when completed, leaves the database in a consistent state provided that the database was in a consistent state at the start of the transaction. Nevertheless, during the life of a transaction, the database could be inconsistent, although if the inconsistencies are not accessible to other transactions, they would not cause a problem.

In the case of concurrent operations, where a number of transactions are running and using the database, we cannot make any assumptions about the order in which the statements belonging to different transactions will be executed. The order in which these statements are executed is called a schedule. Consider the two transactions in Figure 12.4 Each transaction reads some data-item, performs some operation on the data-item that could change its value, and then writes out the modified data-item.

In figure 12.4 and n subsequent example in this chapter, we assume that the read operation reads in the database value of the named variable to a local variable with an identical name. Any modifications by a transaction are made on this local copy. The modifications made by the transactions are indicated by the operators f1 and f2 in Figure 12.4 These modification are not reflected in the database until the write operation is executed, at which point the modifications in the value of the named variable are said to be commited. in effect the write operation is a signal for committing the modifications and reflecting the changes to the physical database.

Transaction T_1	Transaction T_2
Read (*Avg_faculty_Salary*)	Read (*Avg_Staff_Salary*)
Avg_Faculty_Salary : =	*Avg_Staff_Salary* : =
f_1(Avg_Faculty_Salary)	*f_2(Avg_Staff_Salary)*
Write*(Avg_Faculty_Salary)*	Write*(Avg_Staff_Salary)*

Figure 12.4 : Two concurrent transactions

Schedule - 1	Schedule - 2
Read (Avg_Faculty_Salary)	**Read** (Avg_Staff_Salary)
Avg_Faculty_Salary :=	Avg_Staff_Salary : =
f_1(Avg_Faculty_Salary)	*f_2*(Avg_Staff_Salary)
Write (Avg_Faculty_Salary)	**Read** (Avg_Faculty_Salary)
Read(Avg_Staff_Salary)	Avg_Faculty_Salary :=
Avg_Staff_Salary :=	*f_1* (Avg_Faculty_Salary)
f_2(Avg_Staff_Salary)	**Write**(Avg_Faculty_Salary)
Write (Avg_Staff_Salary)	**Write**(Avg_Staff_Salary)

T
i
m
e

Figure 12.5: Possible interleaving of concurrent transactions of Figure 12.7

Figure 12.5 gives two give two possible schedules for executing the transactions of Figure 12.4 in an interleaved manner. Since the transactions of Figure 12.4 are accessing and modifying distinct data-items, (Avg_Faculty_salary, Avg_staff_Salary), there is no problem in executing these transactions concurrently. In other words, regardless of the order of interleaving of the statements of these transactions, we will get a consistent database on the termination of these transactions.

12.4.1 Lost Update Problem

Consider the transactions of Figure 12.6 These transactions are accessing the same data-item A. Each of the transactions modifies the data-item and writes it back. Again let us consider a number of possible interleavings of the execution of the statements of these transactions. These schedules are given in Figure 12.7.

Starting with 200 as the initial value of A, let us see what the value of A would be if the transactions are run without any interleaving. In other words, the transactions are run to completion, without any interruptions, one at a time in a seial manner. If transaction T3 is run first, then at the end of the transaction the value of A will have changed from 200 to 210. Running transaction T_4 after the completion of T_3 will change the value of A from 210 to 231 Running the transactions in the order T_4 followed by T_3 result in a final value for A of 230. The result obtained with neither of the two interleaved execution schedules of figure 12.7 agrees with either of the results of executing these same transactions serially. Obviously something is wrong !

Transaction T_3	Transaction T_4
Read(*A*)	Read(*A*)
A: = A + 10	*A: = A*1.1*
Write(*A*)	Write(*A*)

	Schedule 1	Transaction T_3	Transaction T_4	Value of A
T	Read(*A*)		Read(*A*)	200
i	*A: = A*1.1*		*A: = A*1.1*	
	Read(*A*)	Read(*A*)		
m	*A: = A + 10*	*A : = A + 10*		
e	Write(*A*)	Write(*A*)		210
↓	Write(*A*)		Write(*A*)	220

(a)

	Schedule 2	Transaction T_3	Transaction T_4	Value of A
T	Read(*A*)	Read(*A*)		200
i	*A : = A + 10*	*A: = A + 10*		
	Read (*A*)		Read(*A*)	
m	*A : = A * 1.1*		*A : = A * 1.1*	
e	Write(*A*)		Write(*A*)	220
↓	Write(*A*)	Write(*A*)		210

(b)

Figure 12.6 : Two Transactions Modifying the Same Data-Item

In each of the schedules given in Figure 12.7 we have lost the update made by one of the transactions. In schedule 1, the update made by transaction T_3 is lost; in schedule 2, the update made by transaction T_4 is lost. Each schedule exhibits an example of the so-called lost update problem of the concurrent execution of a number of transactions.

It is obvious that the reason for the lost update problem is that even though we have been able to enforce that the changes made by one concurrent transaction are not accessible by the other transactions until it commits, we have not enforced the atocity requirement. This demands that only one transaction can modify a given data-item at a given time and other transactions should be locked out from even viewing the unmodified value (in the database) until the modifications (being made to a local copy of the data) are committed to the database.

12.4.2 Inconsistent Read Problem

The lost update problem was caused by concurrent modifications of the same data-item. However, concurrency can also cause problems when only one transaction modifies a given set of data while that set of data is being used by other transaction.

Transaction T_5	**Transaction T_6**
Read(*A*)	*Sum : = 0*
A : = A-100	Read(*A*)
Write(*A*)	*Sum : = Sum + A*

Read(*B*)	Read(*B*)
B : = B + 100	*Sum : = Sum + B*
Write(*B*)	Write(*Sum*)

Figure 12.8 : Two transactions; one modified while the other reads

Consider the transactions of Figure 12.8 Suppose A and B represent some data-items containing integer valued data, for example, two accounts in a bank (or a quantity of some part X in two different locations, etc.). Let us assume that transaction T_5 transfers 100 units from A to B. Transaction T_6 is concurrently running and it wants to dind the total of the current values of data items A and B (the sum of the balance in case A and B represent two accounts, or the total quantity of part X in the two different locations, etc.)

Figure 12.9 gives a possible schedule for the concurrent execution of the transactions of Figure 12.8 with the intial value of A and B being 500 and 1000, respectively. We notice from the schedule that transaction T6 uses the value of A before the transfer was made, but it uses the modified value of B after the transfer. The result is that transaction T_6 erroneously determines the total of A and B as being 1600 instead of 1500. We can also come up with another schedule of the concurrent execution of these transactions that will give the total of A and B as 1400, and of course other schedules that will give the correct answer.

	Schedule	**Transaction T5**	**Transaction T6**	**Value of Database items**		
				A	**B**	**Sum**
	Read (A)	**Read** (A)		500	1000	___
	Sum : = 0		Sum : = 0			0
T	**Read** (A)		**Read** (A)			
i	A : = A - 100	A : = A - 100				
m	**Writer** (A)	**Writer** (A)		400		
e	Sum :=Sum+A		Sum : = Sum+A			500
	Read(B)	**Read** (B)				
	B : + B + 100	B : + B + 100				
	Write (B)	**Write** (B)				
	Read (B)		**Read** (B)		1100	
	Sum : = Sum + B		Sum : = Sum + B			
	Write (Sum)		**Write** (Sum)			1600

Figure 12.9 : Example of Inconsistent Reads

The reason we got an incorrect answer in the schedule of Figur 12.9 was because that transaction T_6 was using values of data-items A and B while they were being modified by transaction T_5. Locking out transaction T_6 from these data-items indicidually would not have solved the problem of the inconsistent read. The problem would have been resolved in this example only if transaction T_5 had not released the exclusive usage of the data item A after locking data-item B. We discuss this scheme, **called two phase locking**.

12.4.3 Semantics of Concurrent Transactions

In concurrent operations, where a number of transactions are running and modifying parts of the database, we not only have to hide the changes made by a transaction from other transactions, but we also have to make sure that only one transaction has exclusive access to these data-items for at least the duration of the original transaction's usage of the data-items. This requires that an appropriate locking mechanism be used to allow exclusive access of these data-items to the transaction requiring them. In the case of the transactions of Figure 12.6 no such locking was used with the consequence that the result is not the same as the result we would have obtained had these transactions run consequently.

Now let us see why the results obtained when we run two transactions, one after the other, need not be the same for different orderings. The modification operations performed by two transactions are not necessarily commutative.

The operations A : = (A + 10) + 20 give the same result as A : = (A + 20) + 10 for the same initial value for A (which is assumed to be an integar valued data-item); this is so because the addition operation is commutative. Similarly, (A * 10) * 20 = (A * 20) * 10.

However, commuting the order of operations, as illustrated by the following expressions, does not always give the same result :

Salary : = (Salary + 1000) * 1.1

Salary : = (Salary * 1.1) + 1000

In the above example we have two transformations. In the first the salary is initially modified by adding 1000 to it and then the result is augmented by 10% to give the revised Salary. In the second the Salary is first augmented by 10% and then 1000 is added to the result, which becomes the revised Salary. The reasonable approach, to make sure that the intended result is obtained in all cases (i.e. to make sure that transaction T_i is completed before transaction T_j is run), would be to code the operations in a single transaction and not to divide the operations into two or more transactions. Thus, if the above set of operations on Salary were written as two transactions as given below, we cannot be sure which of the above two results would be obtained with their concurrent execution.

Transaction T_i	**Transaction T_j**
Read Salary	**Read** Salary
*Salary : = Salary * 1.1*	*Salary : = Salary + 1000*
Write Salary	**Write** Salary

In efffect, the division of a transaction into interdependent transactions run serially in the wrong order would give erroneous results. Furthermore, these interdependent transactions must not be run concurrently, otherwise the concurrent execution will lead to results that could be incorrect again and not agree with the result obtained by any serial execution of the same transactions. It is logical error to divide a single set of operations into two or more transactions. We assume hereafter that transactions are semantically correct.

12.5 Serializability

In the above examples consider the transactions are independent. An execution schedule of these transactions as shown in figure 12.10 is called a serial execution. In a serial execution, each transaction runs to completion before any statements from any other transaction are executed. In Schedule A given in Figure 12.10 a, transaction T_3 is run to completion before transaction T_4 is executed. In schedule B, transaction T_4 is run to completion before transaction T_3 is started. If the initial value of A in the database were 200, Schedule A would result in the value of A being changed to 231. Similarly, Schedule B with the same mitial value of A would give a result of 230.

This may seem odd, but in a shared environment, the result obtained by independent transactions that modify the same data-item always depends on the order in which these transactions are run; and any of these results is considered to be correct.

	Schedule A	Transaction T_3	Transaction T_4
	Read (A)	**Read** (A)	
T	A : = A + 10	A : = A + 10	
i	**Write** (A)	A : = A + 10	
m	**Read** (A)		**Read** (A)
e	A : = A * 1.1		A : = A * 1.1
↓	**Write** (A)		**Write** (A)

(a)

	Schedule B	Transaction T_3	Transaction T_4
	Read (A)		**Read** (A)
T	A : = A * 1.1		A : = A * 1.1
i	**Write** (A)		**Write** (A)
m	**Read** (A)	**Read** (A)	
e	A : = A + 10	A : = A + 10	
↓	**Write** (A)	A : = A + 10	

(b)

Figure 12.10 : Two serial Schedules

If there are two transactions and if they refer to and use distinct data-items, the result obtained by the interleaved execution of the statements of these transactions would be the same regardless of the order in which these statements are executed (Provided there are no other concurrent transactions that refer to any of these data-items). In this chapter, we assume that the concurrent transactions share some data-items, hence we are interested in a correct ordering of execution of the statements of these transactions.

A nonserial schedule wherein the operations from a set of concurrent transactions are interleaved is considered to be serializable if the execution of the operations in the schedule leaves the database in the same state as some serial execution of these transactions. With two transactions, we can have at most two district serial schedules, and starting with the same state of the database, each of these serial schedules could give a different final state of the database. Starting with an initial value of 200 for A, the serial schedule illustrated in Figure 12.10 a would give the final value of A as 231, and for the serial schedule illustrated in part b the final value of A would be 230. If we have n concurrent transactions, it is possible to have n!, where n! = n * (n-1) * (n - 2) * ...... * 3 * 2 * 1 distint serial schedules, and possibly that many district resulting modifications to the database. For a serializable schedule, all we require is that the schedule gives a result that is the same as any one of these possibly distinct results.

When *n* transactions are run concurrently and in an interleaved manner, the number of possible schedules is much larger than n!. We would like to find out if a given interleaved schedule produces the same results as one of the serial schedules. If the answer is positive, then the given interleaves schedule is said to be serializable.

Definition - Serializable Schedule :

Given an interleaved execution of a set of n transactions; the following conditions hold for each transaction in the set :

- All transactions are correct in the sense that if any one of the transactions is executed by itselt on a consistent database, the resulting database will be consistent.
- Any serial execution of the transactions is also correct and preserves the consistency of the database; the results obtained are correct. (This implies that the transactions are logically correct and that no two transactions are interdependent.)

The given interleaved execution of these transactions is said to be serializable if it produces the same result as some serial execution of the transactions.

Since a serializable schedule gives the same result as some serial schedule and since that serial schedule is correct, then the serializable schedule is also correct. Thus, given any schedule, we can say it is correct if we can show that it is serializable.

Algorithm given in Section 12.5.2 establishes the serializability of an arbitrarily interleaved execution of a set of transactions on a database. The algorithm does not consider the nature of the computations performed by a transaction nor the exact effect of each such computational operation on the database. In effect, the algorithm ignores the semantics of the operations performed by the transactons including the commuting property of algebraic of logical computations of the transactions. We may conclude from the algorightm that a given schedule is not serializable, when in effect it is, if some of the semantics and the algebraic commutability were not ignored. However the algorithm will never lead us to conclude that a schedule is serializable, when it does not produce the same result as some serial schedule. The computation involved in analyzing each transaction and seeing if its operations could be safely interleaved with those of other concurrent transactions is not justified by the greater degree of concurrency of the resulting "better" serializable schedule.

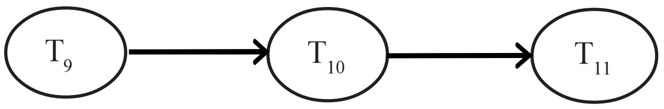
In Algorithm 12.5.2 we make the following assumptions :

- Each transaction is a modifying transaction, i.e., it would change the value of at least one database item.
- For each such item A that a transaction modifies, it would first read the value a of the item from the database (this is the read-before-write protocol)
- Having read the value it would transform a to f(a). where f is some transaction-dependent computation of transformation.
- It would then write this new value to the database.

Before presenting the algorithm we present the notion of precedence graph.

	Schedule	**Transaction T_9**	**Transaction T_{10}**	**Transaction T_{11}**
	Read (A)	**Read** (A)		
	A : = f_1(A)	A : = f_1 (A)		
T	**Write** (A)	**Write** (A)		
i	**Read** (A)		**Read** (A)	
m	A : = f_2(A)		A:=f_2(A)	
e	**Write** (A)		**Write** (A)	
	Read (B)		**Read** (B)	
	A:=f_3(B)		B : =f_3(B)	
	Write (B)		**Write** (B)	
	Read (B)			**Read**(B)
	A:=f_4(B)			B:=f_4(B)
	Write (B)			**Write** (B)

(a)



(b)

Figure 12.11 : (a) A schedulue and (b) an acyclic precedence graph.

12.5.1 Precedence Graph

Precedence graph G(V, E) consists of a set of nodes or vertices V and a set of directed arcs or edges E. Figure 12.5 gives an example of a schedule and the corresponding precedence graph. The schedule is for three transactions T_9 ,T_{10} and T_{11} and the corresponding precedence graph has the vertices T_9, T_{10}, and T_{11} there is an edge from T_9 to T_{10} and another edge from T_{10} to T_{11} if T_9 , T_{10} and T_{11} represent three transactions, the precedence graph represents the serial execution of these transactions.

In a precedence graph, a directed edge from a node T_i to a node T_j. i # j, indicates one of the following conditions regarding the read and write operations in transactions T_i and T_j with respect to some database item A:

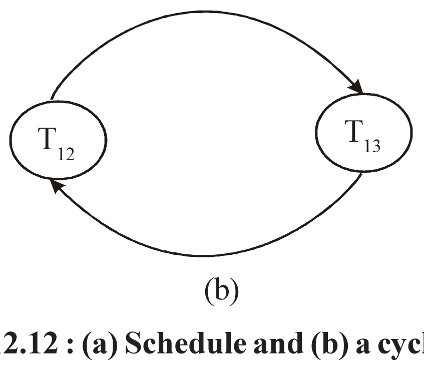
- T_j performs the operation Read(A) to read the value written by T_i performing the operation Write(A)
- T_j performs the operation Write(A) after Ti performs the operation Read(A).

If we limit ourselves to the read-before-write protocol only, we have to look for an edge corresponding to these conditions only.

In figure 12.12 all the statements in transaction T_9 are executed before transaction T_{10} is started. Similarly, all the operations of T_{10} are completed before starting T_{11}. The precedence graph corresponding to the schedule of part a is given in part b.

	Schedule	**Transaction T_{12}**	**Transaction T_{13}**
	Read (A)	**Read** (A)	
	A : =f_1(A)	A : =f_1 (A)	
T	**Read** (A)		**Read** (A)
i	A:=f_2(A)		A:=f_2(A)
m	**Write** (A)		**Write** (A)
e ↓	**Write** (A)	**Write** (A)	

(a)



(b)

Figure 12.12 : (a) Schedule and (b) a cyclic precedence graph.

Figure 12.12 a gives a schedule and Figure 12.12 b gives precedence graph for transactions T_{12} and T_{13} in the precedence graph there is an edge from T_{12} to T_{13} as well as edge from T_{13} to T_{12} The edge T_{13} to T_{12} is included because T_{12} executes a write operation after T_{13} executes a write operations for the same database item A. The edge $T_{12}$to T_{13} is included because T_{13} executes a write optration after T_{12} executes a read operation for the same database item A. We see that the precendence graph has a cycle, since we can start from one of the nodes of the graph and, following the directed edges, return to the starting node.

A precedence graph is said to be acyclic if there are no cycles in the graph. The graph of Figure 12.11b has no cycles. The graph of Figure 12.12b is cyclic, since it has cycle.

The precedence graph for serializable schedule S must be acyclic, hence it can be converted to a serial schedule. To test for the serializability of the arbitraty schedule S for transactions T_1,.......... T_k we convert the schedule into a precedence graph and then test the precedence graph for cycles. If no cycles are detected the schedule is serializable; otherwise it is not. If there are n nodes in the graph for schedule S, the number of operations required to check if there is a cycle in the graph is proportional to n^2.

12.5.2 Serializability Algorithm : Read-before-write Protocol :

In the read-before-write protocol we assume that a transaction will read the data-item before it modifies it and after modifications, the modified value is written back the the database. In the Algorithm, we give the method of testing whether a schedule is serializable. We create a precedence graph and test for a cycle in the graph. If we find a cycle. The schedule is nonserializable; otherwise we find a linear ordering of the transactions.

In Examples 12.1 and 12.2 we illustrate the application of this algorithm.

Example 12.1 : Consider the schedule of Figure A. The precendence graph for this schedule is given in Figure B . The graph has three nodes corresponding to the three transactions T_{14}, T_{15} and T_{16}. There is an arc from T_{14} to T_{15} because T_{14} writes data-item A before T_{15} reads it. Similarly, there is an arc from T_{15} to T_{16} because T_{15} writes data-item B before T_{16} reads it. Finally, there is an arc from T_{16} to T_{14} because T_{16} writes data-item C before T_{14} reads it. The precedence graph of Figure B has a cycle formed by the directed edges from T_{14} to T_{15}, from T_{15} to T_{16} and from T_{16} back to T_{14} Hence, the schedule of Figure A is not serializable. We cannot execute the three transactions serially to get the same result as the given schedule.

Schedule	**Transaction T_{14}**	**Transaction T_{15}**	**Transaction T_{16}**
Read(A)	**Read**(A)		
Read(B)		**Read**(B)	
A: =f_1(A)	*A : =f_1(A)*		
Read(C)			**Read**(C)
B:=f_2(B)		B:=f_2(B)	
Write(B)		**Write**(B)	
C: =f_3(C)			*C : =f_3(c)*
Write(C)			**Write**(C)
Write(A)	**Write**(A)		
Read(B)			**Read**(B)
Read(A)		**Read**(A)	
A : =f_4(A)		*A : =f_4(A)*	
Read(C)	**Read**(C)		
Write(A)		**Write**(A)	
C: =f_5(C)	*C : =f_5(C)*		
Write(C)	**Write**(C)		
B : =f_6(B)			*B : =f_6(B)*
Write(B)			**Write**(B)

Figure : (A) An Execution Schedule Involving Three Transactions

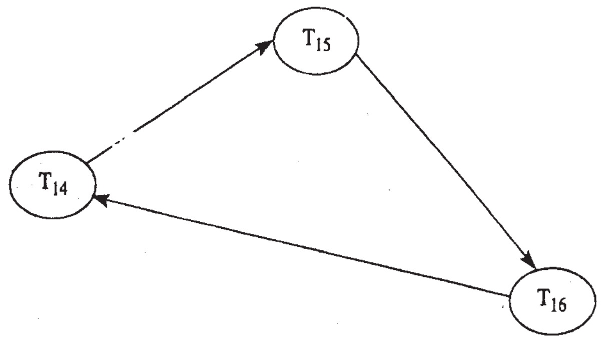


Figure : (B) A precedence Graph with a Cycle.

Example 12.2 presents a serializable schedule.

Example 12.2 Consider the schedule given in Figure C. The execution schedule of the figure is serializable because the precedence graph for this schedule given in Figure D, does not contain any cycles. The serial schedule is T_{17}, followed by T_{18}, followed by T_{19}

	Schedule	Transaction T_{17}	Transaction T_{18}	Transaction T_{19}
	Read (A)	**Read** (A)		
	A : =f_1(A)	A : =f_1 (A)		
	Read (C)	**Read** (C)		
	Write (A)	**Write** (A)		
	A : =f_2(C)	A : =f_2 (C)		
	Read (B)		**Read** (B)	
T	**Write**(C)	**Write** (C)		
i	**Read** (A)		**Read** (A)	
m	**Read** (C)			**Read** (C)
e	B : =f_3(B)		B : =f_3 (B)	
	Write(B)		**Write** (B)	
	C : =f_4(C)			C : =f_4 (C)
	Read (B)			**Read** (B)
	Write (C)			**Write** (C)
	A : =f_5(A)		A : =f_5(A)	
	Write (A)		**Write** (A)	
	B :=f_6(B)			B : =f_6(B)
	Write (B)			**Write** (B)

Figure : (C) An Execution Schedule Involving Three Transactions

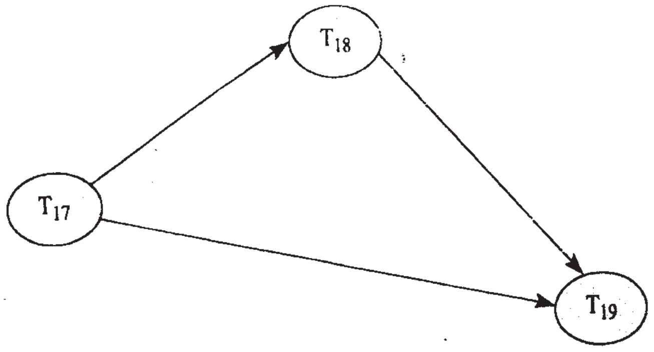


Figure : (D) Precedence graph for schedule of Figure C.

12.5.3 Serializability Algorithm : Read-Only and Write-Only Protocols :

The Algorithm is for a set of transactions that follow the read before write protocol. Some transactions, in addition to having a set of data-items that are read before rewritten have another set of data-items that are only read and a further set of data-items that are only written. In such a case some additional edges must be added to the graph.

12.6 Recovery

In designing a reliable system we try to anticipate different types of failures and provide for the means to recover without loss of information. Some very rare failures may not be catered to for economic reasons. Recovery from failures that are not thought of, overlooked, or ignored may not be possible. In common practice, the recovery system of a DBMS is designed to anticipate and recover from the following types of failure :

Failures without loss of data : This type of failure is due to errors that the transaction discovers before it reaches the start to commit state. It can also be due to the action of the system, which resets its state to that which existed before the start of the transaction. No loss of data is involved in this type of failure, especially in the case where the transactions are run in a batch mode; these transactions can be rerun later in the same sequence.

Failure with loss of volatile storage : Such a failure can occur as a result of software or hardware errors. The processing of an active transaction is terminated in an unpredictable manner before it reaches its commit or rollback state and the contents of the volatile memory are lost.

Failure with loss of nonvolatile storage : This is the sort of failure that can occur after the failure of a nonvolatile storage system; for example, a head crash on a disk drive or errors in writing to a nonvolatile device.

Failure with a loss of stable storage : This type involves loss of data stored on stable storage. The cause of the loss could be due to natural or man-made disasters. Recovery from this type of failure requires manual regeneration of the database. The probability of sush a failure is reduced to a vary small value by having multiple copies of data in stable storage, stored in physically secure environments in geographically dispersed locations.

The basic technique to implement the database transaction paradigm in the presence of failures of various kinds is by using date redundancy in the from of logs, check-points and archival copies of the database.

12.6.1 Logs

The log, which is usually written to stable storage, contains the redundant date required to recover from volatile storage failures and also from errors discovered by the transaction or the database system. For each transaction the each transaction the following data is recorded on the log:

- A start of transaction marker.
- The transaction identifier.
- The record identifiers, which include the identifiers for the record occurrences.
- The operation(s) performed on the records (insert, delete, modify).
- The previous value(s) of the modified data. This information is required for undoing the changes made by a partially completed transaction; it is called the undo log. Where the modification made by the transaction is the insertion of a new record, the previous values can be assumed to be null.
- The updated value(s) of the modified record(s). This information is required for making sure that the changes made by a committed transaction are in fact reflected in the database and can be used to redo these modification. This information is called the redo part of the log. In case the modification made by the transaction is the deletion of a record, the updated values can be assumed to be null.
- A commit transaction marker if the transaction is committed; otherwise an abort or rollback-transaction marker.

The log is written before any updates are made to the database. This is called the **write-ahead log strategy**. In this strategy a transaction is not allowed to modify the physical database until the undo portion of log (i.e. the portion of the log that contains the value (s) of the modified data) is written to stable storage. Furthermore, the log write-ahead strategy requires that a transaction is allowed to commit only after the redo portion of the log and the commit transaction marker are written to the log. In iffect, both the undo and redo portin of the log will be written to stable storage before a transaction commit. Using this strategy, the partial updates made by an uncommitted transaction can be undone using the undo portion of the log, and a failure occurring between the writing of the log and the completor of updating the database corresponding to the action implied by the log can be redone.

Let us see how the log information can be used in the case of a system crash with the loss of volatile information. Consider a number of transaction, as shown in figure 12.13. The figure shows the system start-up at the time $\mathbf{t_0}$ and a number of concurrent transaction T_0, T_1, ... , T_{i+6} are made on the database. Suppose a system crash occures at time t_x

We have stored the log information for transaction T_0 through T_{i+2} on stable storage, and we assume that this will be available when the system comes up after.

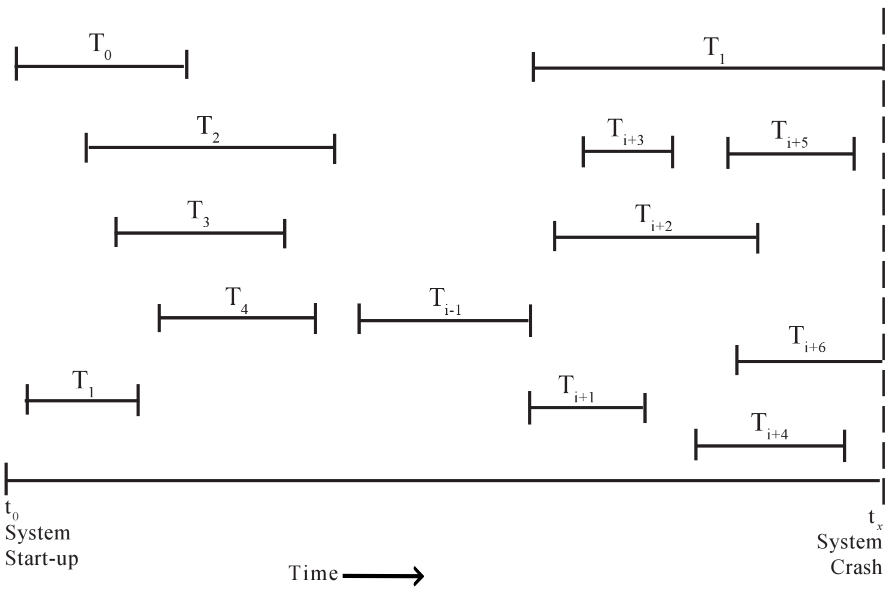


Figure 12.13 : DBMS operation to a system crash

The crash, Furthermore, we assume that the database existing on the nonvolatile storage will also be availble. It is clear that the transaction that were not committed at the time of the system crash will have to be undone. The changes made by these uncommitted transactions will have to be rolled back. The transactions that have not been committed can be found by examining the log and those transactions that have a start of transactions marker but no commit or abort transactions marker are considered to have been active at the time of the crash. These transactions have to be rolled back to restore the database to a consistent state. In figure 12.13 the transactions T_i and T_{i+6} started before the crash, but they had not been committed and, hence, are undone.

However, it is not clear from the log to what extent the changes made by committed transactions have actually been propagated to the database on the nonvolatile storage. The reason for this uncertainly is the fact that buffers (implemented in volatile storage) are used by the system to hold the modified data. Some of the changed data in these buffers may or may not have been propagated to the database on the nonvolatile storage. In the absence of any method of finding out the extent of the loss, we will be forced to redo the effects of all committed transactions. For figures 12.13, this involves redoing the changes made by all transactions from t_0 Under such a scenario, the longer the system operates without a crash, the longer it will take to recover from the crash.

In the above, we have assumed that the log informatin is available up to the time of the system crash in nonvolatile storage. However, the log information is also collected in buffers. In case of a system crash with loss of volatile information, the log information collected in buffers will also be lost and transactions that had been completed for some period prior to the system crash may be missing their respective end-of-transactions markers in the log. Such transactions, if rolled back, will likely be partially undone. The write-ahead log strategy avoids this type of recovery problem, since the log information is forced to be copied to stable storage before the transactions commits.

These problems point to the conclusion that some means must be devised to propogate to stable storage at regular intervals all the log informaton, as well as modificatins to the database existing at a given time. Then the recovery operation after a system crash will not have to reprocess all transactions from the time of start-up of the system.

12.6.2 Checkpoints

In an on- line database system, for example an airline reservation system, there could be hundreds of transactions handled per minute. The log for this type of database contains a very large volume of information. A scheme called checkpoint is used to limt the the volume of log information that has to be handled and processed in the event of a system failure involving the loss of volatile information. The checkpoint scheme is an additional component of the logging scheme described above

In the case of a systeam crash the log information being collected in buffers will be lost. A checkpoint operation, performed periodically,copies log information onto stable storage. The information and operations performed at each checkpoint consist of the following :

- A start-of-checkpoint record giving the identification that it is a check point along with the time and date of the checkpoint is written to the log on a stable storage device.
- All log information from the buffers in the volatile storage is copied to the log on stable storage device
- All database udates from the buffers in the volatile storage are propagated to the physical data base.
- An end-of-checkpoint record is written and the address of the checkpoint record is saved on a file accessible to the recovery routine on start-up-after a system crash.

For all transactions active at checkpoint, their identifiers and their database modification actions, which at that time are reflected only in the database buffers will be propagated to the appropriate storage.

The frequency of checkpointing is a design consideration of the recovery system. A checkpoint can be taken at fixed intervals of time (say, every 15 minutes). If this approach is used, a choice has to be made regarding what to do with the transactions that are active when the checkpoint signal is generated by a system timer. In one alternative, called **transaction-consistent checkpoint**, the transactions that are active when the system timer signals a checkpoint are allowed to be started until the checkpoint are allowed to complete, but no new transactions (requiring modifications to bhe database) are allowed to be started until the checkpoint is completed. This scheme, though attractive, makes the database unavailable at regular intervals and may not be acceptable for certain online applications. In addition, this approach is not appropriate for long transactions. In the second variation, called action consistent checkpoint, active transactions are allowed to complete the current step before the checkpoint and no new actions can be started on the database until the checkpoint is completed; during the chekpoint no actions are permitted on the database. Another alternative, called **transaction-oriented checkpoint**, is to take a checkpoint at the end of each transaction by forcing the log of the transaction onto stable storage. In effect, each commit transaction is a chekpoint.

How does the checkpoint information help in recovery ? To answer this question, reconsider the set of transactions of Figure 12.13, shown in Figure 12.14 with the addition of a checkpoint being taken at time t.

Suppose, as before, the crash occurs at time, $t_{x.}$ Now the fact that a checkpoint was taken at time t_c indicates that at that time all log and data uffers were propagated to storage. Transactions T_0, ..., T_{i-1} as well as transactions T_{i+1} and T_{i+3} were committed, and their modifications are reflected in the database. With the checkpoint scheme these transactions are not required to be redone during the recovery operation following a system crash occurring after time t_c. A transaction such as T_i (which started before checkpoint time t_c), as well as transaction T_{i+6} (which started after checkpoint time t_c,) were not committed at the time of the crash and have to be rolled back. Transactions such as T_{i+4} and T_{i+5} which started after checkpoint time t_c and were committed before the system crash, have to be redone. Similarly, transactions such as T_{i+2}, which started before the checkpoint time and were committed before the system crash, will have to be redone. However, if the commit transaction information is missing for any of the transactions T_{i+2}, T_{i+4}, or T_{i+5}, then they have to be undone.

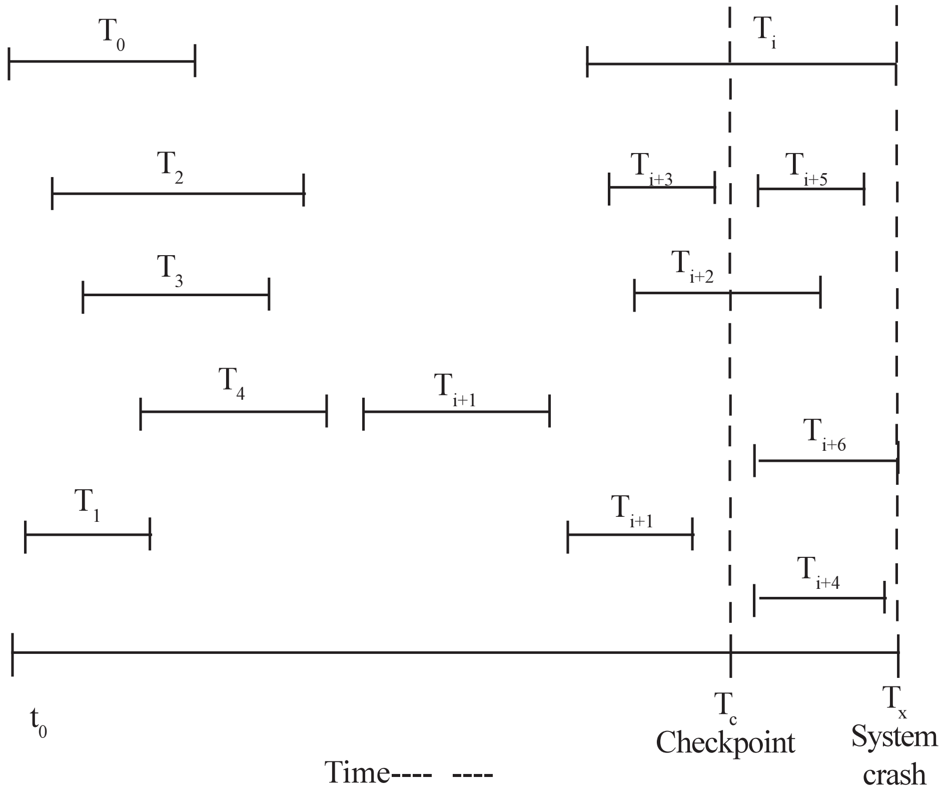


Figure 12.14 : Checkpointing

Let us now see how the system can perform a recovery at time tx. Suppose all transactions that started before the checkpoint time but were not committed at that time, as well as the transactions started after the checkpoint time, are placed in an undo list, which is a list of transactions to be undone. The undo list for the transactions of Figure 12.14 is given below:

UNDO List (T_i, T_{i+2}, T_{i+4}, T_{i+5}, T_{i+6})

Now the recovery system scans the log in a backward direction from the time t_x of system crash. If it finds that a transaction in the undo list has committed, that transaction is removed from the undo list and placed in the redo list. The redo list contains all the transactions that have to be redone. The reduced undo list and the redo list for the transactions of Figure 12.14 are given below :

REDO List; (T_{i+4}, T_{i+5}, T_{i+2})

UNDO List; (T_i, T_{i+6})

Obviously, all transactions that were committed before the checkpoint time need not be considered for the recovery operation. In this way the amount of work required to be done for recovery from a system crash is reduced. Without the checkpoint scheme, the redo list will contain all transactions except T_i and T_{i+6}. A system crash occurring during the checkpoint operation, requires recovery to be done using the most recent previous checkpoint.

The recovery scheme described above takes a pessimistic view about what has been propagated to the database at the time of a system crash with loss of volatile information. Such pessimism is adopted both for transactions committed after a checkpoint and transactions not committed since a checkpoint. It assumes that the transactions committed since the checkpoint have not been able to propagate their modifications to the database and the transactions still in progress have done so.

Note that in some systems the term checkpoint is used to denote the correct state of system files recorded explicitly in a backup file and the term checkpointing is used to denote a mechanism used to restore the system files to a previous consistent state. However, in a system that uses the transaction paradigm, checkpoint is a startegy to minimize the search of the log and the amount of undo and redo required to recover from a system failure with loss of volatile storage.

Archival Database and Implementation of the Storage Hierarchy of a Database System

Figure 12.15 gives the different categories of data used in a database system. These storage types are sometime called the storage hierarchy. It consists of the archival database, physical database, archival log, and current log.

Physical database : This is online copy of the database that is stored in nonvolatile storage and used by all active transactions.

Current database : The current version of the database is made up of the physical database plus modifications implied by buffers in the volatile storage.

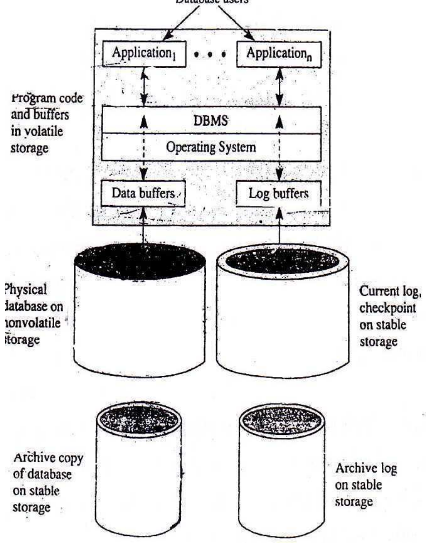

Database users
Application₁
Applicationₙ
Program code and buffers in volatile storage
DBMS
Operating System
Data buffers
Log buffers
Physical database on nonvolatile storage
Current log, checkpoint on stable storage
Archive copy of database on stable storage
Archive log on stable storage

Figure 12.15 : Database storage hierarchy

Archival Database in Stable Storage :

This is the copy of the database at a given time, stored on stable storage. It contains the entire database in a quiescent mode (i, e, no transactions were active when the database was copied to the stable storage) and could have been made by simple dump routines to dump the physical database (which in quiescent state would be the same as the current or online database) onto stable storage. The purpose of the archival database is to recover from failures that involve loss of nonvolatile storage. The archiving process is a relatively time-consuming operation and during this period the database is not accessible. Consequently, archiving is done at infrequent intervals. The frequency of archiving is a loss of nonvolatile data being the arbitrator. All transactions that have been executed on the database from the time of archiving have to be redone in a global recovery operation. No undoing is required in the global recovery operation since the archival database is a copy of the database in a quiescent state, and only the committed transactions since the time of archiving are applied to this database.

Current log : This contains the log information (including the checkpoint) required for recovery from system failures involving loss of volatile information

Archival log : This log is used for failure involving loss of nonvolatile information. The log contains information on all transactions made on the database from the time of the archival copy. This log is written in chronological order. The recovery from loss of nonvolatile storage uses the archival copy of the database and the archival log to reconstruct the physical database to the time of the nonvolatile storage failure.

With the above storage hierarchy of a database, we can use the following terms to denote different combinations of this hierarchy.

The on line or current database is made up of all the records (and the auxiliary structures such as indexes) that are accessible to the DBMS during its operation. The currrent database consists of the data stored in nonvolatile storage (Physical database) as well as the data stored in buffers (in the volatile storage) and not yet propagated to the nonvolatile storage.

The materialized database is that portion of the database that is still intact after a failure. All the data stored in the buffers would have been lost and some portion of the database would be in an inconsistent state. The log information is to be applied to the materialized database by the recovery system to restore the database to as close a state as possible to the online database prior to the crash. Obviously, it will not be possible in all cases to return to exactly the same state as the precrash online database. The intent is to limit the amount of lost data and the loss of completed transactions.

12.6.3 Do, Undo, and Redo

A transaction on the current database transforms it from the current state to a new state. This is the so-called do operation. The undo and redo operations are functions of the recovery subsystem of the database system used in the recovery process. The undo operation undoes or reverses the actions (Possibly partially executed) of a transaction and restores the database to the state that existed before the start of the transaction. The redo operation redoes the action of a transaction and restores the database to the state it would be in at the end of the transaction. The undo operation is also called into play when a transaction decides to terminate itself (Suicidal termination). Figure 12.6(b) shows the transformation of the database as a result of a transaction do, redo, and undo.

The undo and redo operations for a given transaction are required to be idempotent; that is, for any transaction, performing one of these operations once is equivalent to performing it any unmber of times. Thus :

undo (any action) = undo (undo(..undo(any action)....))

Redo(any action) = redo (redo(..redo(any action)..))

The reason for the requirement that undo and redo be idempotent is that the recovery process, while in the process of undoing or redoing the actions of a transaction, may fail without a trace, and this type of failure can occur any number of times before the recovery is completed successfully.

Transaction Undo :

A transaction that discovers an error while it is in progress and consequently needs to abort itself and roll back any changes made by it uses the transaction undo feature. A transaction also has to be undone when the DBMS forces the transaction to abort. A transaction undo removes all database changes, partial of otherwise, made by the transaction.

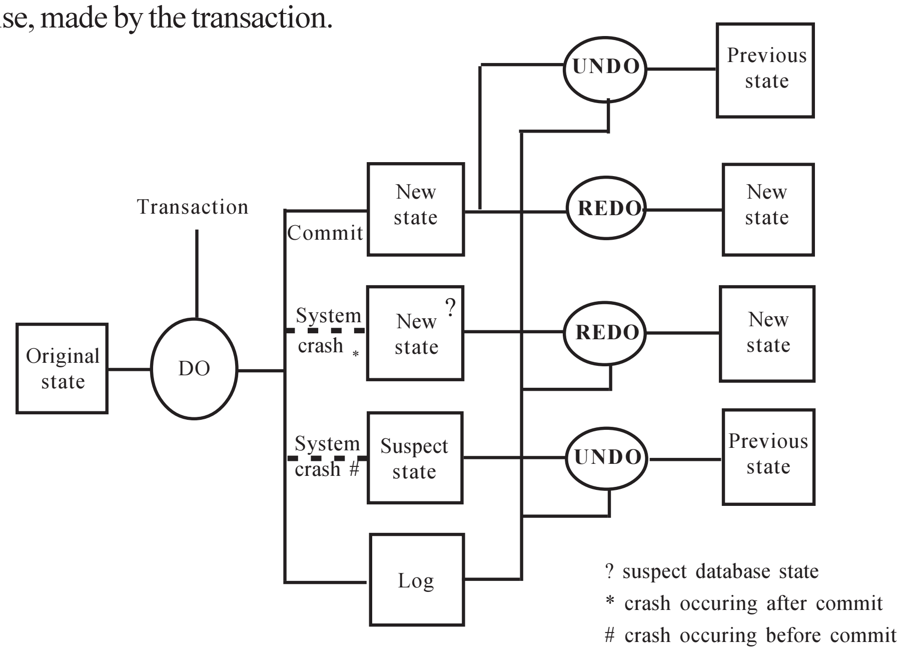


Figure 12.6(b) Do, Undo, and redo operations

Transaction Redo :

Transaction redo involves performing the changes made by a transaction that commited before a system crash. With the write-ahead log strategy, a commited transaction implies that the log for the transaction would have been written to nonvolatile storage, but the physical database may or may not have been modified before the system failure. A transaction redo modifies the physical database to the new values for a committed transaction. Since the redo operation is idempotent, redoing the partial or complete modifications made by a transaction to the physical database will not pose a problem for recovery.

Global Undo :

Transactions that are partially complete at the time of a system crash with loss of volatile storage need to be undone by undoing any changes made by the transaction. The global undo operation, intiated by the recovery system, involves undoing the partial or otherwise updates made by all uncommitted transactions at the time of a system failure.

Global Redo :

The global redo operation is required for recovery from failures involving nonvolatile storage loss. The archival copy of the database is used and all transactions committed since the time of the archival copy are redone to obtain a database updated to a point as close as possible to the time of the nonvolatile storage loss. The effects of the transaction in progress at the time of the nonvolatile loss will not be reflected in the recovered database. The archival copy of the database could be anywhere from months to days old and the number of transactions that have to be redone could be large. The log for the committed transaction needed for performing a global redo operation has to be stored on stable storage so that they are not lost with the loss of nonvolatile storage containing the physical database.

Reflecting updates to the Database and Recovery :

Let us assume that the physical database at the start of a transaction is equivalent to the current database, i.e., all modifications have been reflected in the database on the nonvolatile storage. Under this assumption. Whenever a transaction is run against a database, we have a number of options as to the strategy that will be followed in reflecting the modifications made by the transaction as it is executed. The strategies we will explore are the following :

Update in place : In this approach the modifications appear in the database in the original locations and in the case of a simple update, the new values will replace the old values.

Concurrent execution of a number of transactions implies that the operations from these transactions may be interleaved. This is not the same as serial execution of the transactions where each transaction is run to completion before the next transaction is started. Concurrent access to a database by a number of transactions requires some type of concurrency control to preserve the consistency of the database, to ensure that the modifications made by the transactions are not lost, and to guard against transations reading data that is inconsistent. The serializability criterion is used to test whether or not an interleaved execution of the operations from a number of concurrent transactions is correct. The Serializability test consists of generating a precedence graph from a interleaved execution schedule. If the precedence graph is acyclic, the schedule is serializable, which means that the database will have the same state at the end of the schedule as some serial execution of the transactions. In this chapter, we intruduce a number of concurrency control schemes.

12.7 Summary

In this chapter we discussed the recovery of the data contained in a database system after failures of various types. The aim of the recovery scheme is to allow database operations to be resumed after a failure with minimum loss of information and at an economically justifiable cost. The checkpoint information is used to limit the amount of recovery operations to be done following a system crash resulting in the loss of volatile storage. The archival database is the copy of the database at a given time stored to stable storage.

Whenever a transaction is run against a database , a number of options can be used in reflecting the modifications made by the transactions. the options we have examined are update in place and indirect update with careful replacement: the shadow page scheme and the update vai log scheme are two versions of the later. In the update in place scheme, the transaction updates the physical database and the modified record replaces the old record in the database. The write ahead log strategy is used.

12.8 Self Assessment Questions

1. Define the following terms:
   i) Write ahead log strategy
   ii) Transaction- consistent checkpoint
   iii) Transaction oriented checkpoint
   iv) Two-phase commit
2. How is the checkpoint information used in the recovery operation following a system crash?
3. What is concurrency in dbms? Explain.
4. What is lost update problem? Discuss.
5. Describe the inconsistent read problem in dbms
6. what is serializability and how it works in concurrency?
7. Write a note on precedence graph?
8. What is locking scheme and how it is utilized in database?
9. Write a note on granularity of locking
10. Describe the hierarchy of locks and intention mode locking.

′ ′ ′

Unit - 13 : Concurrency Control

Structure of the Unit

13.0 Objective

This chapter covers the basic locks such as exclusive, shared for the concurrency control of the transactions.

* Locking: shared and exclusive locks
* Two phase locking
* Time stamp based order
* Serializable schedule

13.1 Introduction

If all schedules in a concurrent environment are restricted to serializable schedules, the result obtained will be consistent with some serial execution of the transactions and will be considered correct. However, using only serial schedules unnecessarily limits the degree of concurrency. Furthermore, testing for serializability of a schedule is not only computationally expensive but it is an after-the-fact technique and im practical. Thus, one of the following concurrency control schemes is applied in a concurrent database environment to ensure that the schedules produced by concurrent transactions are serializable. The schemes we discuss are locking, timestamp-based order.

The intent of locking is to ensure serializability by ensuring mutual exclusion in accessing data-items. In the timestamp-based ordering scheme, the order of execution of the transactions is selected a priori by assigning each transaction an unique value. This value, usually based on the system clock, is called a timestamp. The values of the timestamp of the transactions determine the sequence in which transactions contesting for a given data-item will be executed. Conflicts in the timestamp scheme are resolved by abort and rollback.

13.2 Locking and types of Locking

From the point of view of locking, a database can be considered as being made up of a set of data items. A lock is a variable associated with each such data-item. Manipulating the value of a lock is called locking. The value of a lock variable is used in the locking scheme to control the concurrent access and manipulation of the associated data-item. Locking the items being used by a transaction can prevent other concurrently running transactions from using these locked items. The locking is done by a subsystem of the database management system usually called the lock manager.

So that concurrency is not restricted unnecessarily, at least two types of locks are defined: exclusive lock and shared lock.

Exclusive lock : The exclusive lock is also called an update or a write lock. The intention of this mode of locking is to provide exclusive use of the data item to one transaction. If a transaction T locks a data item Q in an exclusive mode, no other transaction can access Q, not even to read Q, until the lock is released by transaction T.

Shared lock : The shared lock is also called a read lock. The intention of this mode of locking is to ensure that the data item does not undergo any modifications while it is locked in this mode. Any number of thansactions can concurrently lock and access a data item in the shared mode, but none of these transactions can modify the data item. A data item locked in a shared mode cannot be locked in the exclusive mode until the shared lock is released by all transactions holding the lock. A data item locked in the exclusive mode cannot be locked in the shared mode until the exclusive lock on the data item is released.

The protocol of sharing is as follows. Each transaction, before accessing a data item, requests that the data item be locked in the appropriate mode. If the data item is not locked, the lock request is honored by the lock manager. If the data item is already locked, the request may or may not be granted, depending on the mode of locking requested and the current mode in which the data item is locked. If the mode of locking requested is shared and if the data item is already locked in the shared mode, the lock request can be granted. If the data item is locked in an exclusive mode, then the lock request cannot be granted, regardless of the mode of the request. In this case the requesting transaction has to wait till the lock is released.

The compatibility of a lock request for a data item with respect to its current state of locking is given in Figure 13.1. Here we are assuming that the request for locking is made by a transaction not already holding a lock on the data item.

		Current state of locking of data-item		
		Unlocked	Shared	Exclusive
Lock mode of request	Unlock		yes	yes
	Shared	yes	yes	no
	Exclusive	yes	no	no

Figure 13.1 : Compatibility of locking

If transaction T_x makes a request to lock data item A in the shared mode and if A is not locked or if it is already locked in the shared mode, the lock request is granted. This means that a subsequent request from another transaction, T_y, to lock
data item A in the exclusive mode would not be granted and transaction T_y will have to wait until A is unlocked. While A is locked in the shared mode, if transaction T_Z makes a request to lock it in the shared mode. this request can be granted. Both T_X and T_Z can concurrently use data item A.

If transaction T_x makes a request to lock data-item A in the shared mode and if A is locked in the exclusive mode, the request made by transaction T_x cannot be granted. Similarly, a request by transaction T_z to lock A in the exclusive mode while it is already locked in the exclusive mode would also result in the request not being granted, ant T_z would have to wait until the lock on A is released.

From the above we see that any lock request for a data-item can only be granted if it is compatible with the current mode of locking of the data-item. If the request is not compatible, the requesting transaction has to wait until the mode becomes compatible.

The releasing of a lock on a data-item changes its lock status. If the data-item was locked in an exclusive mode, the release of lock request by the transaction holding the exclusive lock on the data-item would result in the data-item being unlocked. Any transaction waiting for a release of the exclusive lock would have a chance of being granted its request for locking the data-item. If more than one transaction is waiting, it is assumed that the lock manager would use some fair scheduling teachnique to choose one of these waiting transactions.

If the data-item was locked in a shared mode, the release of lock request by the transaction holding the shared lock on the data-item may not result in the data-item being unlocked. This is because more than one transaction may be holding a shared lock on the data-item. Only when the transaction releasing the lock is the only transaction having the shared lock does the data-item become unlocked. The lock manager may keep a count of the number of transactions holding a shared lock on a data-item. It would increase this value by one when an additional transaction is granted a shared lock and decrease the value by one when a transaction holding a shared lock releases the lock. The data-item would then become unlocked when the number of transactions holding a shared lock on it becomes zero. This count could be stored in an appropriate data structure along with the data-item but it would be accessible only to the lock manager.

The lock manager must have a priority scheme whereby it decides whether to allow additional transactions to lock a data item in the share-mode in the following situation:

- The data-item is already locked in the shared mode.
- There is at least one transaction waiting to lock the data-item in the exclusive mode.

Allowing a higher priority to share lock requests could result in possible starvation of the transactions waiting for an exclusive lock. Similarly, the lock manager has to deal with a situation where a data-item is locked in an exclusive mode and there are transactions waiting to lock the data-item in the shared mode and the exclusive mode.

13.3 Granting of Locks

In the following discussions we assume that a transaction makes a request to lock data-item A by executing the statement Locks(A) or Lockx(A). The former is for requesting a shared lock; the latter, an exclusive lock. A lock is released by simply executing an Unlock(A) statement. We assume that the transactions are correct. In other words, a transaction would not request a lock on a data-item for which it already holds a eock, nor would a transaction unlock a data-item if it does not hold a lock for it.

A transaction may have to hold onto the lock on a data-item beyond the point when it last needs it to preserve consistency and avoid the inconsistent read problems . We illustrate this point by reworking the example of Figure 12.7 here each transaction request locks for the data-items A and B : transaction T_5 in exclusive mode and transaction T_6 in shared mode. The transactions with the lock requests are given in Figure 13.2 As shown there, the transactions attempt to release the locks on the data-items as soon as possible.

Now consider Figure 13.3 which given a possible schedule of execution of the transactions of Figure 13.2 The locking scheme did not resolve the inconsistent read problem; the reason is that transactions T_5 and T_6 are performing an operation made up of many steps and all these have to be executed in an atomic manner. The database is in an inconsistent state after transaction T_5 has taken 100 units from A but not added it to B. Allowing transaction T_6 to read the vlues of A and B before transaction T_5 is complete leads to the inconsistent read problem.

A possible solution to the inconsistent read problem is shown in Figure 13.4 Here transactions T_5 and T_6 are rewritten as transactions T_{20} and T_{21}. The possible schedules of concurrent executions of these transactions are shown in Figure 13.5 and 13.6 Both of these solutions extend the period of time for which they keep some data-items locked even though the transactions no longer need these items. This extended locking forces a serialization of the two transactions and given correct results.

Transaction T_5	**Transaction T_6**
Lockx (A)	**Lockx** (Sum)
Read (A)	Sum : = 0
A : = A - 100	**Locks** (A)
Write (A)	**Read** (A)
Unlock (B)	Sum : = Sum + A
Lockx (B)	**Unlock** (A)
Read (B)	**Locks** (B)
B : - B + 100	**Read** (B)
Write (B)	Sum : = Sum + B
Unlock(B)	**Write** (Sum)
	Unlock (B)
	Unlock (Sum)

Figure 13.2 : Two Transactions with Lock Requests

	Shedule	**Transaction T_5**	**Transaction T_6**
	Lockx (Sum)		**Lockx** (Sum)
	Sum : = 0		Sum : = 0
	Locks(A)		**Locks** (A)
	Read(A)		**Read** (A)
	Sum : = Sum + A		Sum : = Sum + A
T	**Unlock** (A)		**Unlock** (A)
i	**Lockx**(A)	**Lockx** (A)	
m	**Read**(A)	**Read** (A)	
e	A : = A - 100	A : = A – 100	
	Write (A)	**Write** (A)	
	Unlock (A)	**Unlock**(A)	
	Lockx (B)	**Lockx** (B)	
	Read (B)	**Read** (B)	
	B : = B + 100	B : = B + 100	
	Write (B)	**Write** (B)	
	Unlock (B)	**Unlock** (B)	
	Locks (B)		**Locks**(B)
	Read (B)		**Read** (B)
	Sum : = Sum + B		Sum : = Sum + B
	Write (Sum)		**Write** (Sum)
	Unlock (B)		**Unlock** (B)
	Unlock (Sum)		**Unlock** (Sum)

Figure 13.3 : A possible Schedule Causing an Inconsistent Read

Transaction T_{20}	**Transaction T_{21}**
Lockx (A)	**Lockx** (Sum)
Read (A)	Sum : = 0
A : = A - 100	**Locks** (A)
Write (A)	**Read** (A)
Lockx (B)	Sum : = Sum + A
Unlock (A)	**Locks** (B)
Read (B)	**Read** (B)
B : = B + 100	Sum : = Sum + B
Write (B)	**Write** (Sum)
Unlock(B)	**Unlock** (B)
	Unlock (A)
	Unlock (Sum)

Figure 13.4 : Transactions Locking All Items Before Unlocking

	Shedule	**Transaction T_{20}**	**Transaction T_{21}**
	Lockx (Sum)		**Lockx** (Sum)
	Sum : = 0		Sum : = 0
	Locks(A)		**Locks** (A)
	Read(A)		**Read** (A)
	Sum : = Sum + A		Sum : = Sum + A
T	**Locks** (B)		**Locks** (B)
i	**Read** (B)		**Read** (B)
m	Sum : = Sum + B		Sum : = Sum + B
e	**Write** (Sum)		**Write** (Sum)
	Unlock (B)		**Unlock** (B)
	Unlock (A)		**Unlock** (A)
	Unlock (Sum)		**Unlock** (Sum)
	Lockx (A)	**Lockx** (A)	
	Read (A)	**Read** (A)	
	A : = A – 100	A : = A – 100	
	Write (A)	**Write** (A)	
	Lockx (B)	**Lockx** (B)	
	Unlock (A)	**Unlock** (A)	
	Read (B)	**Read** (B)	
	B : = B + 100	B : = B + 100	
	Write (B)	**Write** (B)	
	Unlock (B)	**Unlock** (B)	

Figure 13.5 : A Posible Solution to The Inconsistent Read Problem

Some data-items locked even though the transactions no longer need these items. This extended locking forces a serialization of the two transactions and gives correct results.

13.4 Two-Phase Locking

The correctness of the schedules of Figure 13.5 and 13.6 and of the transactions in Figure 13.4 lead us to the observation that both these solutions involve transactions whose locking and unlocking operations are monotonic, in the sense that all locks are first acquired before any of the locks are released. Once a lock is released; no additional locks are requested. In other words, the release of the locks is delayed until all locks on all data-items required by the transaction have been acquired.

This method of locking is called two-pha1se locking. It has two phases, **a growing phase** where in the number of locks increase from zero to the maximum for the transaction, and **contracting phase** where in the number of locks held decreases from the maximum to zero. Both of these phases are monotonic; the number of locks are only increasing in the first phase and decreasing in the second phase. Once a transaction starts releasing locks, it is not allowed to request any further locks. In this way a transaction is obliged to request all locks it may need during its life before it releases any. This leads to a possible lower degree of concurrency.

	Shedule	**Transaction T_{20}**	**Transaction T_{21}**
	Lockx (A)	**Lockx** (A)	
	Read (A)	**Read** (A)	
	A : = A – 100	A : = A – 100	
	Write (A)	**Write** (A)	
	Lockx (B)	**Lockx** (B)	
T	**Unlock** (A)	**Unlock** (A)	
i	**Read** (B)	**Read** (B)	
m	B : = B + 100	B : = B + 100	
e	**Write** (B)	**Write** (B)	
	Unlock (B)	**Unlock** (B)	
	Lockx (Sum)		**Lockx** (Sum)
	Sum : = 0		Sum : = 0
	Locks (A)		**Locks** (A)
	Read (A)		**Read** (A)
	Sum : = Sum + A		Sum : = Sum + A
	Locks (B)		**Locks** (B)
	Read (B)		**Read** (B)
	Sum : = Sum + B		Sum : = Sum + B
	Write (Sum)		**Write** (Sum)
	Unlock (B)		**Unlock** (B)
	Unlock (A)		**Unlock** (A)
$\downarrow$	**Unlock** (Sum)		**Unlock** (Sum)

Figure 13.6 : Another Solution to The Inconcistent Read Problem.

The two-phase locking protocol ensures that the schedules involving transactions using this protocol will always be serializable. For instance, if S is a schedule containing the interleaved operations from a number of transactions, T_1, T_2,......... T_k and all the transactions are using the two-phase locking protocol, schedule S is serializable. This is because if the schedule is not serializable, the precdence graph for S will have a cycle made up of a subset of {T_1, T_2, ...,T_k} Assume the cycle consists of $T_a \rightarrow T_b \rightarrow T_c ....T_x \rightarrow T_a$ This means that a lock operation by T_6 is followed by an unlock operation by T_a; a lock operation by T_c is followed by an unlock operation by T_b.........., and finally a lock operation by T_a is followed by an unlock operation by T_x. However this is a contradiction of the assertion that T_a is using the two phase protocol. Thus the assumption that there was a cycle in the precedence graph is incorrect and hence S is serializable.

The transactions of Figure 13.4 use the two-phase locking protocol, and the schedules derived from the concurrent execution of these transactions given n Figures 13.5 and 13.6 are serializable. However, the transactions of Figure 13.2 do not follow the two-phase locking protocol and the schedule of Figure 13.3 is not serializable.

13.5 Time Stamp-Based Order

In the timestamp-based method, a serial order is created among the concurrent transaction by assigning to each transaction a unique nondecreasing number. The usual value assigned to each transaction is the system clock value at the start of the transaction, hence the name **timestamp ordering**. A variation of this scheme that is used in a distributed environment includes the site of a transaction appended to the system wide clock value. This value can then be used in deciding the order in which the conflict between two transactions is resolved. A transaction with a smaller timestamp value is considered to be an "older" transaction thann another transaction with a larger timestamp value.

The serializability that the system enforces is the chronological order of the timestamps of the concurrent transactions. If two transaction T_i and T_j with the time stamp values t_i and t_j respectively, such that $t_i < t_j$, are to run concurrently, then the schedule produced by the system is equivalent to running the older transaction T_i first, followed by the younger one, T_j.

The contention problem between two transactions in the timestamp ordering system is resolved by rolling back one of the conflicting transactions. A conflict is said to occur when an older transaction tries to read a value that is written by a younger transaction or when an older transaction tries to modify a value already read or written by a younger transaction. Both of these attempts signify that the older transaction was "too late" in performing the required read/write operations and it could be using values from different "generations" for different data-items.

In order for the system to determine if an older transaction is processing a value already read by or written by a younger transaction, each data-item has, in addition to the value of the item, two timestamps: a **write timestamp** and a **read timestamp**. Data-item X is thus represented by a triple X: $\{x, W_x, R_x\}$ where each component of the triple is interpreted as given below :

x, the value of the data-item X

W_x, the write timestamp value, the largest timestamp value of any transaction that was allowed to write a vlue of X.

R_x, the read timestamp value, the largest timestamp value of any transaction that was allowed to read the current value X.

Now let us see how these timestamp values find their way into the data structure of a data-item and how all these values are modified. A transaction Ta with the timestamp value of t_a issues a read operation for the data-item X with the values $\{x, W_x, R_x\}$.

- This request will succeed if t_a > Wx since transaction T_a is younger than the transaction that last wrote (or modified) the value of X. Transaction T_a is allowed to read the value x of X and if the value t_a is larger than R_x, then t_a becomes the new value of R_x.
- This request will fail if $t_a \leq W_x$, i.e. transaction T_a is an older transaction than the last transaction that wrote the value of X.

The failure of the read request is due to the fact that the older transaction was trying to read a value that had been overwritten by a younger transaction. Transaction T_a is too late to read the previous outdated value and any other values it has acquired are likely to be inconsistent with the updated value of X. It is thus safe to abort and roll back T_a, T_a, is assigned a new timestamp and restarted.

A transaction T_a with the timestamp value of t_a issues a write operation for the data-item X with the values {x, W_x, R_x}

- If $t_a \geq W_x$ and $t_a \geq R_x$, i.e. both the last transaction that updated the value of X and the last transaction that read the value of X are older than transaction T_a, then T_a is allowed to write the value of X and t_a becomes the current value of W_x, the write timestamp.
- If $t_a < R_x$, it means that a younger transaction is already using the corrent value of X and it would be an error to update the value of X. Transaction T_a is not allowed to modify the value of X. T_a is rolled back and its timestamp is reset to the current system-generated timestamp value and restarted.
- If $R_x < t_a < W_x$. this means that a younger transaction has already updated the value of X, and the value that T_a is writing must be based on an obsolete value of X and is obsolete. Transaction T_a is not allowed to modify the value of X; its write operation is ignored.

The reason for ignoring the write operation in the last alternative is as follows. In the serial order of transaction processing, transaction T_a with the timestamp of t_a wrote the value for the data-item X. This was followed by another write operation to the same data-item by a younger transaction with a timestamp of W_x. No transaction read the data-item between the writing by T_a and the time W_x. Hence, ignoring the writing by T_a indicates that the value written by T_a was immediately overwritten by a younger transaction at time W_x.

Let us illustrate the timestamp ordering by considering transactions T_{22} and T_{23} given below in Figure 13.7. Each of these transactions has a local variable Sum and the intent is to show a user the sum of two data-items A and B. However, transaction T_{23} not only reads these values, it also transfers 100 units from A to B and writes the modified values to the database. Now let us suppose that $t_{23} > t_{22}$. This means that transaction T_{23} is younger than transaction T_{22}. Also, let the data-items A and B be stored as follows (here the Wi's and Ri's have some values assumed to be less then t_{22} and t_{23}) :

A : 400, W_a, R_a B: 500, W_b, R_b

Example 13.1 : Consider the transaction of Figure 13.7 in the schedule given in Figure E transaction T_{22} (t_{22}) and T_{23} (t_{23}) run concurrently and produce the correct result. A similar serialiable schedule could have been obtained using the two phase locking protocol.

Step	Shedule	Transaction T_{22}	Transaction T_{23}
1	Sum : = 0	Sum : = 0	
2	**Read** (A)	**Read** (A)	
3	Sum : = Sum + A	Sum : = Sum + A	
4	Sum : = 0		Sum : = 0
5	**Read** (A)		**Read** (A)
6	A : = A – 100		A : = A – 100
7	**Write** (A)		**Write** (A)
8	**Read** (B)	**Read** (B)	
9	Sum : = Sum + B	Sum : = Sum + B	
10	**Show** (Sum)	**Show** (Sum)	
11	Sum : = Sum + A		Sum : = Sum + A
12	**Read** (B)		**Read** (B)
13	B : = B + 100		B : = B + 100
14	**Write** (B)		**Write** (B)
15	Sum : Sum + B		Sum : = Sum + B
16	**Show** (Sum)		**Show** (Sum)

Figure 13.7 : Serializable schedule based on timestamp scheme

The steps of the schedule of Figure E cause the following modifications to the triple for A and B.

Initially	A : 400, W_a, R_2	B : 500, W_b, R_b
After step 2	A : 400, W_a, t_{22}	B : 500, W_b, R_b
After step 5	A : 400, W_a, t_{23}	B : 500, W_b, R_b
After step 7	A : 300 t_{23}, t_{23}	B : 500, W_b, R_b
After step 8	A : 300, t_{23}, t_{23}	B : 500, W_b, t_{22}
After step 10	the value displayed will be 900	
After step 12	A : 300, t_{23}, t_{23}	B : 500, W_b, t_{23}
After step 14	A : 300, t_{23}, t_{23}	B : 500, t_{23}, t_{23}

After step 14 the value displayed will be 900

In the following example we illustrate a schedule where the older transaction is rolled back.

Example 13.2 : In the example illustrated in Figure G, we have three transactions. T_{24}, T_{25}, and T_{26} with timestamp value of T_{24}, T_{25} and T_{26} respectively ($t_{24}<t_{25}<t_{26}$). Note that transactions T_{24} and T_{26} are write-only with respect to data-item B.

Step	**Shedule**	**Transaction T_{24}**	**Transaction T_{25}**	**Transaction T_{26}**
1	**Read** (A)	**Read** (A)		
2	A : = A + 1	A : = A + 1		
3	**Write** (A)	**Write** (A)		
4	**Read** (C)		**Read**(C)	
5	C : = C * 3		C : = C * 3	
6	**Read** (C)			**Read** (C)
7	**Write** (C)		**Write** (C)* cause a rollback of transaction T_{25}	
8	C : = C * 2			C : = C * 2
9	**Write** (C)			**Write** (C)
10	B : = 100			B : = 100
11	**Write** (B)			**Write** (B)
12	B : = 150	B : = 150		
13	**Write** (B)	**Write** (B)** cause the write operation to be ignored		
14	**Read** (C)		**Read** (C)	
15	C : = C * 3		C : = C * 3	
16	**Write** (C)		**Write** (C)	

Initially	A : 10, W_a, R_a	B : 50, W_b, R_b	C : 5, W_c, R_c
After step 1	A : 10, W_a, t_{24}	B : 50, W_b, R_b	C : 5, W_c, R_c
After step 3	A : 11, t_{24}, t_{24}	B : 50, W_b, R_t	C : 5, W_c, R_c
After step 4	A : 11, t_{24}, t_{24}	B : 50, W_b, R_t	C : 5, W_c, t_{25}
After step 5	A : 11, t_{24}, t_{24}	B : 50, W_b, R_t	C : 5, W_c, t_{25}
After step 6	A : 11, t_{24}, t_{24}	B : 50, W_b, R_b	C : 5, W_c, t_{26}

At step 7 transaction t_{25} with a timestamp value of t25 attempts to write the value of C : however, since the read timestamp value of is t26, which is greater than t25, transaction T25 would be rolled back; the transaction would be reassigned a timestamp value of, say, t25 (>t26) and rerun at step 14.

After step 9	A : 11, t_{24}, t_{24}	B : 50, W_b, R_b	C : 10, t_{26}, t_{26}
After step 11	A : 11, t_{24}, t_{24}	B : 100, t_{26}, R_b	C : 10, t_{26}, t_{26}

At step 13, the attempt by transaction T_{24} to write a value of B is ignored since t_{24} the timestamp of T_{24}, is less than the write timestamp (t_{26}) of B, and greater than the read timestamp value (R_b) of B.

After step 14	A : 11, t_{24}, t_{24}	B : 100, t_{26}, Rb	C : 10, t_{26}, t_{25}'
After step 16	A : 11, t_{24}, t_{24}	B : 100, t_{26}, Rb	C : 30, t_{25}', t_{25}'

Figure 13.9 : Another serializable schedule.

It is obvious from the above examples that the timestamping scheme ensures serializability without waiting but causes transactions to be rolled back. Since there is no waiting there is no possibility of a dedlock. However, when transactions are rolled back, a cascading rollback may be needed. For instance, if transaction T22 had written a value for a data-item Q before it was rolled back, this data-item value must be restored to its old value. If another transaction, T', had used the modified value of the data-item Q, transaction T' has to be rolled back as well.

Transaction T_{22}	**Transaction T_{23}**
Sum =0;	Sum=0;
Read(A)	**Read**(A)
Sum:= Sum + A	A= A - 100
Read(B)	**Write**(A)
Sum = Sum +B	Sum = Sum +A
Show(sum)	**Read**(B)
	B: = B + 100
	Write(B)
	Sum = Sum + B
	Show(Sum)

tep	**Shedule**	**Transaction T_{22}**	**Transaction T_{23}**
1	Sum : = 0	Sum : = 0	
2	Sum : = 0		Sum : = 0
3	**Read** (A)		**Read** (A)
4	A : = A – 100		A : = A – 100
5	Write (A)		Write (A)
6	Read (A)	Read(A)* causes a rollback of T_{22}	
7	Sum : = Sum + A		Sum : = Sum + A
8	Read (B)		Read (B)

9	B : = B + 100		B : = B + 100
10	Write (B)		Write (B)
11	Sum : = Sum + B		Sum : = Sum + B
12	Show (Sum)		Show (Sum)
13	Sum : = 0	Sum : = 0 with a timestamp t_{22}' (> t_{23})	
14	Read (A)	Read (A)	
15	Sum : Sum + B	Sum : = Sum + A	
16	Read (B)	Read (B)	
17	Sum : = Sum + B	Sum : = Sum + B	
18	Show (Sum)	Show (Sum)	

Figure 13.8 : Serializable Schedule Produced After a Rollback

Consider the schedule shown in Figure F. Transacation T_{22} is rolled back and rerun after step 6. When it is rolled back, a new timestamp value t_{22}' which would be greater than t_{23}, is assigned to it. The sequence of changes is given below :

Initially	A : 400, W_a, R_a	B : 500, W_b, R_b
After step 3	A : 400. W_a, t_{23}	B : 500, W_b, R_b
After step 5	A : 300, t_{23}, t_{23}	B: 500, W_b, R_b
After step 6	A : 300, t_{23}, t_{23}	B : 500, W_b, R_b*

(*cause a rollback of T_{22} which would be reassigned a new timestamp (t_{22}^1, > t_{23}) and would be reesecuted)

After step 8	A : 300. t_{23}, t_{23}	B : 500, W_b, t_{23}
After step 10	A : 300, t_{23}, t_{23}	B: 600, t_{23}, t_{23}
After step 12	the value displayed will be 900	
After step 14	A : 300. t_{23}, t_{22}	B : 600, t_{23}, t_{23}
After step 16	A : 300, t_{23}, t_{22}	B: 600, t_{23}, t_{23}

After step 18 the value displayed will be 900

13.6 Summary

The concurrency control scheme ensures that the schedule that can be produced by a set of concurrent transactions will be serialable. The locking protocol, timestamp based ordering, optimizing scheduling, and multiversion technique are used in concurreny control. In the locking protocol, before a transaction can access a data-item, it is required to lock the data-item in an appropriate mode. It releases the lock onthe data-item once it no longer needs it. In the locking scheme, the two-phase locking protocol is usually used. The principle characteristic of the two-phase locking protocol is that all locks are acquired before a transaction starts releasing any locks. This ensures seriazability; however, deadlock is possible.

In timestamp-based ordering, each transaction is assigned an unique idetified, which is usually based on the system clock. This identifier is called a timestamp and the value of the time-stamp is used to schedule contending transactions.

13.7 Self Assessment Questions

1. What is shared locks?
2. Explain two phase locking
3. What is concurrency control?
4. Explain time-stamp based ordering
5. How is granting locks?
6. What is the role of exclusive locks in two phase locking
7. Explain the growing phase of locking.
8. How timestamp ordering ensure concurrency control?

′ ′ ′

Unit - 14 : Emerging Trends in Database Management System - I

Structure of the Unit

14.0 Objective

At the end of this unit, you should be able to -

- Describe the concepts of distributed databases
- Describe the concepts of object oriented DBMS
- Describe the client server systems
- Describe the failure and recovery

14.1 Introduction

Database management systems are standard tools that enable the storage and retrieval of data within modern information systems. Applications in domains such as Multimedia, Geographical Information Systems, and digital libraries demand a completely different set of requirements in terms of the underlying database models. The conventional relational database model is no longer appropriate for these types of data. Furthermore the volume of data is typically significantly larger than in classical database systems. Finally, indexing, retrieving and analyzing these data types require specialized functionality.

14.2 Introduction to Distributed Databases

A Distributed database system is a database in which the data is stored at several computers that are located at geographically distributed locations connected through a network. Each site or node has its own database management system, transaction management software which also includes local logging, logging and recovery. The characteristics of distributed databases are:

- The computers are connected through some communication media.
- Such systems do not share the disks or memory.
- The applications can access data stored at local as well as at remote locations.
- Distributed databases show data independence that is the user can specify the query without specifying the location at which data is stored.Distributed transaction atomicity is also a characteristic of distributed databases as all the changes to the database are made permanent only if the transaction commits and no changes are made if the transaction aborts.

The computers at one site are connected to the computers located at various other sites as shown in Figure 14.1

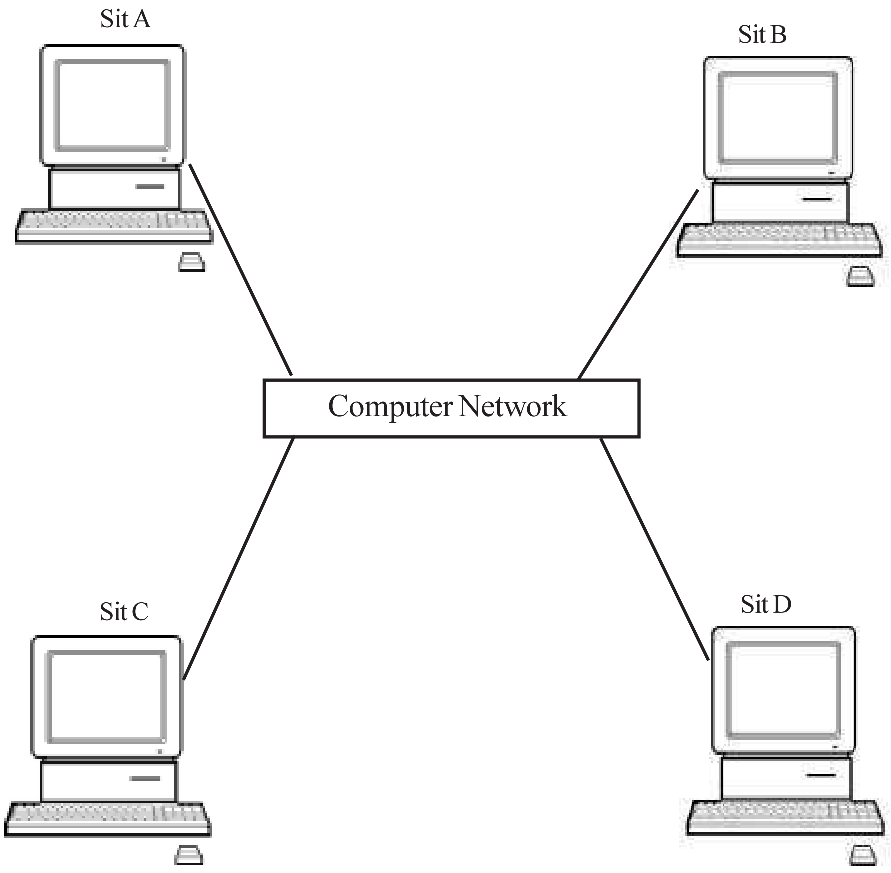
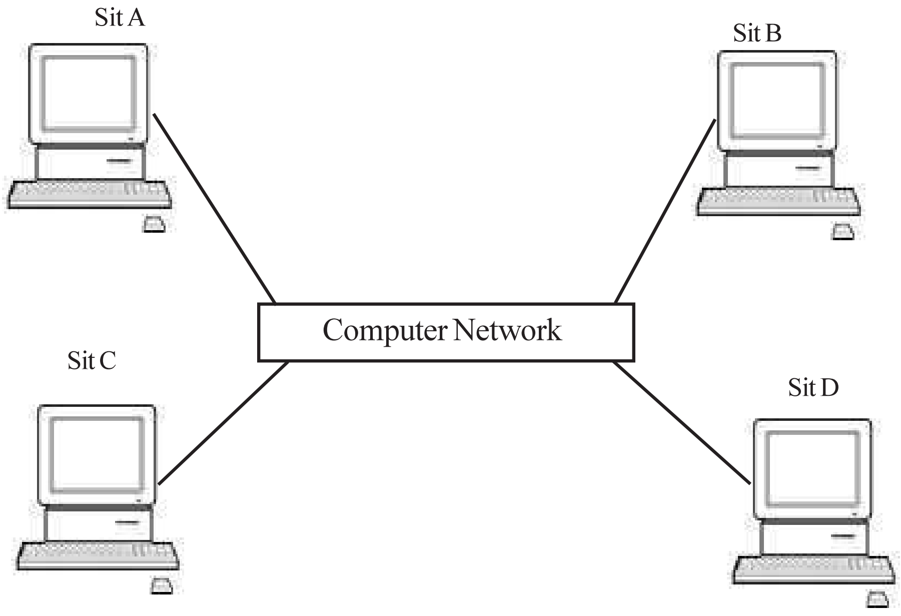


Figure 14.1: A Distributed Network

The distributed databases are of following types :

- **Homogeneous Distributed Databases :** Such kind of databases serve the purpose of location transparency i.e. all the sites have identical DBMS software and all the clients are aware of one another. Also the clients cooperate with each other
- **Heterogeneous Distributed Databases :** In a heterogeneous distributed database, different sites have different DBMS and may use different schemas and software. Such sites may not be aware of one another and have cooperation to only some extent. Such a system is also known as multi-database system or federated database system.



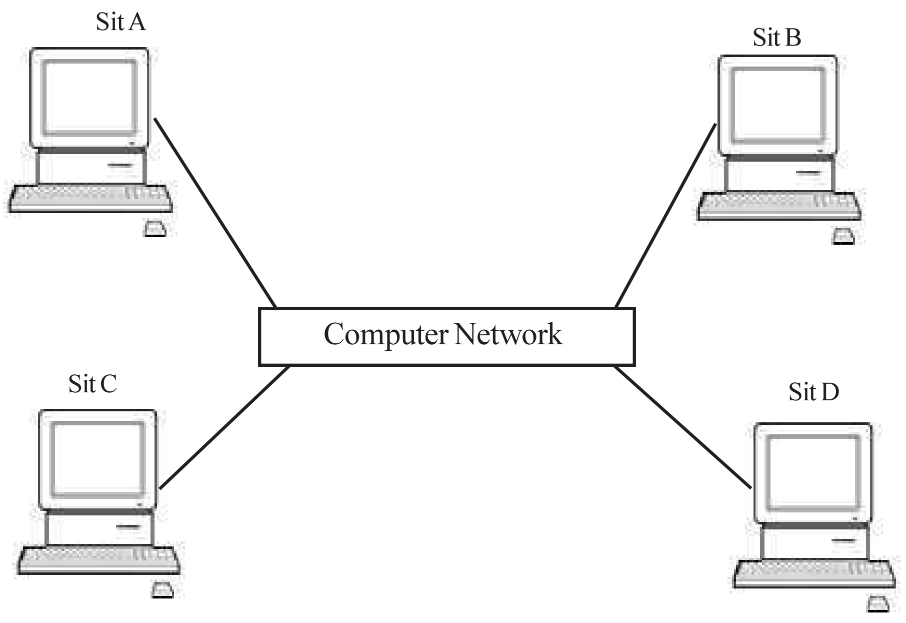


Figure 14.2: Heterogeneous v/s Homogeneous Distributed Databases

A distributed database must have some additional capabilities as compared to a centralized system. Some of these capabilities are as follows:

- **Network Transparency :** The user must not be concerned about the details of the locations where data is stored. This is further categorized into Location Transparency in which data is retrieved independent of the location and Naming transparency which indicates that once a name has been specified, the data can be accessed from anywhere.
- **Replication Transparency :** Refers to the fact that multiple copies of the data may be stored at several locations and the user is unaware of the details.
- **Fragmentation Transparency :** This can be further classified into :
  1. Horizontal fragmentation: where the database tuples or rows are distributed.
  2. Vertical fragmentation: where the database columns are distributed.
  3. Hybrid fragmentation includes an intermix of the above two types of fragmentation.
- **Local Autonomy and Independence :** The data stored at one location must be independent of the data stored at other locations.
- **Distributed Query Processing and Transaction Management :** The data stored at various locations is synchronized so as to maintain the integrity of the database. A distributed database must be independent of the hardware, operating system, network and DBMS that is used.
- **Distributed Catalog Management :** The directory or catalog contains metadata about the entire database.
- **Security :** Since a distributed database contains several computers connected at several locations, it should be secure enough. Also, access and authorization to the database must be checked.

Because of its distributed nature and its capability to store and access data at several locations, a distributed database has several advantages and disadvantages. The advantages include, sharing of data resulting into increased availability, efficiency and performance. Due to its extensible nature, a distributed database is also scalable. On the contrary it suffers from the disadvantage that recovery from failure becomes more complex due to its distributed nature. Again there can be breach of security due to increased transparency. Other disadvantages include increased cost, increased overhead of creation and maintenance of the database and lack of proper standards for synchronizing various sites.

A distributed database may have a varied architecture. Some of the architectures that are available in distributed databases are client/server architecture, collaborating server and middleware systems. The client server architecture has a collection of clients connected to a server that fulfils the requests of the clients. The clients provide the user interface and server is responsible for managing data and executing transactions as shown in Figure 14.3.

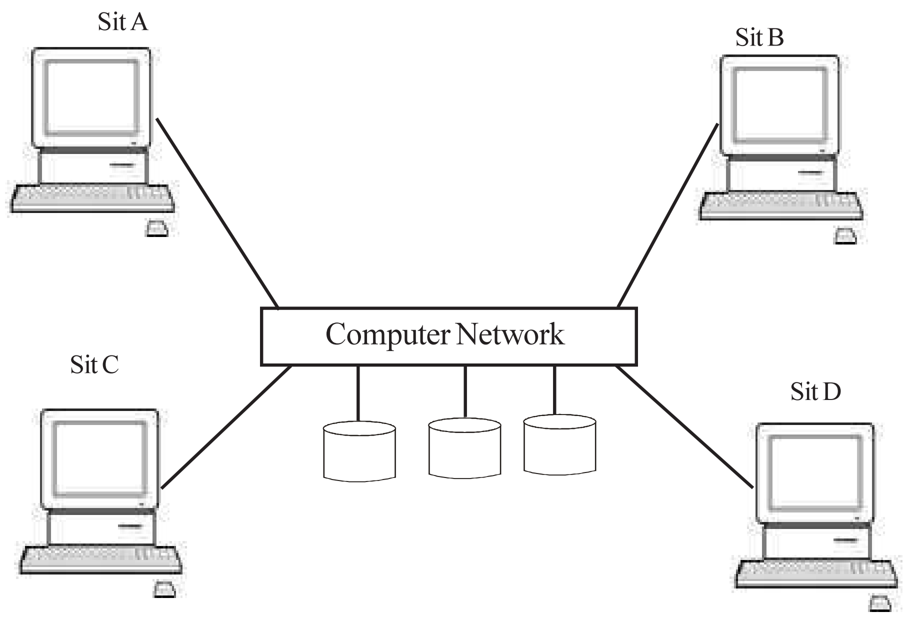


Figure 14.3: Client Server Architecture

The client server architecture is relatively simple in implementation and is comparatively less expensive but an increase in the number of client computers may increase the cost and complexity of the system. This architecture also suffers from the drawback that if the server breaks down then the entire system becomes difficult to manage. To overcome this problem, the architecture of Collaborating server systems may be used. In this architecture several servers may cooperate with each other to solve the client requests.

Another architecture that may be used for distributed databases is middleware systems. In this architecture, a layer of software called the middleware is used that coordinates the execution of queries and transactions across multiple servers as illustrated in Figure 14.4.

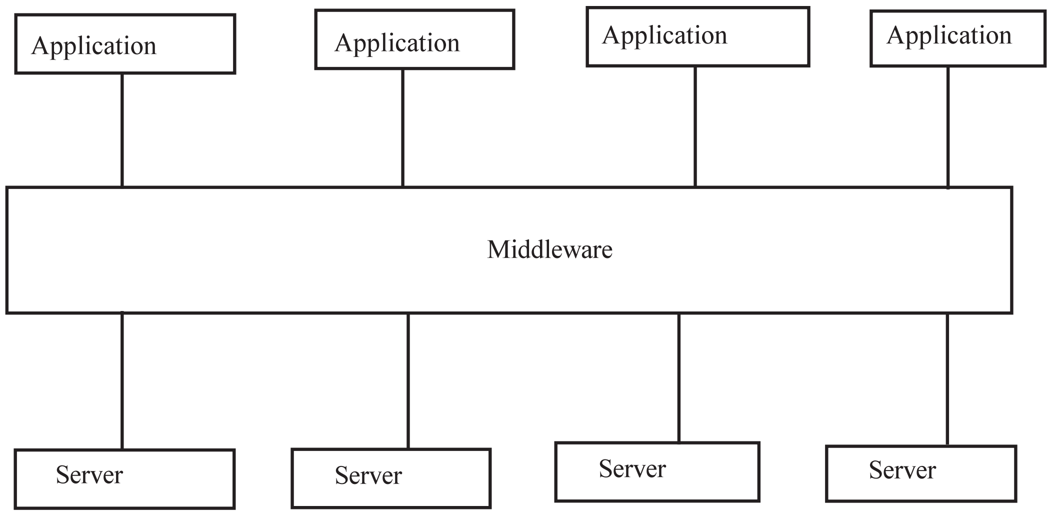


Figure 14.4: Middleware Architecture

Due to the distributed nature of the database several concurrency related problems arise. For example if one site fails, its database must be recovered to the state it was earlier in. Similarly, if a communication link fails then two or more sites may be disconnected from each other. In an extreme case the network may be partitioned into two. Another problem that arises in distributed databases is distributed commit i.e. if a site wants to commit a transaction that is stored at multiple locations and some of the participating sites fail. Two-phase commit protocol or three-phase commit protocol may be used to solve this problem. Two-phase commit (2PC) protocol has two phases: voting phase and decision phase. In the voting phase, the coordinator asks all the participating sites whether they wish to commit and in the decision phase, a decision whether to commit or abort all the sites is made. This protocol suffers from the drawback that if the coordinator fails then all the sites are blocked. Three phase commit protocol (3PC) is an extension of 2PC.This protocol postpones the decision to commit until a specified number of sites commit the transaction. 3PC has an additional phase of pre-commit in which even if the coordinator fails, the new coordinator checks if one of the site has the decision of the old coordinator.

14.3 Distributed Database Architecture

A distributed database may access data from remote or local databases. A Homogeneous distributed database system may have the same database whereas a heterogeneous distributed database may have different databases on different systems.

A distributed homogeneous database has the same database installed on different computers that are available on one or more machines. A single application may access different databases simultaneously in the distributed environment. The location and other details are transparent to the clients connected to the distributed databases. A single copy of all the database objects may be stored at distributed locations or in some cases replication of data may be done. The term replication refers to the fact that multiple copies of the data are maintained and stored at several locations. Replication may be done to ensure availability of data even if one site fails and thus improves the performance of the database. In such a case the replicated data remains available in case a site fails.

In a heterogeneous database, at least one of the databases is of different type as opposed to other sites in the distributed environment. In such cases, a gateway may be required to access databases of different types.

Some more issues that need to be addressed in a distributed environment include network authentication so that only valid users can access the network. There may be a global user that can access all the databases in the distributed environment, or there may be local users accessing the data at local sites. In case of global users, authentication needs to be done so that it can be checked which databases the user can access, what are the roles corresponding to the databases and what schema the user can connect to.

14.4 Object Oriented Database Management System

Several database models like hierarchical, network and relational models were suggested. However due to the increased complexity and the need of flexibility led to the development of Object oriented data model (OODM) and Object-relational data model (ORDM) (or Extended relational data model). These models support the concept of object oriented programming. Apart from providing the capabilities of traditional databases like persistence, concurrency control, recovery, querying, atomicity, durability etc., Object oriented databases support the properties of Object oriented programming like encapsulation, inheritance, polymorphism etc. As opposed to relational databases that support simple data types, Object based models support complex data types.

Object Oriented Database Management System (OODBMS) is designed to manage the Object oriented database (OODB). Some of the popular OODBMS include Orion, IRIS, Versant and Vbase. An OODBMS must have several features to support Object oriented design. These features include, support for encapsulation, classes and object. Such classes must also provide inheritance. The Object identity indicates the existence of the object. An object may show its identity as:

- Intra-procedure. The object exists in a particular procedure only
- Intra-program. Such an object exits throughout the program
- Inter-program. Such an object is accessible across several programs
- Persistent. These object persist even after the completion of the program

Object Data Management Group (ODMG) was a consortium developed for object oriented DBMSs. This standard provides the object model, Object Definition Language (ODL) and Object Query Language (OQL). This standard was designed to provide portable applications that could store objects in any database management system. The Object Definition Language designed in a manner similar to the Data definition Language of RDBMs is independent of any kind of programming language. Object Query Language is the query language defined for ODMG binding with programming languages like SmallTalk, Java and C++.

14.5 Client Server Systems

Apart from the general requirements of a computer system like the network software and the operating system, a client server system also has following components:

- Client, the resource user and
- Server, the resource and service provider

The applications are deployed on the client whereas the database management system resides on the server. The applications residing on the client make requests for services while the database on the server processes those requests and sends the results back to the client. The client server system can be scaled horizontally by adding or removing clients or vertically by migrating to a larger and faster server machines. Sometimes more than one server may be added to the architecture.

A Client server system consists of clients that are intelligent workstations through which the user can interact with the system, the server processing the requests, the communication networks connecting the clients and the server and several applications interacting with the clients and the server. The general architecture of the client server system is shown in Figure 14.5. Often, but not always several clients may be connected to a single server.

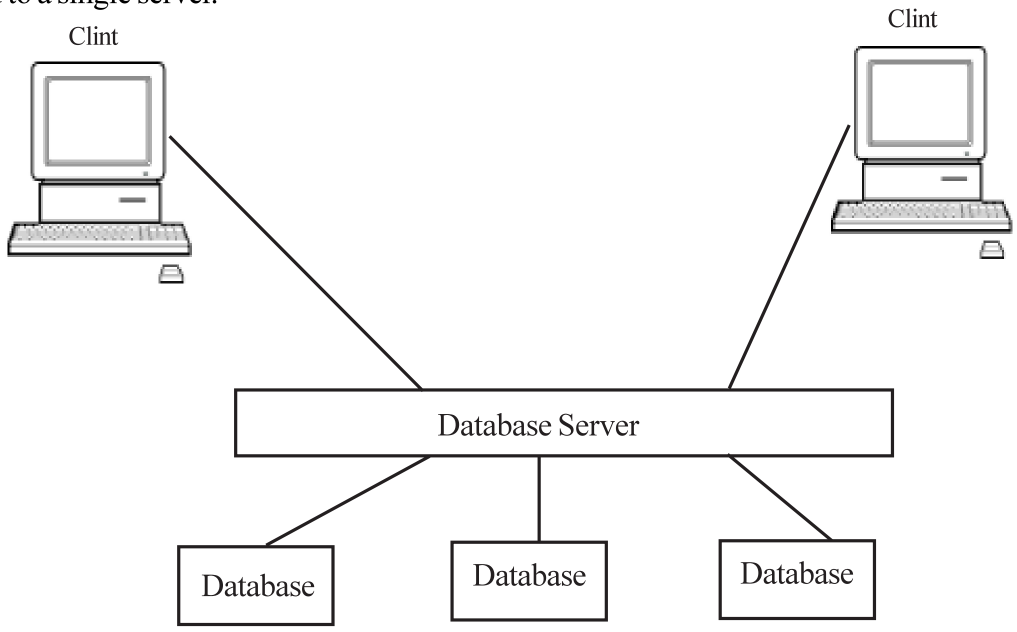


Figure 14.5 : Client Server System

The clients have their own operating system and run one or more applications using the client machine's CPU, memory etc. The application communicates with the database system residing on the server using the database driver which acts as the middleware to establish a connection with the DBMS over the network.

The server machines are characterized by the database management system. These systems also have a listening daemon that accepts connections from the clients. Apart from the client and the server, the middleware plays an important role. The middleware is a small portion of software that sits between the client and the server, establishes the connection and sends requests and responses between the two. Such kind of architecture is described as two-tier architecture.

Sometimes, an application server is added to the client-server architecture to make it three-tier architecture. Some of the advantages of such architecture include scalability, flexibility, reduced cost, improved services to the clients and competitive advantage. The middleware may interact through Remote Procedure Calls (RPC) or Message oriented middleware (MOM). Several commonly used client-server middleware include application program interface (API), Open database connectivity (ODBC), java database connectivity (JDBC), common object request broker architecture (CORBA) and Distributed Component Object Model (DCOM). Remote procedure calls provide a transparent mechanism such that a client communicates as if it is directly communicating with the server. Whereas message oriented middleware allows many to many communication via message queues as shown in Figure 14.6.

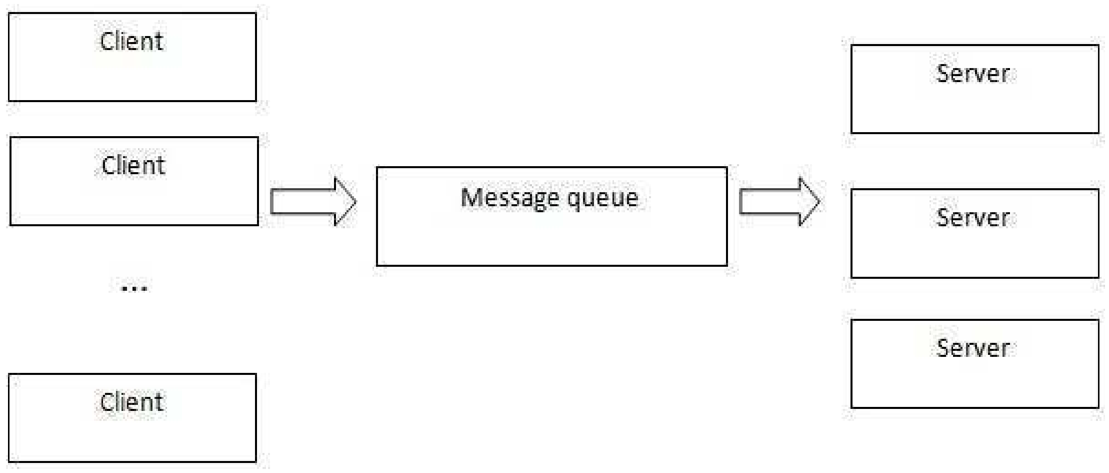


Figure 14.6: Message Oriented Middleware

Different types of software tools that are included in the development of client-server database systems include client builders used to develop the clients, database programming and administration tools that are used to develop and maintain the database and CASE/data modeling tools that help in the design and implementation of the database.

Like any other system, client server system also has its own advantages and drawbacks. Some of the advantages of the client server system include:

- Increased modularity
- Simple to implement
- Economy of communication
- In most of the cases it is hardware and software independent
- Improved functionality and efficiency
- Improved performance

- Due to the maintenance of the database on the server, it is more robust

The disadvantages of such a system are as follows:

- Difficult failure detection due to increased modularity
- Security issues may arise with the increase in the number of clients
- Network traffic may cause problems when the number of clients is large

Some of the commonly used database management systems like Oracle Server, Oracle Developer, Sybase Adaptive Server RDBMS utilize the functionality of client server systems.

14.6 Failure and Recovery

The database may fail due to several reasons. And the failure may affect part of the database or the entire database. This results into a short-term or long-term loss to the valuable information. Some of the common types of failure that may affect the database system include:

- Hardware failures including memory errors, disk crashes, bad disk sectors, system crash etc.
- Software failures that include failure of the software that may the failure of system software or application software
- System crashes are due to hardware or software problems and the entire system may stop functioning altogether. In client server architecture, failure of a single client may affect only the system concerned whereas failure of the server may affect the processing of all the systems connected to the server.
- Network failures may affect the network resulting into loss of communication between the database and the systems connected to the network.
- Application software errors cause logical errors in the application software
- Natural disasters such as flood, fire, earthquake may affect the functioning of the entire system or even the entire network
- Deliberate attacks include failures due to intentional corruption of the data, software or hardware of the system. Such attacks may be made to intentionally cause damage to the system.
- Media failure such as damage to the primary or secondary storage may cause loss of data.

Some types of failures are easy to fix and take less time to recover. One of the important aspects of failure and recovery is maintenance of backup. Depending upon the type of failure, the type and frequency of backups may vary. Frequent backups may be required when database changes are frequent such as addition and deletion of tables, insertions and deletions of rows or columns in existing tables and frequent updates are done to the database.

Recovery algorithms are techniques to ensure transaction atomicity and durability despite failures. A recovery algorithm is used, that ensures atomicity by undoing the actions of transactions that do not commit and durability by making sure that all actions of committed transactions survive even if failures occur. Such an algorithm takes the database back to the most recent consistent state that was available before the failure. In order to restore the database to the consistent state, the current state may be reconstructed by applying the changes again or any changes made to the database may be reversed so that the state of the database is maintined. The system log or trail or journal is a useful tool that maintains the information about the changes that were made to the database. The basic unit of recovery of the database is a transaction. It should be ensured that either all the changes are made to the database or none are made. Two types of transaction recovery are as follows :

- Forward Recovery or REDO

- Backward Recovery or UNDO

Forward recovery or roll-forward is a technique in which the intermediate results of the transactions are stored in the buffers in the main memory. From the main memory, this data may be transferred to the secondary storage whenever required.

Two of the commonly used techniques for recovery techniques include log-based recovery and shadow paging.

The recovery techniques are as discussed below :

1. Deferred Update Techniques :

This technique defers the update to the database until the transaction commits. Till the commit, all the updates are stored in buffers or the local transaction workspace. These updates are recorded in the log and then written to the database. In case the transaction fails before commit, no UNDO is required as the changes have not been saved to the database. But in case of a crash, it may be essential to REDO the committed transactions from the log in order to record their effects on the database. This technique is also known as NO-UNDO/REDO algorithm.

2. Immediate Update Techniques :

Before committing, the database may be updated by some operations. In order to perform recovery, the changes must be written to the log. If a transaction fails before commit, it must be rolled back to the consistent state by performing UNDO on the database. Also it may be required to REDO some of the transactions. This technique is also known as UNDO/REDO algorithm. A variation of the algorithm requires only REDO and is known as UNDO/ NO-REDO algorithm.

Database recovery restores the database to the most consistent state that existed before the failure. The consistent state is achieved by either completing all the transactions or aborting the transactions that cannot be completed. Maintaining a database backup is essential for recovery of the database. This backup can be achieved on secondary storage devices such as CDs, tapes etc. Also, backup may be local or situated at a remote place. The level of backup required may also vary from one scenario to another. Full system backup may be required to maintain a copy of the entire system database. But due to the cost involved in the entire system backup, differential backup, differential backup may be done that maintains a copy of only some part of the database.

Forward Recovery (REDO) :

The database recovery is achieved by flushing the data to the database buffers. This data is further transferred from the buffers to the secondary storage. The update operation becomes permanent only when this data is transferred to the secondary storage. The failure may occur before writing the updates to the database or after the COMMIT operation. If the transaction has already committed, the redo operation writes the transaction updates to the database. This operation is called forward recovery or REDO or roll forward.

Backward Recovery (UNDO) :

This type of recovery occurs when an error occurs in between the normal operation of the database. Such recovery is called Backward Recovery, REDO or roll-backward. For instance backward recovery may be required on abnormal termination of the program or due to some wrongly entered value. Such a recovery is essential to maintain the atomicity of the transactions. The transaction is started from the current state and is traversed in the backward direction. During the traversal the changes made to the database are undone.

14.6.1 Shadow Paging :

An alternative to log-based recoveries is shadow paging. It was introduced in 1977 by Lorie. In this technique, the database is considered to be made up of logical units of storage of fixed size blocks or pages. The virtual or logical blocks are mapped onto the physical blocks of same size. This mapping is done with the help of a page table. To page tables are maintained, the current page table and shadow page table. During the initial stages both the page tables are the same and point to the same blocks of physical storage.

14.7 Summary

Database Administration is the function of manging and maintaining database management system (DBMS) software.

Client/Server Computing is the logical extension of modular programming. Modular programming has as its fundamental assumption that sepration of a large piece of software into its constituent parts creats the possibility for easier development and better maintainbility.

Distributed computing system consists of a number of processing elements, not necessarily homogeneous that are interconnected by a computer network, and that cooprate in performaing certain assigned task.

14.8 Self Assessment Questions

1. How client/server databases are different from distributed databases?
2. Explain the working of CASE tools with the help of example.
3. What do you understand by repositories explain?
4. Explain the different types of failures of databases with the help of example.
5. Discuss the concept of databases security by giving example of any practical database.

′ ′ ′

Unit - 15 : Emerging Trends in Database Management System - II

Structure of the Unit

15.0 Objective

At the end of this unit, you should be able to -

- Describe the concept of spatial and temporal databases
- Describe the concept of knowlege database
- Describe the data warehouse
- Describe the difference between database and data warehouse

15.1 Introduction

Databases function in many applications, spanning virtually the entire range of computer software. Databases have become the prefrred method of storage for large multi-user applications, where coordination between many users is needed. Even individual users find the them convenient, and many electronic mail programs and personal organizes are based on standard databases technology. Software database drivers are available for most database platforms so that applications software can use a common API to retrieve the information stored in a database.

15.2 Spatial Database

Modern applications are both data and computationally intensive and require storage and manipulation of voluminous traditional (alphanumeric) and nontraditional (images, text, geometric objects, etc.). Examples of such application domains are Geographical Information Systems (GIS), Multimedia Information Systems, CAD/CAM applications, Medical Information Systems. Spatial database management systems store data like points, lines, regions, volumes, and aim at supporting queries that involve the space characteristics of these data. In order to handle such queries, special techniques and tools enhance a spatial database system. These include new data types and models, sophisticated data structures and algorithms for eûcient query processing that diûer from their counterparts in a conservative alphanumeric database. When a spatial database is enhanced by temporal characteristics we get spatiotemporal database system. In such a system, the time of insertions, deletions, and updates is of great importance, since it must be able to store and manipulate the evolution of spatial objects. Spatial database systems are database systems for the management of spatial data.

Spatial data are point objects or spatially extended objects in a 2D or 3D space or in some high-dimensional vector space. Knowledge discovery becomes more and more important in spatial databases since increasingly large amounts of data obtained from satellite images, X-ray crystallography or other automatic equipment are stored in spatial databases.

In various ûelds there is a need to manage geometric, geographic, or spatial data, which means data related to space. The space of interest can be, for example, the 2D abstraction of (parts of) the surface of the earth that is, geographic space, the most prominent example like the layout of a VLSI design, a volume containing a model of the human brain, or another 3D space representing the arrangement of chains of protein molecules. At least since the advent of relational database systems there have been attempts to manage such data in database systems. Characteristic for the technology emerging to address these needs is the capability to deal with large collections of relatively simple geometric objects, for example, a set of 100,000 polygons. This is somewhat diûerent from areas like CAD databases (solid modeling, etc.) where geometric entities are composed hierarchically into complex structures, although the issues are certainly related.

A spatial database system is a database system which oûers SDTs in its data model and query language. It supports SDTs in its implementation, providing at least spatial indexing and eûcient algorithms for spatial join.

15.3 Temporal Database

Temporal database stores data relating to time instances. It offers temporal data types and stores information relating to past, present, and future time, for example, the history of the stock market or the movement of employees within an organization. Thus, a temporal database stores a collection of time related data.

15.3.1 Temporal Database Introduction : A temporal database is formed by compiling, storing temporal data. The difference between temporal data and non-temporal data is that a time-period is appended to data expressing when it was valid or stored in the database. The data stored by conventional databases consider data valid at present time as in the time instance "now." When data in such a database is modified, removed, or inserted, the state of the database is overwritten to form a new state. The state prior to any changes to the database is no longer available. Thus, by associate time with data, it is possible to store the different database states. In essence, temporal data is formed by time-stamping ordinary data (type of data we associate and store in conventional databases). In a relational data model, tuples are time-stamped and in an object-oriented data model, objects/attributes are time stamped. Each ordinary data has two time values attached to it, a start time, and an end time to establish the time interval of the data. In a relational data model, relations are extended to have two additional attributes, one for start time and another for end time.

15.3.2 Different Forms of Temporal Databases: Time can be interpreted as valid time (when data occurred or is true in reality) or transaction time (when data was entered into the database). A historical database stores data with respect to valid time. A rollback database stores data with respect to transaction time. A bitemporal database stores data with respect to both valid and transaction time – they store the history of data with respect to valid time and transaction time.

15.4 Knowledge Database

The concepts of Knowledge Base Management System (KBMS) and the Knowledge Warehouse (KW) are analogues of Database Management System (DBMS) and Data Warehouse. To arrive at a standard practice on the KBMS, and a standard definition of the Knowledge Warehouse, it's reasonable to begin with "straw man" definitions of both these concepts, next develop a general concept of This paper is a working paper, or "straw man," circulated for purposes of collaboration within the Knowledge Management Consortium International's (KMCI) Artificial Knowledge Management Systems Committee (AKMSC). It is intended that this paper be used by the Committee, along with contributions of other committee members to arrive at a collaborative.

Standard Recommended Practice on Artificial Knowledge Base Management Systems, a product of the Committee and the KMCI.2 what a standard practice might encompass, and then subject these products to vigorous criticism and analysis by the AKMSC. To produce this straw man is the purpose of this paper. I will proceed by considering some basic distinctions among data, information, and knowledge, then discuss DBMSs, the DW, DW evolution, and Data Warehousing as a process, and then move from there to develop the analogous concepts in the knowledge and knowledge management sphere.

15.5 Data Warehouse

A Data Warehouse (DW) is a database that stores information oriented to satisfy decision-making requests. It is a database with some particular features concerning the data it contains and its utilization. A very frequent problem in enterprises is the impossibility for accessing to corporate, complete, and integrated information of the enterprise that can satisfy decision-making requests. A paradox occurs: data exists but information cannot be obtained. In general, a DW is constructed with the goal of storing and providing all the relevant information that is generated along the diûerent databases of an enterprise. A data warehouse helps turn data into information. In today's business world, data warehouses are increasingly being used to make strategic business decisions.

15.5.1 Goals of Data Warehousing :

Data warehousing technology comprises a set of new concepts and tools, which support the knowledge worker like executive, manager, and analyst with information material for decision making. The fundamental reason for building a data warehouse is to improve the quality of information in the organization. The key issues are the provision of access to a company-wide view of data whenever it resides. Data coming from internal and external sources, existing in a variety of forms form traditional structural data to unstructured data like text ûles or multimedia is cleaned and integrated into a single repository. A data warehouse is the consistent store of this data which is made available to end users in a way they can understand and use in a business context. The need for data warehousing originated in the mid-to-late 1980s with the fundamental recognition that information systems must be distinguished into operational and informational systems. Operational systems support the day-to-day conduct of the business, and are optimized for fast response time of predeûned transactions, with a focus on update transactions. Operational data are a current and real-time representation of the business state. In contrast, informational systems are used to manage and control the business. They support the analysis of data for decision making about how the enterprise will operate now and in the future. They are designed mainly for ad hoc, complex, and mostly read-only queries over data obtained from a variety of sources. Information data are historical, i.e., they represent a stable view of the business over a period of time. Limitations of current technology to bring together information from many disparate systems hinder the development of informational systems. Data warehousing technology aims at providing a solution for these problems.

15.5.2 Characteristics of Data in Data Warehouse :

Data in the Data Warehouse is integrated from various, heterogeneous operational systems like database systems, ûat ûles, etc. Before the integration, structural and semantic diûerences have to be reconciled, i.e., data have to be "homogenized" according to a uniform data model. Furthermore, data values from operational systems have to be cleaned in order to get correct data into the data warehouse. Since a data warehouse is used for decision making, it is important that the data in the warehouse be correct. However, large volumes of data from multiple sources are involved; there is a high probability of errors and anomalies in the data. Therefore, tools that help to detect data anomalies and correct them can have a high payoû. Some examples where data cleaning becomes necessary are: inconsistent ûeld lengths, inconsistent descriptions, inconsistent value assignments, missing entries, and violation of integrity constraints.

The need to access historical data are one of the primary incentives for adopting the data warehouse approach. Historical data are necessary for business trend analysis which can be expressed in terms of understanding the diûerences between several views of the real-time data. Maintaining historical data means that periodical snapshots of the corresponding operational data are propagated and stored in the warehouse without overriding previous warehouse states. However, the potential volume of historical data and the associated storage costs must always be considered in relation to their business beneûts.

Data warehouse contains usually additional data, not explicitly stored in the operational sources, but derived through some process from operational data. For example, operational sales data could be stored in several aggregation levels in the warehouse.

15.5.3 Data Warehouse Architectures :

Data warehouses and their architectures vary depending upon the speciûcs of an organization's situation. Three common data warehouse architectures which are discussed in this section are:

Basic Data Warehouse Architecture

(a) Data Warehouse Architecture with a Staging Area

(b) Data Warehouse Architecture with a Staging Area and Data Marts

(c) Basic Data Warehouse Architecture

Data warehouses contain consolidated data from many sources, augmented with summary information and covering a long time period. Warehouses are much larger than other kinds of databases; sizes ranging from several gigabytes to terabytes are common. Typical workloads involve ad hoc, fairly complex queries and fast response times are important. These characteristics differentiate warehouse applications from OLTP applications, and different DBMS design and implementation techniques must be used to achieve satisfactory results. A distributed DBMS with good scalability and high availability (achieved by storing tables redundantly at more than one site) is required for very large warehouses.

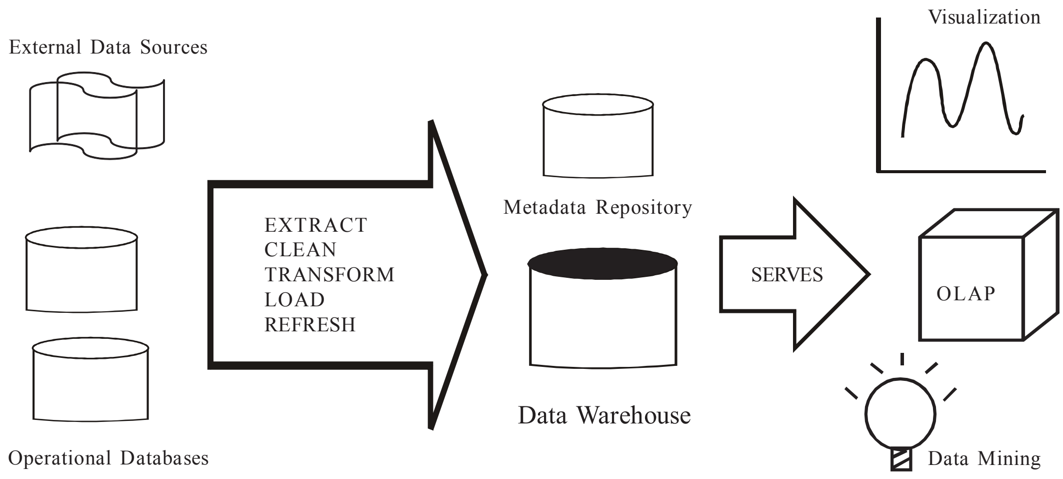


Figure 1. A Typical Data Warehousing Architecture

A typical data warehousing architecture is illustrated in Figure 1. An organization's daily operations access and modify operational databases. Data from these operational databases and other external sources (e.g., customer profiles supplied by external consultants) are extracted by using gateways, or standard external interfaces supported by the underlying DBMSs. A gateway is an application program interface that allows client programs to generate SQL statements to be executed at a server. Standards such as Open Database Connectivity (ODBC) and Open Linking and Embedding for Databases (OLE-DB) from Microsoft and Java Database Connectivity (JDBC) are emerging for gateways.

15.5.3.1 Data Mart

Data marts are complete logical subsets of the complete data warehouse. Data marts should be consistent in their data representation in order to assure Data Warehouse robustness. A data mart is a set of tables that focus on a single task. This may be for a department, such as production or maintenance department, or a single task such as handling customer products.

15.5.3.2 Meta Data

In general metadata are deûned as "data about data" or "data describing the meaning of data." In data warehousing, there are various types of metadata. For example information about the operational sources, the structure and semantics of the data warehouse data, the tasks performed during the construction, maintenance and access of a data ware house, etc. A data warehouse without adequate metadata are like "a ûling cabinet stuûed with papers, but without any folders or labels." The quality of metadata and the resulting quality of information gained using a data warehouse solution are tightly linked. In a data warehouse metadata are categorized into Business and Technical metadata. Business metadata describes what is in the ware house, its meaning in business terms. The business metadata lies above technical metadata, adding some more details to the extracted material. This type of metadata are important as it facilitates business users and increases the accessibility. In contrast, technical metadata describes the data elements as they exist in the ware house. This type of metadata are used for data modeling initially, and once the warehouse is erected this metadata are frequently used by warehouse administrator and software tools.

Implementing a concrete Data Warehouse architecture is a complex task comprising of two major phases. In the conûguration phase, a conceptual view of the ware house is ûrst speciûed according to user requirements which are often termed as data warehouse design. Then the involved data sources and the way data will be extracted and loaded into the warehouse is determined. Finally, decisions about persistent storage of the warehouse using database technology and the various ways data will be accessed during analysis are made.

15.5.4 Data Warehouse Design :

Data warehouse design methods consider the read-oriented character of warehouse data and enables the eûcient query processing over huge amounts of data. The core requirements and principles that guide the design of data warehouses are summarized later:

Data Warehouses Should be Organized Around Subject Areas :

Subject areas are similar to the concept of functional areas like sales, project management, employees, etc. Each subject areas are associated with a conceptual schema and these can be represented using one or more entities in the ER data model or by one or more object classes in the object oriented data model. For example: In company database the relations like employee, sales, and project management are represented as entities in ER data model or object classes in object oriented data model.

Data Warehouses Should have some Integration Capability :

A common database should be designed and used so that all the diûerent individual representations can be mapped to it. This is particularly useful if the warehouse is implemented as multidatabase or federated database.

Data should be Nonvolatile and Mass Loaded :

Data in Data Warehouses should be nonvolatile. For this data extraction from current database to DW requires that a decision should be made whether to extract the data using standard relational database techniques at the row or column level or specialized techniques for mass extraction. Data cleaning techniques are required to maintain data quality, similarly data migration, data scrubbing, and data auditing. Refresh techniques propagate updates on the source data to base data and derived data in the DW. The decision of when and how to refresh is made by the DW administrator and depends on user needs (e.g., OLAP needs) and existing traûc to the DW.

Data Tends to Exist at Multiple Levels of Dimensions :

Data can be deûned not only by time frame but also by geographic region; type of product manufactured or sold type of store and so on. The complete size of the databases is a major problem in the design and implementation of data warehouses, especially for certain queries and updates and sequential backups. This decides whether to select relational databases or multidimensional database for the implementation of a data warehouse.

Data Warehouse Should be Flexible Enough to Meet Changing Requirements Rapidly :

Insertion, updating, and retrieval of data should be very eûcient and ûexible to achieve good and eûcient decision.

Data Warehouse Should have a Capability for Rewriting History, that is, Allowing for "what-if" Analysis :

Data Warehouse should allow the administrator to update historical data temporarily for the purpose of "what-if" analysis. Once the analysis is completed, the data must be correctly rolled back. This assumes that the data must be at the proper dimension in the ûrst place.

Good DW User Interface Should be Selected :

The interface should be very user friendly for eûcient use of DW. The leading choices of today are SQL.

Data Should be Either Centralized or Distributed Physically :

The DWshould have the capability to handle distributed data over a network. This requirement will become more critical as the use of DWs grows and sources of data expand.

15.5.5 Classification of Data Warehouse Design :

The data warehouse design can be broadly classiûed into two categories

(1) Logical design and (2) Physical design.

15.5.5.1 Logical Design :

The logical design is more conceptual and abstract than physical design. In the logical design, the emphasis is on the logical relationship among the objects. One technique that can be used to model organization's logical information requirements is entity-relationship modeling. Entity-relationship modeling involves identifying the things of importance (entities), the properties of these things (attributes), and how they are related to one another (relationships). The process of logical design involves arranging data into a series of logical relationships called entities and attributes. An entity represents a chunk of information. In relational databases, an entity often maps to table. An attribute is a component of an entity that helps deûne the uniqueness of the entity. In relational databases, an attribute maps to a column. Whereas entity-relationship diagramming has traditionally been associated with highly normalized models such as OLTP applications, the technique is still useful for data warehouse design in the form of dimensional modeling.

In dimensional modeling, instead of seeking to discover atomic units of information and all the relationship between them, the focus is on identifying which information belongs to a central fact table and which information belongs to its associated dimension tables. In a nutshell, the logical design should result in (a) a set of entities and attributes corresponding to fact tables and dimension tables and (b) a model of operational data from the source into subject-oriented information in target data warehouse schema. Some of the logical warehouse design tools from Oracle are Oracle Warehouse Builder, Oracle Designer, which is a general purpose modeling tool.

Data Warehousing Schemas :

A schema is a collection of database objects, including tables, views, indexes, and synonyms. The arrangement of schema objects in the schema models

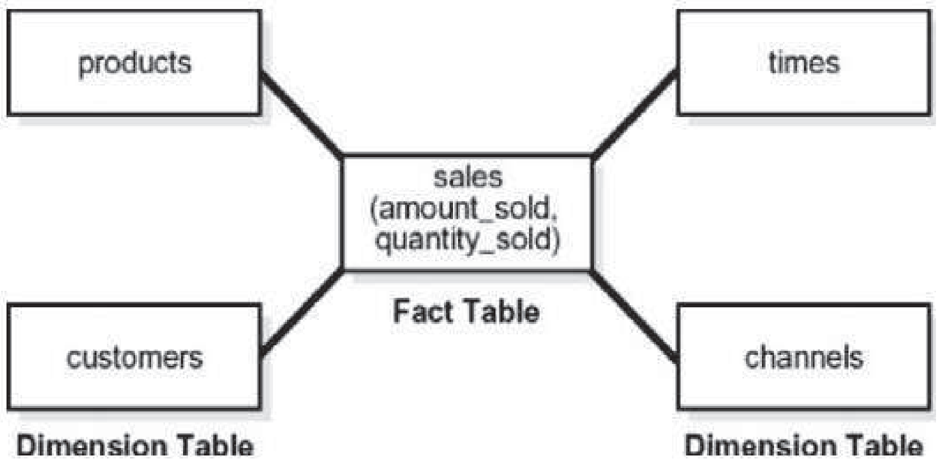


Figure 2. Star Schema

designed for data ware house can be done in a variety of ways. Most data warehouses use a dimensional model.

Star Schema :

The star schema is the simplest data warehouse schema. It is called a star schema because the diagram resembles a star, with points radiating from the center. The center of the star consists of one or more fact tables and the points of the star are the dimension tables as illustrated in Figure2. A star schema optimizes performance by keeping queries simple and providing fast response time. All the information about each level is stored in one row. The most natural way to model a data warehouse is star schema, only one join establishes the relationship between the fact table and any one of the dimension tables.

Another schema that is sometimes useful is the snowûake schema, which is a star schema with normalized dimensions in a tree structure.

Data Warehouse Objects :

Fact and dimension tables are the two types of objects commonly used in the dimensional data warehouse schemas. Fact tables are the large tables in warehouse schema that store business measurements. Fact tables typically contain facts and foreign keys to the dimension tables. Fact tables represent data, usually numeric and additive that can be analyzed and examined. Dimension tables, also known as lookup or reference tables; contain the relatively static data in the warehouse. Dimension tables stores the information that is normally used to contain queries. Dimension tables are usually textual and descriptive.

Fact Tables :

A fact table typically has two types of columns: those that contain numeric facts and those that are foreign keys to dimension tables. A fact table contains either detail-level facts or facts that have been aggregated. Fact tables that contain aggregated facts are often called summary tables. A fact table usually contains facts with the same level of aggregation. Though most facts are additive, they can also be semiadditive or nonadditive. Additive facts can be aggregated by simple arithmetical addition. Semiadditive facts can be aggregated along some of the dimensions and not along others.

Dimension Tables :

A dimension is a structure, often composed of one or more hierarchies, that categorizes data. Dimensional attributes help to describe the dimensional value. They are commonly descriptive, textual values. Several distinct dimensions, combined with facts, enable one to answer business questions. Dimensional data are typically collected at the lowest level of detail and then aggregated into higher level totals that are more useful for analysts. These natural rollups or aggregations within a dimension table are called hierarchies.

Hierarchies :

Hierarchies are logical structures that use ordered levels as a means of organizing data. A hierarchy can be used to deûne data aggregation. For example, in a time dimension, a hierarchy might aggregate data from the month level to the quarter level to the year level. A hierarchy can also be used to deûne a navigational drill path and to establish a family structure. Within a hierarchy, each level is logically connected to the levels above and below it. Data values at lower levels aggregate into the data values at higher levels. A dimension can be composed of more than one hierarchy.

Hierarchies impose a family structure on dimension values. For a particular level value, a value at the next higher level is its parent, and values at the next lower level are its children. These familial relationships enable analysts to access data quickly.

15.5.5.2 Physical Design

During the physical design process the data gathered during the logical design phase is converted into a description of the physical database structure. Physical design decisions are mainly driven by query performance and database maintenance aspects. Figure3. oûers a graphical way of looking at the diûerent ways of logical and physical designs.

Physical Design Structures :

Some of the physical design structures that are going to be discussed in this section include (a) Table spaces (b) Tables and Partitioned Tables (c) Views (d) Integrity Constraints, and (e) Dimensions

Table Spaces :

A table space consists of one or more data ûles, which are physical structures within the operating system. A data ûle is associated with only one table space. From the design perspective, table spaces are containers for physical design structures. Table spaces need to be separated by diûerences. For example, tables should be separated from their indexes and small tables should be separated from large tables. Table spaces should also represent logical business units.

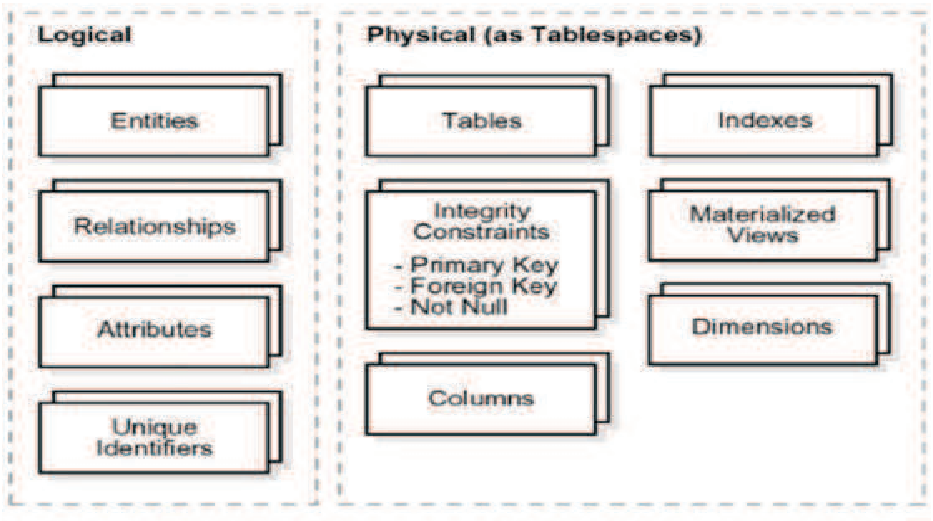


Figure 3. Logical and Physical design of data warehouse design

Tables and Partitioned Tables :

Tables are the basic unit of data storage. They are the container for the expected amount of raw data in the data warehouse. Using partitioned tables instead of nonpartitioned ones addresses the key problem of supporting very large data volumes by allowing you to decompose them into smaller and more manageable pieces. The main design criterion for partitioning is manageability.

Data Segment Compression :

Disk space can be saved by compressing heap-organized tables. A typical type of heap-organized table that one should consider for data segment compression is partitioned tables. Data segment compression can also speed up query execution. There is, however, a cost in CPU overhead. Data segment compression should be used with highly redundant data, such as tables with many foreign keys.

Views :

A view is a tailored presentation of the data contained in one or more tables or other views. A view takes the output of a query and treats it as a table. Views do not require any space in the database.

Integrity Constraints :

Integrity constraints are used to enforce business rules associated with the database and to prevent having invalid information in the tables. Integrity constraints in data warehousing diûer from constraints in OLTP environments. In OLTP environments, they primarily prevent the insertion of invalid data into a record, which is not a big problem in data warehousing environments because accuracy has already been guaranteed. In data warehousing environments, constraints are only used for query rewrite. NOT NULL constraints are particularly common in data warehouses. Under some speciûc circumstances, constraints need space in the database. These constraints are in the form of the underlying unique index.

Indexes and Partitioned Indexes :

Indexes are optional structures associated with tables or clusters. In addition to the classical B-tree indexes, bitmap indexes are very common in data warehousing environments. Bitmap indexes are optimized index structures for set-oriented operations. Additionally, they are necessary for some optimized data access methods such as star transformations.

Dimensions :

A dimension is a schema object that deûnes hierarchical relationships between columns or column sets. A hierarchical relationship is a functional dependency from one level of a hierarchy to the next one. A dimension is a container of logical relationships and does not require any space in the database. A typical dimension is city, state (or province), region, and country.

15.5.6 The User Interface

In this section, we provide a brief introduction to contemporary interfaces for data warehouses. A variety of tools are available to query and analyze data stored in data warehouses. These tools can be classiûed as follows:

a) Traditional query and reporting tools

b) On-line analytical processing, MOLAP, and ROLAP tools

c) Data-mining tools

d) Data-visualization tools

15.5.6.1 Traditional Query and Reporting Tools

Traditional query and reporting tools include spreadsheets, personal computer databases, and report writers and generators.

15.5.6.2 Olap Tools

On-Line Analytical Processing is the use of a set of graphical tools that provides users with multidimensional views of their data and allows them to analyze the data using simple windowing techniques. The term on-line analytical processing is intended to contrast with the more traditional term on-line transaction processing. OLAP is a general term for several categories of data warehouse and data mart access tools. Relational OLAP (ROLAP) tools use variations of SQL and view the database as a traditional relational database, in either a star schema or other normalized or denormalized set of tables. ROLAP tools access the data warehouse or data mart directly. Multidimensional OLAP (MOLAP) loads data into an intermediate structure usually a three or higher dimensional array.

15.5.6.3 Data-Mining Tools

Data mining is knowledge discovery using a sophisticated blend of techniques from traditional statistics, artiûcial intelligence, and computer graphics. As the amount of data in data warehouses is growing exponentially, the users require automated techniques provided by data-mining tools to mine the knowledge in these data.

15.5.6.4 Data Visualization Tools

Data-visualization is the representation of data in graphical and multimedia formats for human analysis. Beneûts of data visualization include the ability to better observe trends and patterns, and to identify correlations and clusters.

15.6 Difference Between Database and Data Warehouse

A Computer Database is a structured collection of records or data that is stored in a computer system. The structure is achieved by organizing the data according to a database model. The model in most common use today is the relational model. Other models such as the hierarchical model and the network model use a more explicit representation of relationships (see below for explanation of the various database models). A computer database relies upon software to organize the storage of data. This software is known as a database management system (DBMS). best examples are Mysql, Oracle etc.

A data warehouse is a repository of an organization's electronically stored data. Data warehouses are designed to facilitate reporting and analysis. However, the means to retrieve and analyze data, to extract, transform and load data, and to manage the data dictionary are also considered essential components of a data warehousing system. Many references to data warehousing use this broader context. Thus, an expanded definition for data warehousing includes business intelligence tools, tools to extract, transform, and load data into the repository, and tools to manage and retrieve meta data.

Database	**Data Warehouse**
Used for Online Transactional Processing (OLTP) but can be used for other purposes such as Data Warehousing. This records the data from the user for history.	Used for Online Analytical Processing (OLAP). This reads the historical data for the Users for business decisions.
The tables and joins are complex since they are normalized (for RDMS). This is done to reduce redundant data and to save storage space	The Tables and joins are simple since they are de-normalized. This is done to reduce the response time for analytical queries
Entity - Relational modeling techniques are used for RDMS database design	Data-Modeling techniques are used for the Data Warehouse design
Optimized for write operation	Optimized for read operations
Performace is low for analysis queries	High performance for analytical queries, is usually a Database

It's important to note as well that Data Warehouse could be sourced zero to many databases.

15.7 Summary

Developes and administrators define the database in the form of tables. They then create forms and reports on the application server. Users run the application and enter data or make choices.

15.8 Self Assessment Questions

1. What do you understand by spatial database? Explain.
2. What is data warehouse? Explain.
3. Explain the classification of Data warehouse Design.
4. What do you understand by user interface? Explain.