

UNIT – 1 <> CONTENTS

(CHAPTER 1, 2 & 3)

Data vs Information	C H A P T E R I
Introducing the Database and DBMS	
Role and advantages of DBMS	
Types of databases	
Why Database design is important	
Historical Roots : Files and File Systems	
Problems with File System Data Management	
Database systems	
Database systems environment	
DBMS Functions	
<hr/>	
Importance of Data Models	C H A P T E R R
Business Rules	
Evaluation of Data Models	
Hierarchical, Network, Relational, ER, OO models	
<hr/>	
Logical view of Data	C H A P T E R R
Keys	
Integrity Rules	
Relational set operators	
Indexes	
Codd's Relational Database Rules	
	III

UNIT – I

INTRODUCTION TO DATA BASE SYSTEMS AND FUNDAMENTALS

BASIC CONCEPTS AND DEFENITIONS:

- ❖ DATA
- ❖ INFORMATION

Data:

Def. 1:

Data referred to known facts that could be recorded and stored on computer media.
Ex: In a student database, the data would include facts such as student name, age, address, and marks.

Def. 2:

Data are raw facts. The word raw indicates that the facts have not yet been processed to reveal their meaning.

Databases today are used to store objects such as documents, photographic images, sound, and even video segments. So the following is broadened definition.

Def. 3:

"Data consist of facts, text, graphics, images, sound and video segments that have meaning in the users' environment".

Information:

The terms data and information are closely related, and in fact are often used interchangeably.

Def:

" Data that have been processed in such a way as to increase the knowledge of the person who uses the data".

To convert data into information is to summarize them or otherwise process and present them for human interpretation.

Steps for transferring raw data into information:

- A) Initial Survey Screen
- B) Raw Data
- C) Information in Summary Format
- D) Information in Graphic Format

Today databases may contain either data or information (or both).

In this “information age” the information must maintain the quality. By **quality information** we mean information that is

1. Accuracy
2. Timeliness
3. Relevancy

1. **Accuracy:** It means that the information is free from mistakes and errors. Accuracy means more than ‘one plus one equals two’.
2. **Timeliness:** It means that the recipients can get the information when they need it. The information should maintain time frame.
3. **Relevancy:** Relevancy means the use of a piece of information for a particular person.

Some key points for information:

- ❖ Data constitute the building blocks of information
- ❖ Information is produced by processing data.
- ❖ Information is used to reveal the meaning of data.
- ❖ Accurate, relevant, and timely information is the key to good decision making.
- ❖ Good decision making is the key to organizational survival in a global environment.

INTRODUCING THE DATABASE AND THE DBMS:

Database:

Def:

“Database is defined as an organized collection of related data”.

By *organized* we mean that the data are structured so as to be easily stored, manipulated, and retrieved by users.

Metadata:

Def. 1:

“Metadata are data that describe the properties or characteristics of other data”.

Some of these properties include data definitions, data structures and rules or constraints.

Metadata show the data item name, the data type, length, minimum and maximum allowable values and brief description of each data item.

Ex: some sample metadata for the Class Roster are listed in the following table.

<i>Data item</i>		<i>value</i>			
<i>Name</i>	<i>type</i>	<i>length</i>	<i>min</i>	<i>max</i>	<i>description</i>
Course	Alphanumeric	30			Course ID and name
Section	Integer	1	1	9	Section number
Semester	Alphanumeric	10			semester and year
Name	Alphanumeric	30			Student name
ID	integer	9			Student ID

Metadata allow database designers and users to understand what data exist, what the data mean and what the fine distinctions are between seemingly similar data items.

Def. 2:

The database contains the data you have collected and “data about data” known as Metadata.

Database Management System:

Def. 1:

“A Database Management system (DBMS) is a collection of programs(software)that allow users to store and access the data and work a database for easy manipulation of data”.

Eg: dbase, oracle

Def. 2:

A database management system (DBMS) is a collection of programs that manages the database structure and controls access to the data stored in the database.

Def. 3:

To implement a database and to manage its contents, you need a database management system. The DBMS serves as the intermediary between the user and the database.

ROLE AND ADVANTAGES OF THE DBMS:

The DBMS serves as the intermediary between the user and the database. Having a DBMS between the end user's applications and the database offers some important advantages.

First, the DBMS enables the data in the database to be shared among multiple applications and users.

Second, the DBMS integrates the many different users' views of the data into a single all-encompassing data repository. The DBMS helps make data management more efficient and effective. In particular, a DBMS provides advantages such as:

1. Program-Data Independence
2. Minimal Data Redundancy
3. Improved Data Consistency
4. Improved Data Sharing
5. Improved Data Quality
6. Increased Productivity of Application Development
7. Improved Data Accessibility and Responsiveness
8. Enforcement Standards
9. Reduced Program Maintenance

1. Program-Data Independence:

The separation of data descriptions from the application programs that use the data is called data independence. With the database approach, data descriptions are stored in a central location called the *repository*. This property of database systems allows an organization's data to change and evolve (within limits) without changing the application programs that process the data.

2. Minimal Data Redundancy:

The design goal with the database approach is that previously separate data files are integrated into a single, logical structure. Each primary fact is recorded in only one place in the database.

3. Improved Data Consistency:

By eliminating (or controlling) data redundancy, we greatly reduce the opportunities for inconsistency.

Ex: If a customer address is stored only once, we cannot disagree on the stored values. Also, updating data values is greatly simplified when each value is stored in one place only. Finally, we avoid the wasted storage space.

4. Improved Data Sharing:

A database is designed as a shared corporate resource. Authorized internal and external users are granted permission to use the database, and each user is provided one or more user views to facilitate this use.

5. Improved Data Quality:

The database approach provides a number of tools and processes to improve data quality.

1. Database designers can specify integrity constraints that are enforced by the DBMS.

2. One of the objectives of a data warehouse environment is to clean up operational data before they are placed in the data warehouse.

6. Increased Productivity of Application Development:

A major advantage of the database approach is that it greatly reduces the cost and time for developing new business applications.

7. Improved Data Accessibility and Responsiveness:

With a relational database, end users without programming experience can often retrieve and display data.

Ex: `select * from student where student_id=999;`

8. Enforcement Standards:

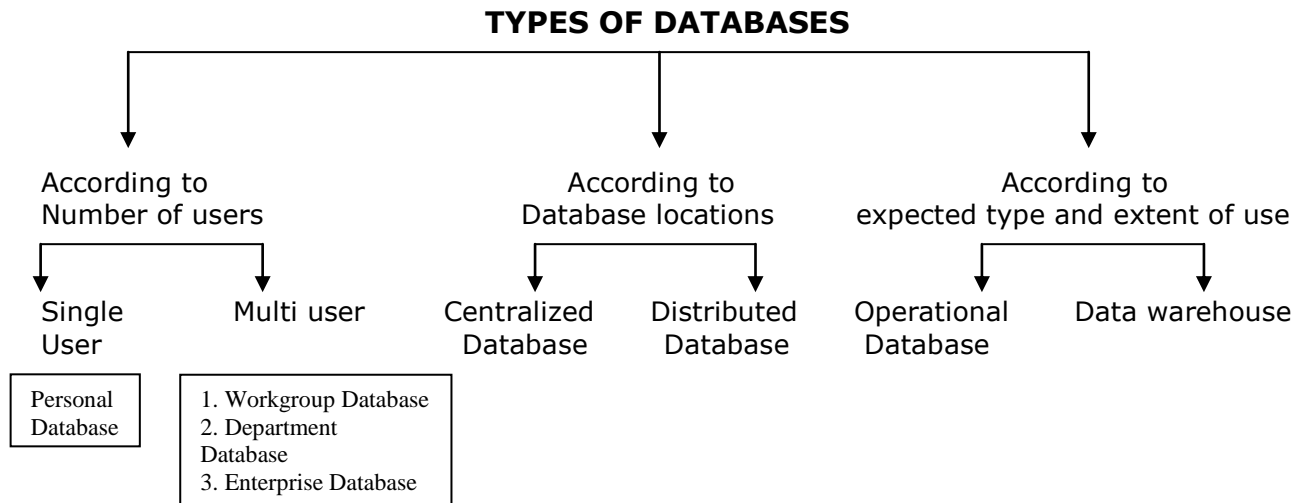
When the database approach is implemented with full management support, the database administration function should be granted single-point authority and responsibility for establishing and enforcing data standards. These standards will include naming conventions, data quality standards, and uniform procedures for accessing, updating, and protecting data.

9. Reduced Program Maintenance:

From the above all factors, as a result program maintenance can be significantly reduced in a modern database environment.

TYPES OF DATABASES:

A DBMS can support many different types of databases. Database can be classified according to number of users, the database locations and the expected type and extent of use.



I. ACCORDING TO NUMBER OF USERS

1. Personal database

“Personal databases are designed to support one user”. Personal databases have long resided on personal computers (PCs), including laptops.

Simple database applications that store customer information and the details of contacts with each customer can be used from a PC and easily transferred from one device to the other for backup and work purposes.

Personal databases are widely used because they can often improve personal productivity. However, they entail a risk: The data cannot easily be shared with other users.

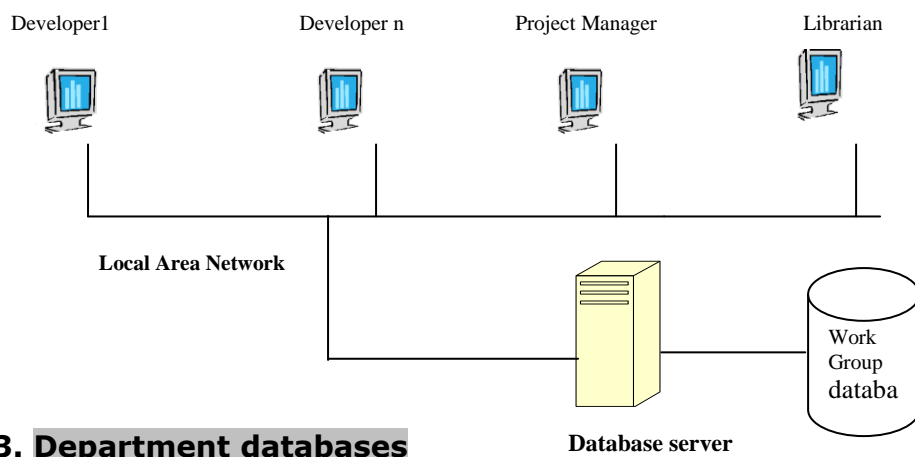
2. Work group databases

“A workgroup is a relatively small team of people who collaborate on the same project or application or on a group of similar projects or applications”.

A work group typically comprises fewer than **25 persons**. These persons might be engaged with a construction project or with developing a new computer application. A workgroup database is designed to support the collaborative efforts of such a team.

The group needs a database that will track each item as it is developed and allow the data to be easily shared by the team members.

The method of sharing the data in this database is shown in the following figure. Each member of the workgroup has a desktop computer and the computers are linked by means of a local area network (LAN). The database is stored on a central device called the *database server*, which is also connected to the network. Thus each member of the workgroup has access to the shared data. Different types of group members may have different user views of this shared database.



3. Department databases

A department is a functional unit within an organization. Typical examples of departments are personnel, marketing, manufacturing and accounting.

"A department is generally larger than a workgroup (typically between 25 and 100 persons) and is responsible for a more diverse range of functions".

"Department databases are designed to support the various and activities of a department".

4. Enterprise databases

When the database is used by the entire organization and supports many users (more than 50 usually hundreds) across many departments, the database is known as an enterprise database.

Type of Database	No. of Users	Typical architecture	Typical size of database
Personal	1	Desktop/laptop computer	Megabytes
Workgroup	5-25	Client/server (two-tier)	Megabytes-Gigabytes
Department	25-100	Client/server (three-tier)	Gigabytes
Enterprise	>100	Client/server(Distributed or parallel server)	Gigabytes-Terabytes

II According to database locations

1. Centralized database

Location might also be used to classify the database. For example, a *database that supports data located at a single site is called a centralized database.*

2. Distributed database

A database that supports data distributed across several different sites is called a distributed database.

III According to expected type and extent of use

1. Operational database

A database that is designed primarily to *support a company's day-to-day operations is classified as an operational database* (sometimes referred to as a *transactional or production database*).

2. Data warehouse

A data warehouse focuses primarily on storing data used to generate information required to make tactical or strategic decisions. Such decisions typically require extensive "data massaging" (data manipulation) to extract information to formulate pricing decisions, sales forecasts, market positioning, etc. Most decision support data are based on historical data obtained from operational databases.

Additionally, the data warehouse can store data derived from many sources. To make it easier to retrieve such data, the data warehouse structure is quite different from that of a transaction oriented database.

WHY DATABASE DESIGN IS IMPORTANT:

A good database - that is, a database that meets all user requirements - does not just happen; its structure must be designed carefully. Even a good DBMS will perform poorly with a badly designed database.

Database design defines the database structure. A well-designed database facilitates data management and generates accurate and valuable information. A poorly designed database can lead to bad decision making and bad decision making can lead to the failure of an organization.

Proper database design requires the database designer to identify precisely the database's expected use. Designing a transactional database emphasizes accurate and consistent data and operational speed. The design of a data warehouse database recognizes the use of historical and aggregated data. Designing a database to be used in a centralized, single-user environment requires a different approach from that used in the design of a distributed, multi-user database.

HISTORICAL ROOTS: FILES AND FILE SYSTEMS

In the recent past, a manager of almost any small organization was (and sometimes still is) able to keep track of necessary data by using a manual file system. Such a file system was traditionally composed of a collection of file folders each properly tagged and kept in a filing cabinet. Organization of the data within the file folders was determined by the data's expected use ideally.

As long as a data collection was relatively small and an organization's managers had few reporting requirements, the manual system served its role well as a data repository. However, as organizations grew and as reporting requirements became more complex, keeping track of data in a manual file system became more difficult. In fact, finding and using data in growing collections of file folders turned into such a time-consuming and cumbersome task that it became unlikely that such data could generate useful information.

Unfortunately, report generation from a manual file system can be slow and cumbersome. Initially, the computer files within the file system were similar to the manual files. A simple example of a customer data file or a small insurance company.

BASIC FILE TERMINOLOGY:

Data:

"Raw" facts, such as a telephone number, a birth date, a customer name, and a year-to-date (YTD) sales value. Data have little meaning unless they have been organized in some logical manner. The smallest piece of data that can be "recognized" by the computer is a single character, such as the letter A, the number 5, or a symbol such as /. A single character requires 1 byte to computer storage.

Field:

A character or group of characters (alphabetic or numeric) that has specific meaning. A field is used to define and store data.

Record:

A logically connected set of one or more fields that describes a person, place, or thing.

File:

A collection of related records. For example, a file might contain data about vendors of ROBCOR company, or a file might contain the records for the students currently enrolled at S.V. University.

Each file in the system used its own application programs to store, retrieve, and modify data. And each file was owned by the individual or the department that commissioned its creation.

As the file system grew, the demand for the DP specialist's programming skills grew even faster, and the DP specialist was authorized to hire additional programmers. The size of the file system also required a larger, more complex computer. The new computer and the additional programming staff caused the DP specialist to spend less time programming and more time managing technical and human resources. Therefore, the DP specialist's job evolved into that of a **data processing (DP) manager**, who supervised a DP department. In spite of these organizational changes, however, the DP department's primary activity remained programming, and the DP manager inevitably spent much time as a supervising senior programmer and program troubleshooter.

PROBLEMS WITH FILE SYSTEM DATA MANAGEMENT:

The file system method of organizing and managing data was a definite improvement over a manual system, and the file system served a useful purpose in data management for over two decades, a very long time span in the computer era.

THE LIMITATIONS OF FILE SYSTEM DATA MANAGEMENT are:

1) It requires extensive programming. There are no ad hoc query capabilities. System administration can be complex and difficult.

Another problem related to the need for extensive programming is that as the number of files in the system expands, system administration becomes more difficult. Each file must have its own file management system composed of programs that allow the user to:

Add, delete, and modify file data.

List the file contents and generate reports.

Making changes in an existing structure can be difficult in a file system environment. For example, changing just one field in the original CUSTOMER file requires a program that:

1. Opens the original file.
2. Reads a record from the original file.
3. Transforms the original data to conform to the new structure's storage requirements.
4. Writes the transformed data into the new file structure.
5. Deletes the original file.

Even a simple file system of only 20 files requires $5 \times 20 = 100$ file management programs. If each of the files is accessed by 10 different reporting programs, an additional $20 \times 10 = 200$ programs must be written. Because ad hoc queries are not possible, the file reporting programs can multiply quickly. And because each department in the organization "owns" its data by creating its own files, the number of files can multiply rapidly.

2) It is difficult to make changes to existing structures

Any file structure change, no matter how minor, forces modifications in all of the programs that use the data in that file. *Modifications are likely to produce errors (bugs), and additional time can be spent using a debugging process to find those errors.*

3) Security features are likely to be inadequate

Another fault is that security features such as effective password protection, the ability to lock out parts of files or parts of the system itself, and other measures designed to safeguard data confidentiality are difficult to program and are, therefore, often omitted in a file system environment. Even when an attempt is made to improve system and data security, the security devices tend to be limited in scope and effectiveness.

Those limitations, in turn, lead to problems of structural and data dependency and data redundancy.

4) Data Dependency

Any addition of data fields or any change in the data field that are defining the file structure, will make it mandatory to change the maintenance program accordingly.

Ex: Suppose the customer master file is used in both the order filling system and the invoicing system. Suppose it is decided to change the customer address field in the records in these files 30 to 40 characters. The file descriptions in each program that is affected (up to five programs) would have to be modified. It is often difficult even to locate all programs affected by such changes.

5) Data redundancy

Repetition of Data is known as data redundancy

The organizational structure promotes the storage of the same basic data in different locations. For example: the agent names and phone numbers occur in both the CUSTOMER and the AGENT files. You need only one correct copy of the agent names and phone numbers. Having them occur in more than one place produces data redundancy. *Data redundancy exists when the same data are stored unnecessarily at different places.*

The duplication is wasteful since it requires additional storage space and increased effort to keep all files up to date. Unfortunately, duplicate data files often result in loss of data integrity.

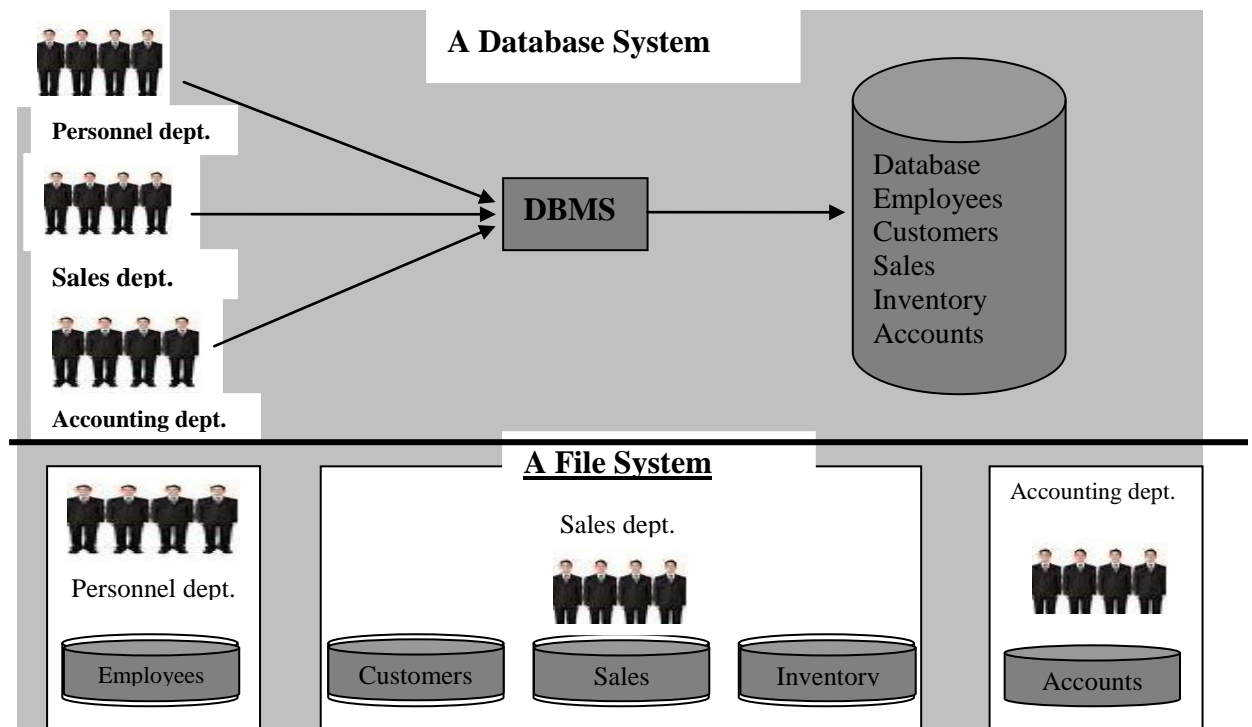
Uncontrolled data redundancy sets the stage for

6) Data inconsistency

Data inconsistency exists when different and conflicting versions of the same data appear in different places. For example, suppose you change an agent's phone number or address in the agent file. If you forget to make corresponding changes in the customer file, the files contain different data for the same agent. Reports will yield inconsistent results depending on which version of the data is used.

7) Data Anomalies

Any change in any field value must be correctly made in many places to maintain data integrity. *A Data Anomaly develops when all of the required changes in the redundant data are not made successfully.* The data anomalies found commonly defined as follows: Update anomalies, Insertion anomalies, Deletion anomalies



Contrasting Database and File Systems

THE DATABASE SYSTEM ENVIRONMENT TYPES:

The term **database system** refers to an organization of components that define and regulate the collection, storage, management, and use of data within a database environment. From a general management point of view, the database system is composed of the five major parts shown in figure below

1. Hardware
2. Software
3. People
4. Procedures
5. Data

1. Hardware:

Hardware refers to all of the system's physical devices; for example, computers (microcomputers, mainframes, workstations, and servers), storage devices, printers, network devices (hubs, switches, routers, fiber optics), and other devices (automated teller machines, ID readers, etc.).

2. Software:

Although the most readily identified software is the DBMS itself, to make the database system function fully, *three types of software are needed:*

- ❖ *Operating system software*
- ❖ *DBMS software, and*
- ❖ *Application programs and utilities.*
 - ❖ *Operating system software manages all hardware components and makes it possible for all other software to run on the computers. Examples of operating system software include Microsoft Windows, Linux, Mac OS, UNIX, and MVS.*
 - ❖ *DBMS software manages the database within the database system. Some examples of DBMS software include Microsoft Access and SQL Server, Oracle Corporation's Oracle, and IBM's DB2.*
 - ❖ *Application programs and utility software are used to access and manipulate data in the DBMS and to manage the computer environment in which data access and manipulation take place. Application programs are most commonly used to access data found within the database to generate reports, tabulations, and other information to facilitate decision making. Utilities are the software tools used to help manage the database system's computer components.*

3. People:

This component includes all users of the database system. On the basis of primary job functions, *five types* of users can be identified in a database system: *systems administrators, database administrators, database designers, systems analysts and programmers, and end users.*

- ❖ *Systems administrators* oversee the database system's general operations.
- ❖ *Database administrators*, also known as DBAs, *manage the DBMS* and ensure that the database is functioning properly.
- ❖ *Database designers design the database structure.* They are, in effect, the database architects. If the database design is poor, even the best application programmers and the most dedicated DBAs cannot produce a useful database environment. Because organizations strive to optimize their data resources. The database designer's job description has expanded to cover new dimensions and growing responsibilities.

- ❖ *Systems analysts and programmers design and implement the application programs.* They design and create the data entry screens, reports, and procedures through which end users access and manipulate the database's data.
- ❖ *End users are the people who use the application programs* to run the organization's daily operations. For example, salesclerks, supervisors, managers, and directors are all classified as end users.

4. Procedures:

Procedures are the instructions and rules that govern the design and use of the database system. Procedures play an important role in a company because they enforce the standards by which business is conducted within the organization and with customers.

5. Data:

The word data covers the collection of facts stored in the database.

DBMS FUNCTIONS:

A DBMS performs several important functions that guarantee the integrity and consistency of the data in the database. They include

1. Data dictionary management
2. Data storage management
3. Data transformation and presentation
4. Security management
5. Multiuser access control
6. Backup and recovery management
7. Data integrity management
8. Database access languages and application programming interfaces, and
9. Database communication interfaces.

1.Data dictionary management

The DBMS *stores definitions of the data elements and their relationships in a **data dictionary***. In turn, all programs that access the data in the database work

through the DBMS. Additionally, any changes made in a database structure are automatically recorded in the data dictionary, thereby freeing you from having to modify all of the programs that access the changed structure.

2. Data storage management

A modern DBMS system provides storage not only for the data, but also for related data entry forms or screen definitions, report definitions, data validation rules, procedural code, structures to handle video and picture formals, etc. Data storage management is also important for database performance tuning.

Performance tuning relates to the activities that make the database perform more efficiently in terms of storage and access speed. Although the user sees the database as a single data storage unit, the DBMS actually stores the database in multiple physical data files. Such data files may even be stored on different storage media.

3. Data transformation and presentation

The DBMS transforms entered data to conform to required data structures. The DBMS relieves you of the chore of making a distinction between the logical data format and the physical data format. For example, imagine an enterprise database used by a multinational company. An end user in England would expect to enter data such as July 11, 2006 as "11/07/2006." In contract, the same date would be entered in the United States as "07/11/2006." Regardless of the data presentation format, the DBMS must manage the date in the proper format for each country.

4. Security management

The DBMS creates a security system that enforces user security and data privacy Security rules determine which users can access the database, which data items each user can access, and which data operations (read, add, delete, or modify) the user can perform. This is especially important in multi-user database systems where many users access the database simultaneously.

5. Multi-user access control

To provide *data integrity and data consistency*, the DBMS uses sophisticated algorithms to ensure that multiple users can access the database concurrently without compromising the integrity of the database.

6. Backup and recovery management

The *DBMS provides backup and data recovery to ensure data safety and integrity*. Current DBMS systems provide special utilities that allow the DBA to perform routine and special backup and restore procedures. Recovery management deals with the recovery of the database after a failure, such as a bad sector in the disk or a power failure. Such capability is critical to preserving the database's integrity.

7. Data integrity management

The *DBMS promotes and enforces integrity rules, thus minimizing data redundancy and maximizing data consistency*. The data relationships stored in the data dictionary are used to enforce data integrity.

8. Database access languages and application programming interfaces

The DBMS provides data access through a query language. A **query language** is a nonprocedural language – one that lets the user specify what must be done without having to specify how it is to be done. The DBMS may also provide data access to programmers via procedural (3GL) languages such as COBOL, C, PASCAL, Visual Basic, and C++. The DBMS also provides administrative utilities used by the DBA and the database designer to create, implement, monitor, and maintain the database. Structured Query Language (SQL) is the de facto query language and data access standard supported by the majority of DBMS vendors.

9. Database communication interfaces

Current-generation DBMSs accept end-user requests via multiple, different network environments. The DBMS might provide access to the database via the internet through the use of Web browsers such as Mozilla Firefox or Microsoft Internet Explorer. In this environment, communications can be accomplished in several ways.

- ❖ End users can generate answers to queries by filling in screen forms through their preferred Web browser.
- ❖ The DBMS can automatically publish predefined reports on a Web site.
- ❖ The DBMS can connect to third-party systems to distribute information via e-mail or other productivity applications.

MANAGING THE DATABASE SYSTEM:

Although the database system yield considerable advantages over previous data management approaches, database systems do impose significant costs. For example:

Increased costs:

Database systems require sophisticated hardware and software and highly skilled personnel. The cost of maintaining the hardware, software, and personnel required to operate and manage a database system can be substantial.

Management complexity:

Database systems interface with many different technologies and have a significant impact on a company's resources and culture. The changes introduced by the adoption of a database system must be properly managed to ensure that they help advanced by they adoption of a database system databases systems hold crucial company data that are accessed from multiple sources, security issues must be assessed constantly.

Maintaining currency:

To maximize the efficiency of the database system, you must keep your system current. Therefore, you must perform frequent updates and apply the latest patches and security measures to all components. Because database technology advances rapidly, personnel training cots tend to be significant.

Vendor dependence:

Given the heavy investment in technology and personnel training, companies may be reluctant to change database vendors. As a consequence, vendors are less likely to offer pricing point advantages to existing customers and those customers may be limited in their choice of database system components.

CHAPTER-II DATA MODELS

Importance of Data Models

A data model is a (relatively) simple abstraction of a complex real-world data environment. Database designers use data models to communicate with applications programmers and end users. The basic data-modeling components are entities, attributes, relationships, and constraints.

DATA MODEL BASIC BUILDING BLOCKS

The basic building blocks of all data models are entities, attributes, relationships, and constraints. An Entity is anything (a person, a place, a thing, or an event) about which data are to be collected and stored. An entity represents a particular type of object in the real world.

An **attribute** is a characteristic of an entity. For example, a CUSTOMER entity would be described by attributes such as customer last name, customer first name, customer phone, customer address, and customer credit limit. Attributes are the equivalent of fields in file systems.

A **relationship** describes an association among entities. For example, a relationship exists between customers and agents that can be described as follows: an agent can serve many customers, and each customer may be served by one agent. Data models use four types of relationships: one-to-many, many-to-one, many-to-many, and one-to-one. Database designers usually used the shorthand notations 1: M, M: 1, M: M, and 1:1, respectively.

BUSINESS RULES

When database designers go about selecting or determining the entities, attributes, and relationships that will be used to build a data model, they may start by gaining a thorough understanding of what types of data are in an organization, how the data are used, and in what time frames they are used.

From a database point of view, the collection of data becomes meaningful only when it reflects properly defined business rules.

A business rule is a brief, precise, and unambiguous description of a policy, procedure, or principle within a specific organization.

Business rules, derived from a detailed description of an organization's operations, help to create and enforce actions within that organization's environment. Business rules must be rendered in writing and updated to reflect any change in the organization's operational environment.

Properly written business rules are used to define entities, attributes, relationships, and constraints.

To be effective, business rules must be easy to understand and widely disseminated to ensure that every person in the organization shares a common interpretation of the rules. Business rules describe, in simple language, the main and distinguishing characteristics of the data as viewed by the company. *Examples of business rules are as follows:*

- ❖ A customer may generate many invoices.
- ❖ An invoice is generated by only one customer.
- ❖ A training session cannot be scheduled for few then 10 employees or for more than 30 employees.

Note that those business rules establish entities, relationships, and constraints. For example, the first two business rules establish two entities, CUSTOMER and INVOCICE, and a 1:M relationship between those two entities. The third business rule establishes a constraint: no fewer then 10 people and no more than 30 people; two entities, EMPLOYEE and TRAINING; and a relationship between EMPLOYEE and TRAINING.

DISCOVERING BUSINESS RULES

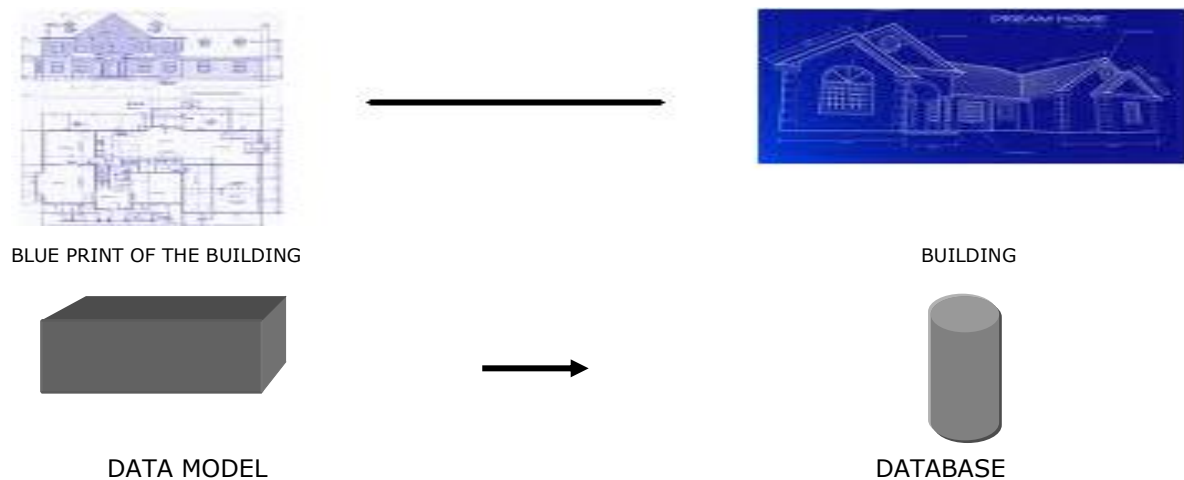
The process of identifying and documenting business rules is essential to database design for several reasons:

- ❖ They help standardize the company's view of data.
- ❖ They can be a communications tool between users and designers.
- ❖ They allow the designer to understand the nature, role, and scope of the data.
- ❖ They allow the designer to understand business processes.
- ❖ They allow the designer to develop appropriate relationship participation rules and constraints and to create an accurate data model.

DATA MODELS:

A data model is a set of concepts that can be used to describe the data such as data types and length, relationships and consistency constraints. *Data models provide necessary data abstraction hiding the details of data storage.* That is, the main purpose of a data model is to provide a level of abstraction. *A data model often includes a set of operations for retrieval and updates and a set of valid user-defined operations that are allowed.*

Implementation of a database approach requires a detailed study of organization's data and information requirements, followed by creating a model depicting the data used by the application and its relationships. *A data model is a blueprint of the data and its relationships used by the system.* The plan created by an architect for a building is an example of a blue print.



The reasons for developing a data model are as follows:

- ❖ Models are abstractions that portray the essentials of a complex problem or structure by filtering out non-essential details, thus making the problem easier to understand.
- ❖ Models help us visualize, understand, organize, complex things thereby promoting cleaner designs and making creation easy.

A data model being an abstraction helps in giving a big picture of how the data and information requirements would be organized thus making it easy to eliminate unnecessarily duplicated data and thereby promoting consistency and accuracy of data.

There are basically three types of data models:

1. Record-based logical models,
2. Object-based logical models
3. Physical models.

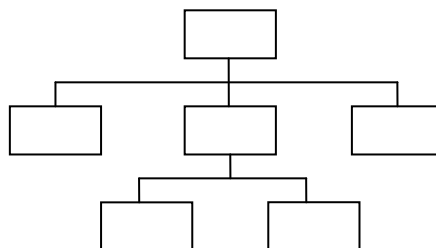
1. Record-based Logical Models:

Record-based logical model is based on the notion of *data stored as a set of records*. Each record consists of a set of fields and each field is of fixed length. Fixed length records reduce the implementation part at the physical level. There are mainly three types of models namely, hierarchical data model, network data model and relational data model. In the hierarchical and network data model, the data is represented in the form of records and the relationships are represented in the form of links (or pointers). In the relational data model, both data and relationships are represented in the form of tables, which consists of a set of records.

(i) Hierarchical Model:

The *hierarchical* database management system is the *oldest* among the database architectures. *A hierarchical DBMS assumed hierarchical relationships between data, that is, parent – child relationship.* In this model, *data is arranged in a top-down structure* that resembles an inverted tree or genealogy chart.

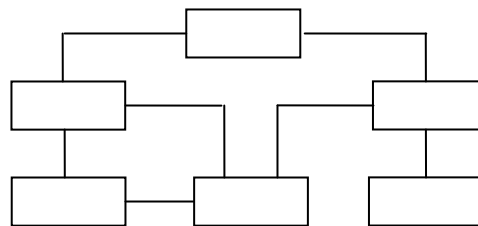
Data is mapped to different nodes of this tree, and they are related in nested, one-to-many relationships. Data at the top node is called the root, data in the nodes at the bottom most layer are called leaves, and data in the nodes at the intermediate levels have one parent node, and one or several child nodes. The hierarchical database management system is best applied when the conceptual data model also resembles a tree, and when most data access begins at the root. An example of Hierarchical database management system is IMS (Information Management System) DBMS.



ii) Network Data Model:

The network data model is formally defined in 1971 in the Conference on Data System Languages (CODASYL).

In the network database management system, *data is mapped to a complex network of nodes*. Although very flexible in-terms of allowing any form of data implementation (a hierarchy is a special case of a network) usually using pointers, this creates significant overhead in storage space and maintenance time. Through network database management system provides fast performance, its design is highly complex (requiring significant programming and database design knowledge and skills), and thereby difficult to change and maintain to accommodate changing data requirements. *The network data model permits modeling many-to-many relationship*. An example of network database management system is IDMS (Integrated Database Management System)



Hierarchical and network models are based on traversing data links to process a database.

(iii) Relational Data Model:

The Relational Database Management Systems are relatively new form historical perspective and built from theoretical considerations of data structures by E.F.Codd.

The relational database management system unlike the Hierarchical or network database management systems does not involve any links. *It represents data and relationships among data by a collection of table, each of which had a number of columns with unique names*. The concept of tables and rows and columns is extremely simple and easy to understand. Complex network diagrams used with the hierarchical and network databases are not used with a relational database.

A relational database provides a much higher degree of data independence than hierarchical and network databases. In relational databases, tables are not hard-linked to one another. Columns can be added to tables, tables can be added

to the database and new data relationship can be added with little or no restructuring of the tables.

CustID	Name	Location	City
1021	Raja Sekhar	Koti	Hyderabad
1022	Praneeth	Dilshuk Nagar	Hyderabad
1023	Ashish	Kotha Pet	Guntur
1024	Geetha	Musab Tank	Hyderabad
1112	Guna	Jubli Hills	Hyderabad

Main difference between relational and non-relational (network and hierarchical) data models.

Non Relational	Relational
Relationships maintained by pointers	Relationship defined by data content
Access to data through predefined paths	Any data item is always directly addressable
Structural changes very difficult, through it is not impossible.	Very flexible at logical level
Inherently complicated data structures	Conceptually simple data structures
Ordering of records significant	No significance in order of rows
Accessing programs require procedural and navigational capability	No such restrictions Supported and maintained by standard set of operations.

The relational data model has established itself as the primary data model best suited for the commercial data processing applications. Examples of RDBMS are Oracle, Sybase, Informix, MS SQL Server, DB2 and PostgreSQL.

2. Object-based Logical Models:

In object-based logical model database is structured in the form of objects of several types. It allows specifying data constraints separately. These models are used in describing data at the conceptual and view levels of abstraction. Some of the most widely known data models that fall under this category are: Entity-Relationship Data Model and Object Oriented Data Model.

(i) Entity-Relationship Model:

The main constructs of the ER model are entities and their attributes, and the relationships between the entities.

(ii) Object –Oriented Model:

Object –oriented model consists of objects. Each object contains attributes and methods, which operate on the attributes. Each attribute value stored in the form of variables and methods are written as a block of code. Two or more objects communicate each other using object methods. Each object is identified with a unique identifier to distinguish with other objects.

3. Physical Data Model:

Physical data model captures the data at the lowest level that are used for database-system implementation. These concepts are mainly meant for computer specialists, but not for the end users. Here data is represented as record formats, record orderings and access paths. Examples of Physical data models are Unifying model and frame-memory model.

CHAPTER-III

THE RELATION DATA MODEL

LOGICAL VIEW OF DATA

The relational data model changed all of that by allowing the designer to focus on the logical representation of the data and its relationships, rather than on the physical storage details. To use an automotive analogy, the relational database uses an automatic transmission to relieve you of the need to manipulate clutch pedals and gearshifts. In short, the relational model enables you to view data logically rather than physically.

The logical view of the relational database is facilitated by the creation of data relationships based on a logical construct known as a table.

RELATIONAL DATAMODEL:

E.F. Codd first introduces the relational data model in 1970.

Basic Definitions:

“The relational data model represents data in the form of tables.”

The relational model is based on mathematical theory and therefore has a solid theoretical foundation.

The relational data model consists of the following three components

1. **Data Structure:** Data are organized in the form of tables with rows and columns.
2. **Data manipulation:** Powerful operations (using the SQL language) are used to manipulate data stores in the relations.
3. **Data Integrity:** Facilities are included to specify business rules that maintain the integrity of data when they are manipulated.

RDBMS Terminology:

Formal Relational term	Informal equivalent
Relation	Table
Tuple	Row, Record
Attribute	Column, Field
Cardinality	No. of Rows
Degree	No. of columns
Domain	Set of legal values
Primary key	Unique identification

Relational Data Structure:

“A Relation is a named, two-dimensional table of data”. Each relation (or table) consists of a set of named columns and an arbitrary number of unnamed rows.

Example: The following example shows a relation named EMPLOYEE1. This relation contains the following attributes describing employees:

Emp-ID, Name, Dept_name, and salary.

Emp_ID	Name	Dept_Name	Slaary
100	Uday	Accounting	55000
201	Vishnu	Marketing	45000
104	Tarun	Finance	65000
106	Seenu	Marketing	35000
166	Rani	Info systems	34000

Fig(a) EMPLOYEE1 relation with sample data

We can express the structure of a relation by a shorthand notation in which the name of the relation is followed (in parentheses) by the names of the attributes in that relation.

For EMPLOYEE1 we would have

EMPLOYEE1 (Emp_ID, Name, Dept_Name, Salary)

Properties of Relations or Characteristics of Relational Table

We have defined relations as two-dimensional tables of data. However, not all tables are relations. Relations have several properties that distinguish them from non-relational tables. We summarize these properties below.

1. A table is perceived as a two-dimensional structure composed of rows and columns.
2. Each relation (or table) in a database has a unique name.
3. An entry at the intersection of each row and column is automatic (single valued). There can be no multi-valued attributes in a relation.
4. Each row is unique; no two rows in a relation are identical.
5. Each attribute (or column) within a table has a unique name.
6. The sequence of columns (left or right) is insignificant. The columns of a relation can be interchanged without changing the meaning or use of the relation.

7. The sequence of rows (top to bottom) is insignificant. As with column, the rows of a relation may be interchanged or stored in any sequence.
8. All values in a column must conform to the same data format. For example, if the attribute is assigned an integer data format, all values in the column representing that attribute must be integers.

Relational database Keys

A key consists of one or more attributes that determine other attributes. We must be able to store and retrieve a row of data in a relation based on the data values in that row. To achieve this goal, every relation must have some type of keys.

1. SUPER KEY
2. CANDIDATE KEY
3. PRIMARY KEY
4. SECONDARY KEY
5. FOREIGN KEY

1. Super Key:

An attribute (or combination of attributes) that uniquely identifies each row in a table.

2. Candidate Key:

A minimal super key that is one that does not contain a subset of attributes that is itself a super key. (OR) **A candidate key can be described as a super key without redundancies, that is, a minimal super key.**

Example: Using this distinction, note that the composite key.

STU_NUM, STU_LNAME

is a super key, but it is not a candidate key because STU_NUM by itself is a candidate key! The combination STU_LNAME, STU_FNAME, STU_INT, STU_PHONE might also be a candidate key, as long as you discount the possibility that two students share the same last name, first name, initial, and phone number.

3. Primary Key:

A primary key is an attribute (or combination of attributes) that "uniquely identifies each row in a relation". We designate a primary key by underlining the attribute name.

Example: The primary key for the relation EMPLOYEE1 is Emp_ID.

In shorthand notation express this relation as follows

EMPLOYEE1 (Emp_ID, Name, Dept_Name, Salary)

4. Secondary (OR) Composite Key:

A composite key is a primary key that consists of more than one attribute. (OR) A key that is used strictly for data retrieval purposes.

Example: A customer is not likely to know his or her customer number (primary key), but the combination of last name, first name, middle initial, and telephone number is likely to make a match to the appropriate table row.

5. Foreign Key:

We must represent the relationship between TWO tables or Relations. This is accomplished through the use of foreign keys.

“A foreign key is an attribute in a relation of a database that serves as the primary key of another relation in the same database.”

Example: Consider the relations EMPLOYEE1 and DEPARTMENT

EMPLOYEE1 (Emp_ID, Name, Dept_Name, Salary)

DEPARTMENT (Dept Name, Location, Fax)

This attribute Dept_Name is a foreign key in EMPLOYEE1. It allows a user to associate any employee with the department to which he or she is assigned. The foreign key represented in a notation by using a “Dashed Underline”.

INTEGRITY CONSTRAINTS (or) RULES:

The relational data model includes several types of constraints, or business rules, the purpose is to facilitate maintaining the accuracy and integrity of data in the database.

The major types of integrity constraints are

1. Domain Constraint
2. Entity Integrity
3. Referential Integrity
4. Action Assertion

1. Domain constraint:

All of the values that appear in a column of a relation must be taken from the same domain. A domain usually consists of the following components:

Domain Name, Meaning, Data type, Size (or length), and allowable values or allowable range.

The following table shows the domain definitions for the domains associated with the attributes.

Table: Domain Definitions for selected attributes:

Attribute	Domain Name	Description	Domain
Customer_ID	Customer_Ids	Set of all possible customer Id	Character: Size 5
Customer_Name	Customer_Names	Set of all possible customer names	Character: Size 25
Customer_Address	Customer_Addresses	Set of all possible customer addresses	Character: Size 30
Customer_City	Cities	Set of all possible cities	Character: Size 20
Customer_State	States	Set of all possible states	Character: Size 2
Customer_Zip	Zips	Set of all possible zip codes	Character: Size 10
Order_ID	Order_IDs	Set of all possible order IDs	Character: Size 5
Order_date	Order_Date	Set of all possible order dates	Character: Size mm-dd-yy
Product_ID	Product_IDs	Set of all possible product IDs	Character: Size 5
Product_Description	Product_Descriptions	Set of all possible product descriptions	Character: Size 25
Product_Finish	Product_Finishes	Set of all possible product finishes	Character: Size 12
Standard_Price	Unit_Price	Set of all possible unit prices	Character: Size 6 digits
On_Hand	On_Hands	Set of all possible on hands	Character: Size 3 digits

2. Entity Integrity:

The entity integrity rule is designed to assure that "every relation has a primary key" and that the data values for that primary key are all valid. In particular, it guarantees that every primary key attribute is not null.

The entity integrity rule states the following:

No primary key attribute (or component of a primary key attribute) may be null.

3. Referential Integrity:

In the relational data model, associations between tables are defined through the use of foreign keys.

Example: The association between the CUSTOMER and ORDER tables is defined by including the Customer_ID attribute as a foreign key in ORDER. CUSTOMER (Customer_ID, Customer_Name, Address, City, State, Zip) ORDER (Order_ID, Order_Date, Customer_ID).

A referential integrity constraint is a rule that maintains consistency among the rows of two relations. The rule states that if there is a foreign key in one relation, either each foreign key value must match a primary key value in another relation or the foreign key value must be null.

4. Action Assertions:

Business rules and introduced a new category of business rules we called action assertions.

Example: A typical action assertion might state the following: "A person may purchase a ticket for the all-star game only if that person is a season – ticket holder."

RELATIONAL SET OPERATORS:

Relational algebra defines the theoretical way of manipulating table contents using the eight relational operators

1. SELECT
2. PROJECT
3. JOIN
4. INTERSECT
5. UNION
6. DIFFERENCE
7. PRODUCT
8. DIVIDE

1. UNION

Union combines all rows from two tables, excluding duplicate rows. The tables must have the same attribute characteristics (the columns and domains must be identical) to be used in the UNION. When two or more tables share the same number of columns, when the columns have the same names, and when they share the same (for compatible) domains, they are said to be union-compatible.

	P_CODE	P_DESCRIPTION	PRICE
▶	123456	Flashlight	\$ 5.26
	123457	Lamp	\$ 25.15
	123458	Box Fan	\$ 10.99
	123455	gv battery	\$ 1.92
	254467	100w buib	\$ 1.47
	311452	powerdrill	\$ 34.99

UNION

	P_CODE	P_DESCRIPTION	PRICE
▶	345678	Microwave	\$ 160.00
	345679	Dishwasher	\$ 500.00

YIELDS



	P_CODE	P_DESCRIPTION	PRICE
▶	123456	Flashlight	\$ 5.26
	123457	Lamp	\$ 25.15
	123458	Box Fan	\$ 10.99
	213345	gv battery	\$ 1.92
	254467	100w buib	\$ 1.47
	311452	powerdrill	\$ 34.99
	345678	Microwave	\$ 160.00
	345679	Dishwasher	\$ 500.00

2. INTERSECT

Intersect yields only *the rows that appear in both tables*.

Note: you cannot use INTERSECT if one of the attributes is numeric and one is character-based.

	F_NAME		F_NAME		F_NAME		
▶	GEORGE	INTERSECT	▶	JANE	YIELDS	▶	JANE
	JANE			WILLIAMS		→	
	ELAIN			JORGE			
	WILFRED			DENNIS			
	JORGE						

3. DIFFERENCE

Difference yields *all rows in one table that are not found in the other table*, that is, subtracts one table from the other.

	F_NAME		F_NAME		F_NAME			
▶	GEORGE	DIFFERENCE	▶	JANE	YIELDS	▶	GEORGE	
	JANE			WILLIAMS		→		ELAINE
	ELAIN			JORGE				WILFRED
	WILFRED			DENNIS				
	JORGE							

4. PRODUCT

Product yields *all possible pairs of rows from two tables – also known as the Cartesian product*. Therefore if one table has six rows and the other table has three rows, the PRODUCT yields a list composed of $6 \times 3 = 18$ rows.

	P_CODE	P_DESCRIPT	PRICE		STORE	AISLE	SHELF
▶	123456	Flashlight	\$ 5.26	PRODUCT	▶	23	W 5
	123457	Lamp	\$ 25.15			24	K 9
	123458	Box Fan	\$ 10.99			25	Z 6
	123455	gv battery	\$ 1.92				
	254467	100w buib	\$ 1.47				
	311452	Powerdrill	\$ 34.99				

↓

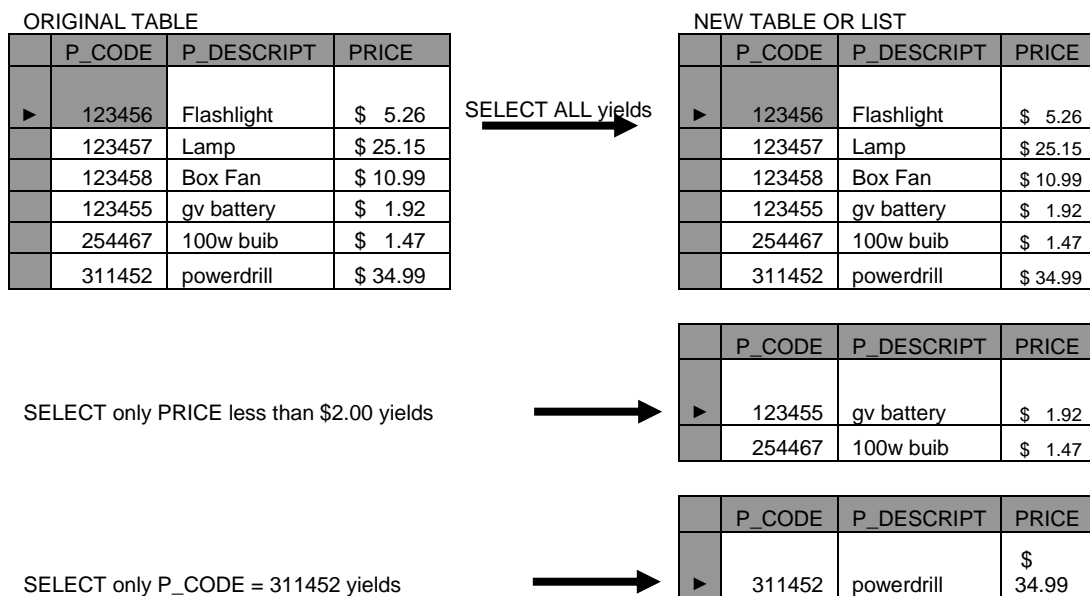
yields

	P_CODE	P_DESCRIPT	PRICE	STORE	AISLE	SHELF
▶	123456	Flashlight	\$ 5.26	23	W	5
	123456	Flashlight	\$ 5.26	24	K	9
	123456	Flashlight	\$ 5.26	25	Z	6
	123457	Lamp	\$ 25.15	23	W	5
	123457	Lamp	\$ 25.15	24	K	9
	123457	Lamp	\$ 25.15	25	Z	6
	123458	Box Fan	\$	23	W	5

			10.99			
	123458	Box Fan	\$ 10.99	24	K	9
	123458	Box Fan	\$ 10.99	25	Z	6
	213345	gv battery	\$ 1.92	23	W	5
	213345	gv battery	\$ 1.92	24	K	9
	213345	gv battery	\$ 1.92	25	Z	6
	311452	powerdrill	\$ 34.99	23	W	5
	311452	powerdrill	\$ 34.99	24	K	9
	311452	powerdrill	\$ 34.99	25	Z	6
	254467	100w buib	\$ 1.47	23	W	5
	254467	100w buib	\$ 1.47	24	K	9
	254467	100w buib	\$ 1.47	25	Z	6

5. SELECT

Select also known as RESTRICT, yield values *for all rows found in a table*. SELECT can be used to list all of the row values, or it can yield only those row values that match a specified criterion. In other words, SELECT yield a horizontal subset of a table.



6. PROJECT

Project yields *all values for selected attributes*. In other words, PROJECT yields a *vertical subset of a table*.

ORIGINAL TABLE					NEW TABLE OR LIST	
	P_CODE	P_DESCRIPT	PRICE			PRICE
▶	123456	Flashlight	\$ 5.26	PROJECT PRICE yield →	▶	\$ 5.26
	123457	Lamp	\$ 25.15			\$ 25.15
	123458	Box Fan	\$ 10.99			\$ 10.99
	123455	gv battery	\$ 1.92			\$ 1.92
	254467	100w buib	\$ 1.47			\$ 1.47
	311452	powerdrill	\$ 34.99			\$ 34.99

PROJECT P_DESCRIPT and PRICE yields



	P_DESCRIPT	PRICE
▶	Flashlight	\$ 5.26
	Lamp	\$ 25.15
	Box Fan	\$ 10.99
	gv battery	\$ 1.92
	100w buib	\$ 1.47
	powerdrill	\$ 34.99

PROJECT P_CODE and PRICE yields



	P_CODE	PRICE
▶	123456	\$ 5.26
	123457	\$ 25.15
	123458	\$ 10.99
	123455	\$ 1.92
	254467	\$ 1.47
	311452	\$ 34.99

7. JOIN

Join allows information *to be combined from two or more tables*. JOIN is the real power behind the relational database, *allowing the use of independent tables linked by common attributes*.

A **natural join** links tables by selecting only the rows with common values in their common attribute. A natural join is the result of a three-stage process.

Table name: CUSTOMER

	CUS_CODE	CUS_LNAME	CUS_ZIP	AGENT_CODE
▶	1132445	Walker	32145	231
	1217782	Adares	32145	125
	1312243	Rakowski	34129	167
	1321242	Rodriguez	37134	125
	1542311	Smithson	37134	421
	1657399	Vanloo	32145	231

Tble name: AGENT

	AGENT_CODE	AGENT_PHONE
▶	125	6152439887
	167	6153426778
	231	6152431124
	333	9041234445

a. First, a PRODUCT of the tables is created, yielding the results shown in below figure.

	CUS_CODE	CUS_LNAME	CUS_ZIP	CUSTOMER.AGENT_CODE	AGENT.AGENT_CODE	AGENT_PHONE
▶	1132445	Walker	32145	231	125	6152439887
	1132445	Walker	32145	231	167	6153426778
	1132445	Walker	32145	231	231	6152431124
	1132445	Walker	32145	231	333	9041234445
	1217782	Adares	32145	125	125	6152439887
	1217782	Adares	32145	125	167	6153426778
	1217782	Adares	32145	125	231	6152431124
	1217782	Adares	32145	125	333	9041234445
	1312243	Rakowski	34129	167	125	6152439887
	1312243	Rakowski	34129	167	167	6153426778
	1312243	Rakowski	34129	167	231	6152431124
	1312243	Rakowski	34129	167	333	9041234445
	1321242	Rodriguez	37134	125	125	6152439887
	1321242	Rodriguez	37134	125	167	6153426778
	1321242	Rodriguez	37134	125	231	6152431124
	1321242	Rodriguez	37134	125	333	9041234445
	1542311	Smithson	37134	421	125	6152439887
	1542311	Smithson	37134	421	167	6153426778
	1542311	Smithson	37134	421	231	6152431124
	1542311	Smithson	37134	421	333	9041234445
	1657399	Vanloo	32145	231	125	6152439887
	1657399	Vanloo	32145	231	167	6153426778
	1657399	Vanloo	32145	231	231	6152431124
	1657399	Vanloo	32145	231	333	9041234445

b. A SELECT is performed on the output of Step a to yield only the rows for which the AGENT_CODE values are equal. The common column(s) is (are) referred to as the join column(s). Step b yields the results shown in below figure.

	CUS_CODE	CUS_LNAME	CUS_ZIP	CUSTOMER.AGENT_CODE	AGENT.AGENT_CODE	AGENT_PHONE
▶	1217782	Adares	32145	125	125	6152439887
	1312243	Rakowski	34129	167	125	6152439887
	1321242	Rodriguez	37134	125	167	6153426778
	1132445	Walker	32145	231	231	6152431124
	1657399	Vanloo	32145	231	231	6152431124

c. A PROJECT is performed on the results of Step b to yield a single copy of each attribute, thereby eliminating duplicate columns. Step c yields the output shown in below figure.

	CUS_CODE	CUS_LNAME	CUS_ZIP	AGENT.AGENT_CODE	AGENT_PHONE
▶	1217782	Adares	32145	125	6152439887
	1312243	Rakowski	34129	125	6152439887
	1321242	Rodriguez	37134	167	6153426778
	1132445	Walker	32145	231	6152431124
	1657399	Vanloo	32145	231	6152431124

Equi join:

Another form of join, known as **equijoin**, links tables on the basis of an equality condition that compares specified columns of each table. The outcome of the equijoin does not eliminate duplicate column, and the condition or criterion used to join the tables must be explicitly defined. The equijoin takes its name from the equality comparison operator (=) used in the condition. If any other comparison operator is used, the join is called a theat join.

LEFT OUTER JOIN & RIGHT OUTER JOIN:

A **left outer join** yields all of the rows in the CUSTOMER table, including those that do not have a matching value in the AGENT table.

	CUS_CODE	CUS_LNAME	CUS_ZIP	AGENT.AGENT_CODE	AGENT_PHONE
▶	1217782	Adares	32145	125	6152439887
	1312243	Rakowski	34129	125	6152439887
	1321242	Rodriguez	37134	167	6153426778
	1132445	Walker	32145	231	6152431124
	1657399	Vanloo	32145	231	6152431124
	1542311	Smithson	37134	421	

A **right outer join** yields all of the rows in the AGENT table, including those that do not have matching values in the CUSTOMER table.

	CUS_CODE	CUS_LNAME	CUS_ZIP	AGENT.AGENT_CODE	AGENT_PHONE
▶	1217782	Adares	32145	125	6152439887
	1312243	Rakowski	34129	125	6152439887
	1321242	Rodriguez	37134	167	6153426778
	1132445	Walker	32145	231	6152431124
	1657399	Vanloo	32145	231	6152431124
				333	9041234445

8. DIVIDE

Divide requires the use of one single-column table and one 2-column table.

a. Table 1 is “divided” by the table 2 to produce table 3. tables 1 and 2 both contain the column CODE but do not share LOC.

b. To be included in the resulting Table 3, a value in the unshared column (LOC) must be associated (in the dividing Table2) with every value in Table 1.

c. The only value associated with both A and B is 5.

	CODE	LOC
▶	A	5
	A	9
	A	4
	B	5
	B	3
	C	6
	D	7
	D	8
	E	8

	CODE
▶	A
	B

yields

	LOC
▶	5

The Data Dictionary and the system catalog

The data dictionary provides a detailed accounting of all tables found within the user/designer-created database. Thus, *the data dictionary contains at least all of the attribute names and characteristics for each table in the system. In short, the data dictionary contains metadata – data about data.*

The data dictionary is sometimes described as “the database designer’s database” because it records the design decisions about tables and their structures.

TAB LE	ATTRIBUTE	CONTENTS	TYPE	FORMAT	RANGE	REQUI- OR	FK REFEREN-
-----------	-----------	----------	------	--------	-------	--------------	----------------

NAME	NAME					RED	PK	FK	TABLE
CUSTOMER	CUS_CODE	Customer account	CHAR(5)	99999	10000-99999	Y	PK	AGENT	
	CUS_LNAME	Customer last name	VCCHAR(20)	Xxxxxxxxxx		Y			
	CUS-FNAME	Customer first name	VCHAR(20)	Xxxxxxxxxx		Y			
	CUS_INITIAL	Customer initial	CHAR(1)	X	100-999		FK		
	CUS_RENEW_DATE	Customer insurance renewal date	DATE	dd-mmm-yyyy					
	AGENT_CODE	Agent code	CHAR(3)	999					
AGENT	AGENT_CODE	Agent code	CHAR(3)	999		Y	PK		
	AGENT_AREACODE	Agent area code	CHAR(3)	999		Y			
	AGENT_PHONE	Agent telephone number	CHAR(8)	999-9999	0.00 - 9,999,999.99	Y			
	AGENT_LNAME	Agent ,last name	VCHAR(20)	Xxxxxxxxxxxx		Y			
	AGENT_YTD_SLS	Agent year-to-date sales	NUMBER(9,2)	9,999,999.99		Y			
FK = Foreign Key PK = Primary Key CHAR = Fixed character length data (1-255 characters) VARCHAR = Variable character length data (1-2,000 characters) NUMBER = Numeric data (NUMBER(9,2) is used to specify numbers with two decimal places and up to nine digits, including the decimal places. Some RDBMSs permit the use of a MONEY or CURRENTY data type.)									
NOTE: Telephone area codes are always composed of digits 0-9. because area codes are not used arithmetically, they are most efficiently stored as character data. Also, the area codes are always composed of three digits. Therefore, the area code data type is defined as CHAR(3). On the other hand, names do not conform to some standard length. Therefore, the customer \first names are defined as VARCHAR(20), thus indicating that up to 20 characters maybe used to store the names. Character data are shown as left0-justified.									

INDEXES

The index (in either a manual or a computer system) points you to the book's location, thereby making retrieval of the book a quick and simple matter. *An index is an orderly arrangement used to logically access rows in a table.*

An index is composed of an index key and a set of pointer's the index key is, in effect, the index's reference point. More formally, an index is an ordered arrangement of keys and pointers. Each key points to the location of the data identified by the key.

For example, suppose you want to look up all of the paintings created by a given painter in the Ch03_Museum database in Figure 3A. Without an index, you must read each row in the PAINTING table and see if the PANINTER_NUM matches the requested painter. However, if you index the PAINTER table and use the index

key PAINTER_NUM, you merely need to look up the appropriate PAINTER_NUM in the index and find the matching pointers.

FIGURE 3A

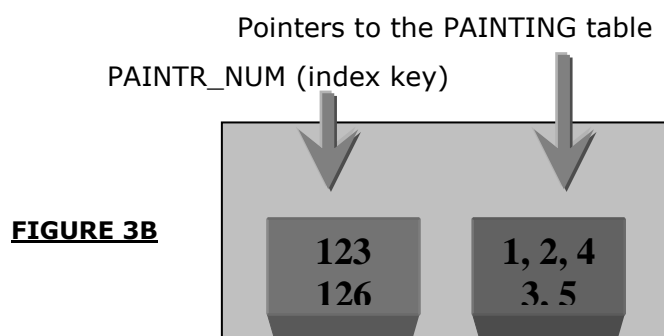
Table name: PAINTER
 Primary key: PAINTER_NUM
 Foreign key: none
 Database name: Cho3_Museum

	PAINTER_NUM	PAINTER_LNAME	PAINTER_FNAME	PAINTER_INITIAL
-	123	Ross	Georgette	P
+	126	Itero	Julio	G

Table name: PAINTING
 Primary key: PAINTING_NUM
 Foreign key: PAINTER_NUM

	PAINTING_NUM	PAINTING_TITLE	PAINTER_NUM
▶	1338	Dawn Thunder	123
	1339	Vanilla Roses To Nowhere	123
	1340	Tired Flounders	126
	1341	Hasty Exit	123
	1342	Plastic Paradise	126

As you examine Figure 3B and compare it to the Ch03_Museum database tables showing in Figure 3A, note that the first PAINTER_NUM index key value (123) is found in records 1, 2, and 4 of the PAINTING table in Figure 3A. the second PAINTER_NUM index key value (126) is found in records 3 and 5 of the PAINTING table in Figure 3A.



DBMS s use indexes for many different purposes. *An index can be used to retrieve data more efficiently. But indexes can also be used by a DBMS to retrieve data ordered by a specific attribute or attributes.*

Indexes play an important role in DBMS s for the implementation of primary keys. When you define a table’s primary key, the DBMS automatically creates a unique index on the primary key column(s) you declared.

A table can have many indexes, but each index is associated with only one table. The index key can have multiple attributes (composite index). Creating an index is easy.

CODDS RELATIONAL DATABASE RULES (OR) THE CODD COMMANDMENTS

E.F. Codd defined 12 rules for relational database, which any database should satisfy in order to be regarded as a relational database. These 12 rules are called as CODD 12 commandments. It is fact that no commercial RDBMS existing today satisfies all these 12 rules.

Besides the 12 Codd Rules, there is a single, overall rule which in some ways covers all others and is called Rule 0. Rule 0 states that any truly relational database must be manageable entirely through its own relational capabilities.

These twelve rules can be used as the basic relational design criteria, and as such are clear indications of the purity of the relational concept.

THE TWELVE CODD RULES

1. The Information Rule
2. Guaranteed Access Rule
3. Systematic Treatment of Null Values
4. The Data Description Rule
5. The Comprehensive Sub-Language Rule
6. The view Updating Rule
7. High-Level, Update, and Delete
8. Physical Data Independence
9. Logical Data Independence
10. Integrity Independence Rule
11. The Distribution Independence Rule
12. The Non-Subversion Rule

Rule 1: The Information Rule

This rule states that all data in an RDBMS should exist in a relational form, i.e., in the form of rows and columns.

Rule 2: Guaranteed Access Rule

Any piece of data can be accessed from an RDBMS using a combination of table name, column name, and primary key value.

Rule 3: Systematic Treatment of Null Values

Missing information in a table is represented as a NULL value. Support for NULL values in a RDBMS must be consistent and independent of data type.

Rule 4: The Data Description Rule

This rule requires that a description of the database, or data about data (metadata) should be stored in an online data dictionary that is part of the RDBMS and is therefore relational (composed of tables).

Rule 5: The Comprehensive Sub-Language Rule

This rule states that an RDBMS must be completely manageable through its own dialect of SQL. This rule also scopes the functionality of SQL to include the following:

- ❖ Data Definition Language – DDL
- ❖ View Definition Language – DDL
- ❖ Data Manipulation Language – DML
- ❖ Integrity Constraints – DDL
- ❖ Authorization – DCL
- ❖ Transaction Boundaries – DCL

Rule 6: The view Updating Rule

This rule states that all views that are theoretically updatable are also updatable by the system. This rule holds true in a very restricted sense.

Rule 7: High-Level, Update, and Delete

This rule insists that a DBMS's relational language be able to insert, update, and delete more than one row with a single command.

Rule 8: Physical Data Independence

This rule states that application programs remain logically unimpaired whenever any changes are made in either the storage representation of data or access methodology of data.

Rule 9: Logical Data Independence

This rule states that application programs remain logically unimpaired when information preserving changes of any kind that theoretically permit ,un-impairment are made to the base tables.

Rule 10: Integrity Independence Rule

This rule addresses two issues:

- ❖ **Entity Integrity:** Each and every instance of an entity should be uniquely identifiable. This is accomplished through a Primary Key.
- ❖ **Referential Integrity:** For each distinct non-null foreign key value in a relational database, there must exist a matching primary key value from the same domain.

Rule 11: The Distribution Independence Rule

This rule states that a distributed database must look to the user or the application as a centralized database. Application programs and interactive users should not be required to know where data are stored.

Rule 12: The Non-Subversion Rule

This rule states that if an RDBMS supports a lower level language that permits for examples, record-at-a-time processing, then this language must not be able to bypass any integrity rules or constraints defined in the higher level, set-at-a-time relational language.

Unit - II

DATA MODELING AND NORMALIZATION

ER- MODEL

ATTRIBUTES

ENTITY SETS

RELATIONSHIPS

DATABASE DESIGN CHALLENGES CONFLICTING GOALS

EER-MOEL

SUPER TYPE

SUB TYPE

GENERALIZATION

SPECIALIZATION

CONSTRAINTS

ENTITY CLUSTERING

ENTITY INTEGRITY & SELECTING PRIMARY KEYS

NORMALIZATION

NEED FOR NORMALIZATION

BASIC NORMAL FORMS

HIGHER NORMAL FORMS

NORMALIZATION DATABASE DESIGN

DE-NORMALIZATION

Unit-II

Modeling data in the organization

Entity relation model (ER Model)

An ER model is a detailed, logical and representation of the data for an organization for a business area.

"The ER model is expressed in term of entities the relationships among those entities and attributes of the both the entities and their relationships.

An ER model is normally expressed as an entity relationship diagram, which is graphically representation of an ER model.

Attributes

An Attribute is a "Property or characteristics of an entity type." The following are a some typical entity types and their associated attributes.

STUDENT Student_ID, Student_Name, Home_Address, Phone_Number,

AUTOMOBILE Registration_Number, Color, Model, Weight, Horsepower.

Rules for giving names to attributes

1. In naming attributes we use an initial capital letter followed by lower case letters.
2. If an attribute name consists of two words we use an underscore (_) character.
3. To connect the words with underscore we start each word with a capital letter.

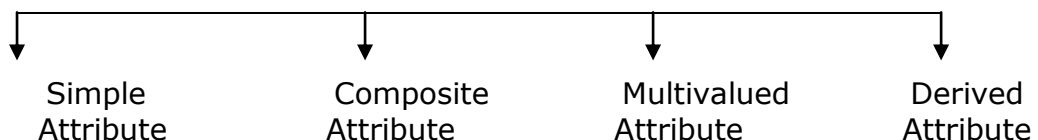
Example: Name, Emp_Address.

In ER diagram we represent an attribute using the symbol "Ellipse".



Types of attributes

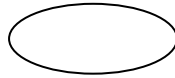
The attributes are classified into four types.



1. Simple attribute

A simple or atomic attribute is an attribute that "cannot be broken down into small components"

Example: Color, this attribute cannot be smaller components broken down the simple attribute represented by "ellipse symbol".



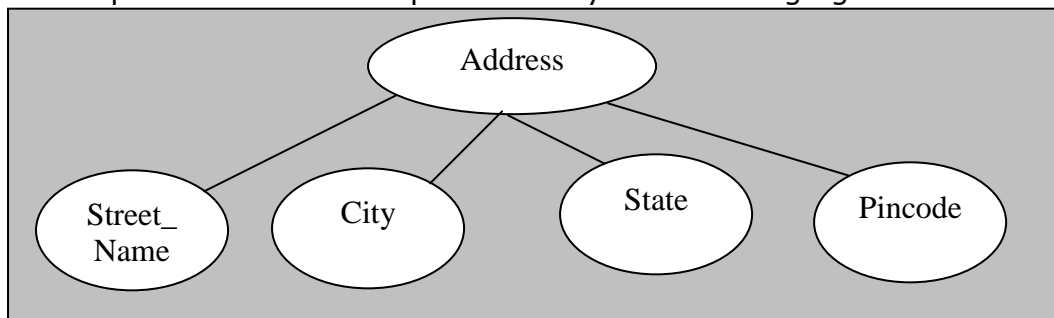
2. Composite attribute

An attribute that can be broken down into components parts.

Example: The most common example is address, which usually be broken down into the following components.

Street_Name, City, State, Pincode.

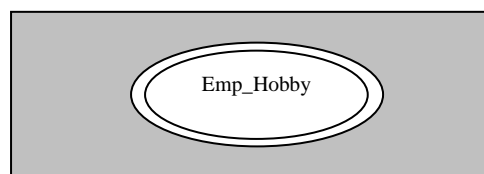
A composite attribute is represented by the following figure.



3. Multi-valued attribute

An attribute that may take on "more than one value for a given entity instance".

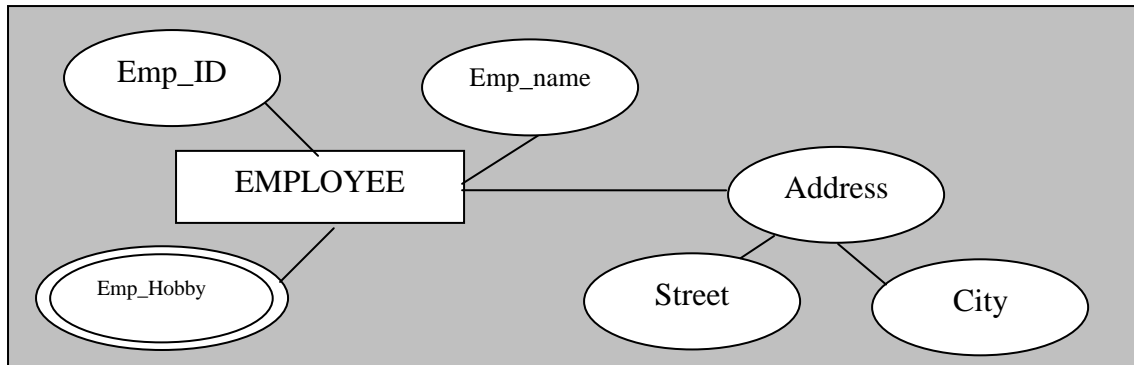
We indicate a multi-valued attribute with an ellipse with *double lines*, the following is the symbol of multi-valued attribute



Example: The employee (Employee) entity type having the following attributes

Emp_ID, Emp_Name, Emp_Address, Emp_Hobby

In the above attribute the last attribute i.e. hobby that can take more than one value. So Emp_Hobby is a multi-valued attribute.

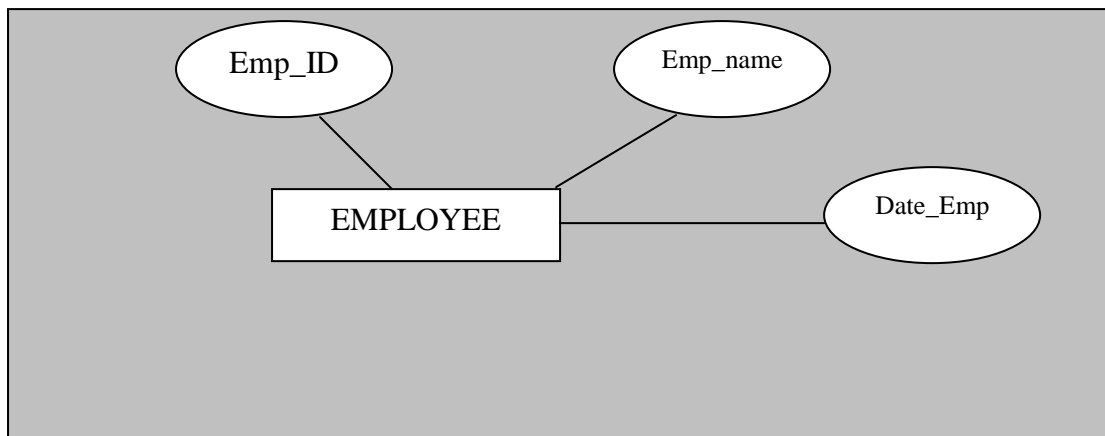


4. Derived attribute

A derived attribute is an attribute "whose values can be calculated from related attribute values", we indicate a derived attribute in an ER diagram by using an ellipse with dashed line.

Years_Employed know how many years a person has been employees, that value can be generated using date-employee and to-days date.

Derived attribute



Years_Employed is a derived attribute.

ENTITIES, ENTITY TYPES AND ENTITY INSTANTS

Entities

An entity is a person, place, object event or concept in the uses environment about which the organization wishes to maintain data.

Example:

- Person : EMPLOYEE, STUDENT, PATIENT
- Place : STORE, WAREHOUSE, STATE
- Object : MACHINE, BUILDING, AUTOMOBILE
- Event : SALE, REGISTRATION, RENEWAL

Concept : ACCOUNT, COURSE, WORK CENTER

Entity type

An entity type is a "Collection of entities that share common properties or characteristics". Each entity type in an ER model is given a name. Since the name represents a collection of items, it is always singular. We use capital letters for names of entity types. In an ER diagram the entity is represented by the following symbol (Rectangle).



Entity instance

An entity instance is a "Single occurrence of an entity type". The following example explain the differences between an entity type and two of its instances.

<u>Entity type (EMPLOYEE)</u>	
Attributes	
EMPLOYEE NUMBER	CHAR (10)
NAME	CHAR (25)
ADDRESS	CHAR (30)
CITY	CHAR (20)
STATE	CHAR (2)
ZIP	CHAR (9)
DATE HIRED	DATE
BIRTHDATE	DATE
Two Instances of EMPLOYEE:	
101	102
AKBAR	KAREEM
LBS Road	Nagaraja
Piler	Hyderabad
A.P.	A.P.
517214	500082
07-06-1990	07-07-1986
07-06-1991	07-07-1987

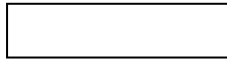
Entity Types

1. Weak entity type
2. Strong entity type
3. Associative entity type

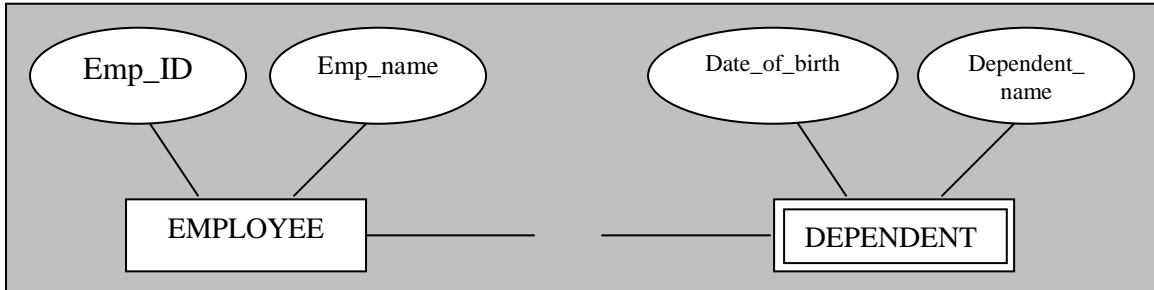
1. Weak entity type

*An entity type whose existence "depends on some other entity type".
A weak entity is as indicated by the double line rectangle.*





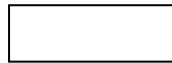
Example: Consider the following example
 An EMPLOYEE - Emp_ID, Emp_name
 DEPENDENT - Date_of_birth , dependent_name.



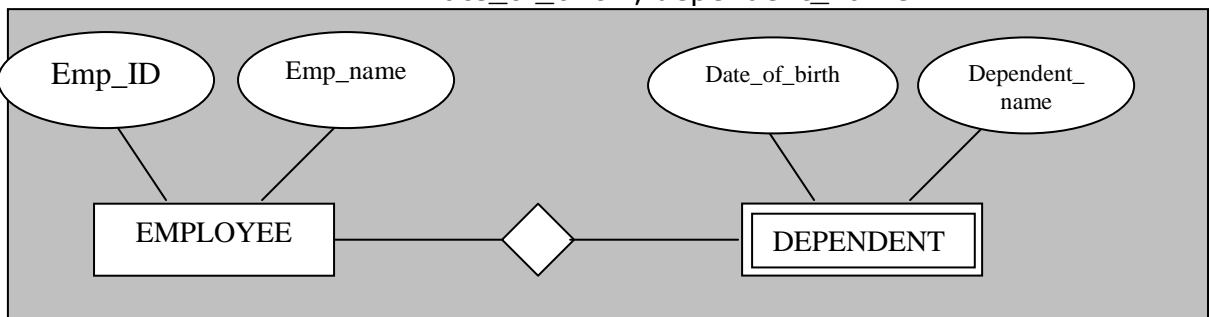
2.

Strong entity type

"An entity that exists independently of other entity types" that means it does not depend on another entity type. Its basic symbol of representing a strong entity is as follows.



Example: Consider the following example
 An EMPLOYEE - Emp_ID, Emp_name
 DEPENDENT - Date_of_birth , dependent_name.



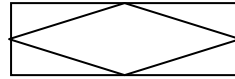
In the above example the **EMPLOYEE** entity is a **strong or owner** entity. Because this entity is not depend on any other entity type. So, EMPLOYEE entity is an owner or strong entity.

The relationship between the weak entity type and its owner entity is called an identifying relationship.

3. Associative entity

“An associative entity is an entity that associates the instances of one or more entity types and contains attributes that are peculiar to the relationship between these entity instances”.

The associative entity CERTIFICATE is represented with the “diamond relationship symbol enclosed with in the entity box.”



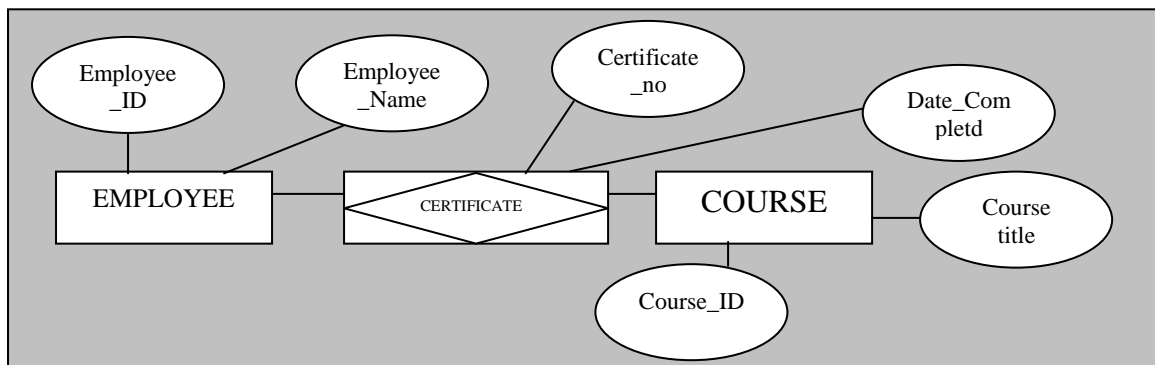
Example:

EMPLOYEE - Emp_ID, Emp_No,

COURSE - Course_ID, course_Title.

CERTIFICATE - Certificate_No, Date_Completed

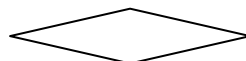
Now the relationship between these three entities by using a associative entity is as follows.



In the above example the entity **CERTIFICATE** is an **associative** entity because we can use this entity as an entity type or we can use as a relationship.

Relationship or Degrees of Relationship

A relationship type is a meaningful “association between entity types.” A relationship type is denoted by a “diamond symbol” containing the name of the relationship.



Degrees of a relationship

“The degrees of a relationship are the number of entity types that participate in that relationship.”

The three most common relationship degrees in ER models are

1. **Unary relationship** (Degree 1)

Definition: A relationship between the instances of a single entity type

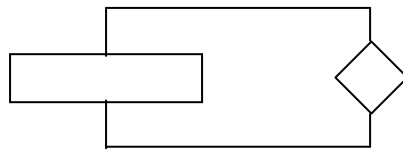
2. **Binary relationship** (Degree 2)

Definition: A relationship between the instances of two entity types

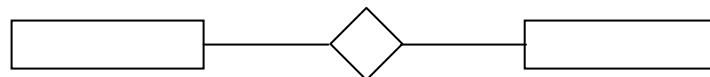
3. **Ternary relationship** (Degree 3)

Definition: A simultaneous relationship among the instances of three entity types.

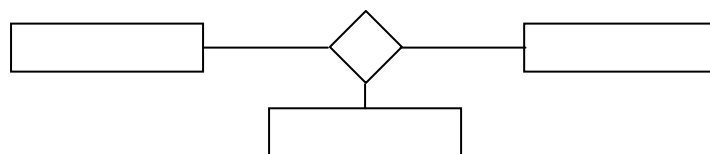
The following are the symbols of the relationship degree:



1) Degree 1 relationship (OR) unary relationship.



2) Binary relationship (OR) Degree 2 relationship



3) Ternary relationship (OR) degree 3 relationship.

Under *unary relationship* we have *one - one* and *one - to - many* relationships.

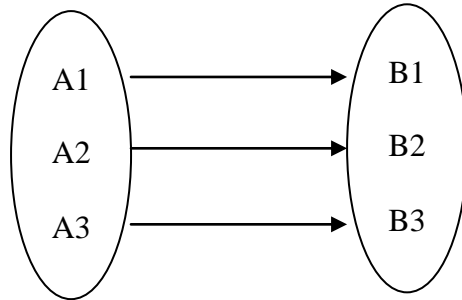
Under *binary relationship* we have

- i) one-to-one relationship
- ii) one-to-many relationship
- iii) many-to-one relationship

iv) many-to-many relationship

i) One-to-one relationship

An entity in A is associated with "At most one entity" in B and an entity in B is associated with "at most one entity" in A.

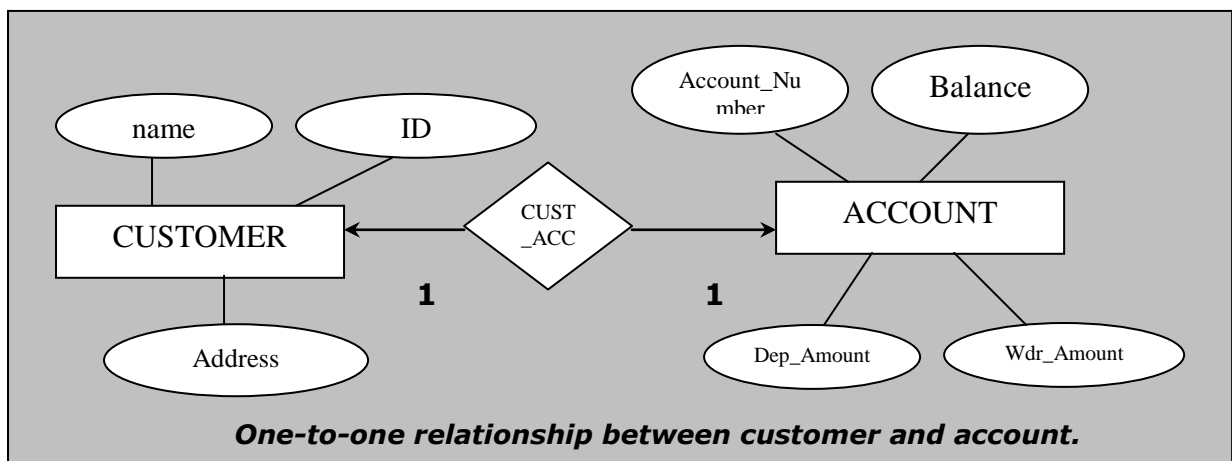


Example: Consider the following entity sets having the attributes.

CUSTOMER - Customer_Name, Customer_ID, Customer_Address

ACCOUNT - Account_Number, Deposited_Amount, Withdrawal_Amount, Balance

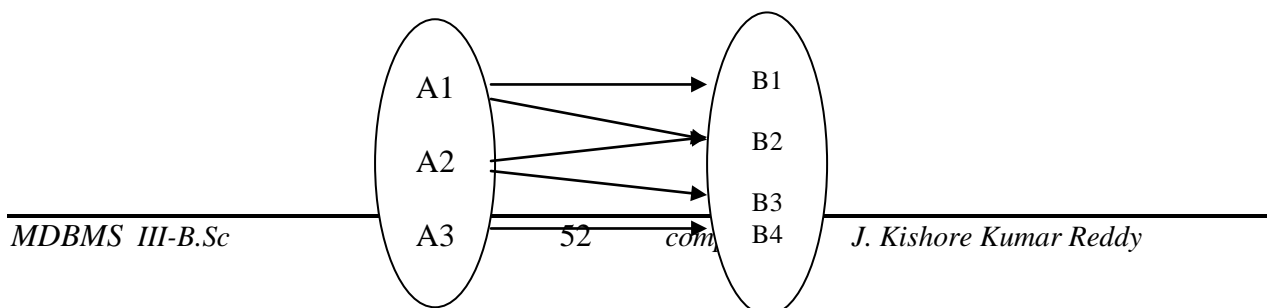
The relationship set Cust_Acct may be one - to - one relationship. (A customer has only one account in a bank).



DENOTED BY TWO DIRECTED LINES

ii) One-to-many relationship

An entity in A is associated with any number of entities in B. An entity in B, however in B, however with at most one entity A.



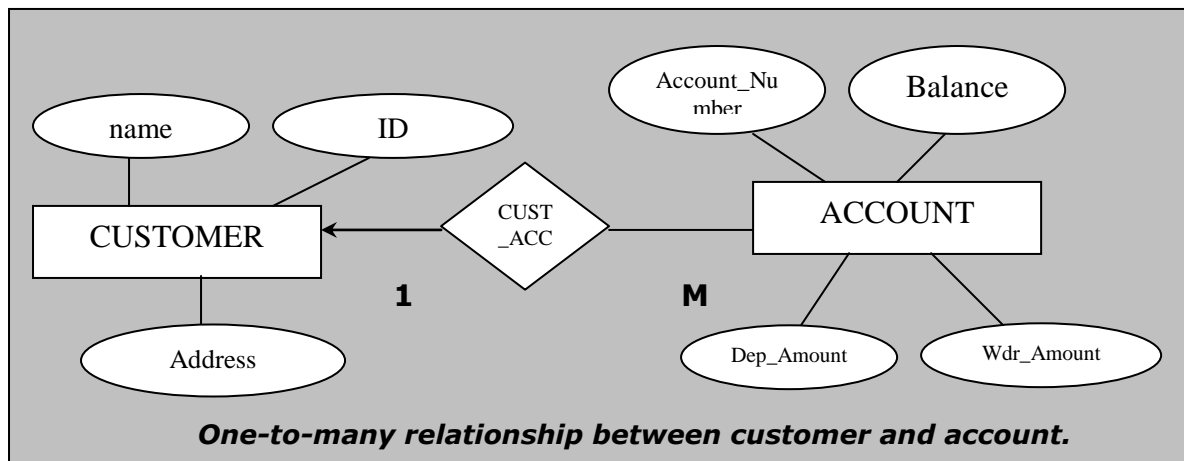
One - to - many relationship

Example: Consider the following entity sets having the attributes.

CUSTOMER - Customer_Name, Customer_ID, Customer_Address

ACCOUNT - Account_Number, Deposited_Amount, Withdrawal_Amount,
Balance

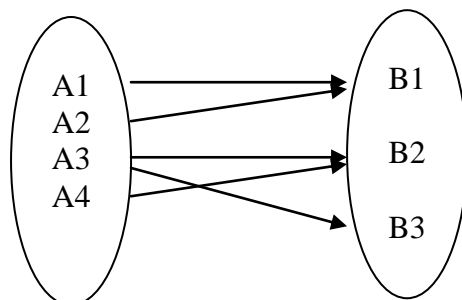
The relationship set Cust_Acct may be one - to - many relation-ship. (A customer may have many accounts in several banks).



DENOTED BY ONE DIRECTED LINE AND ONE UNDIRECTED LINE

iii) Many-to-One relationship:

An entity in "A" is associated with at most one entity in "B". An entity in "B", however can be associated with any number of entities in "A".



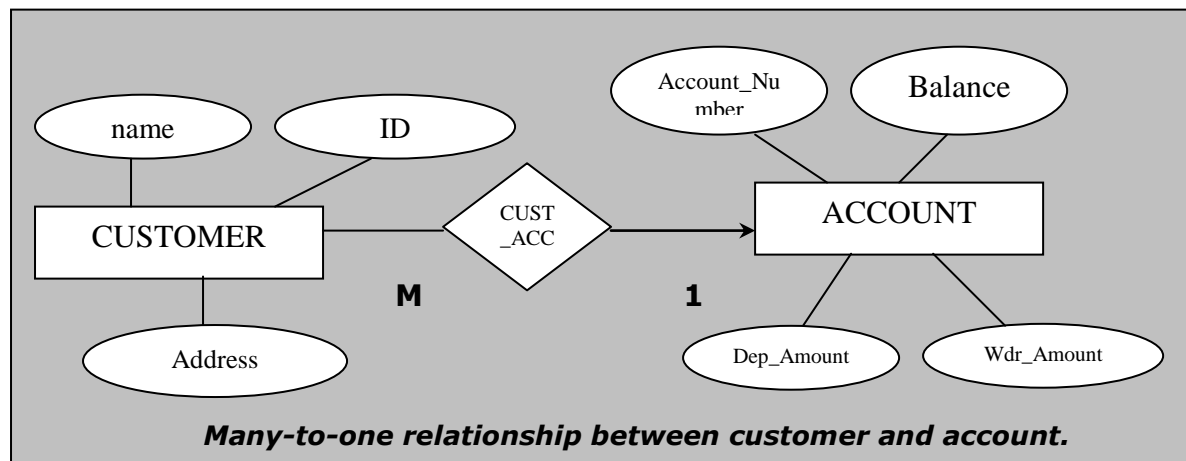
Many-to-one relationship

Example: Consider the following entity sets having the attributes.

CUSTOMER - Customer_Name, Customer_ID, Customer_Address

ACCOUNT - Account_Number, Deposited_Amount, Withdrawal_Amount,
Balance

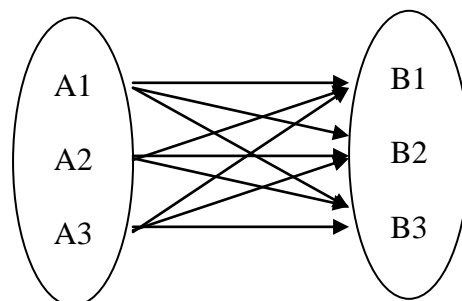
The relationship set Cust_Acct may be many - to - one relationship. (A customer may have many accounts in one bank {joint A/c}).



DENOTED BY ONE UN-DIRECTED LINE AND ONE DIRECTED LINE

iv) Many-to-many relationship

An entity in "A" is associated "with any number of entities" in "B" and an entity in "B" is associated with any number of entities in "A".

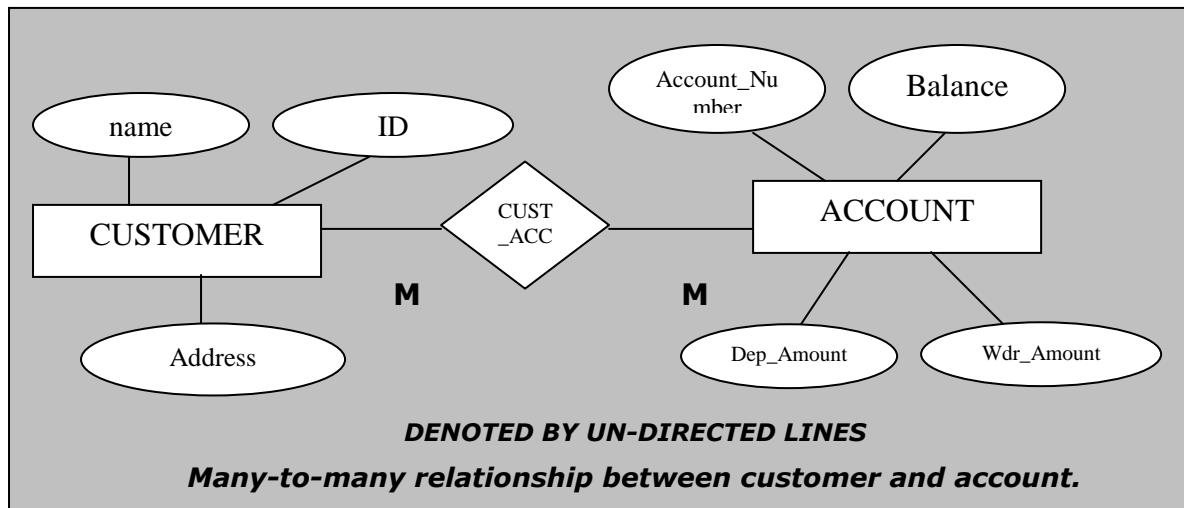


Example: Consider the following entity sets having the attributes.

CUSTOMER - Customer_Name, Customer_ID, Customer_Address

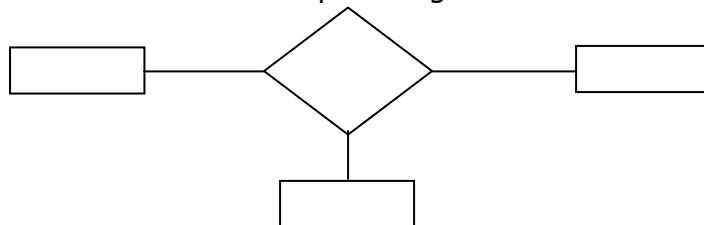
ACCOUNT - Account_Number, Deposited_Amount, Withdrawal_Amount,
Balance

The relationship set Cust_Acct may be many – to – many relation-ship. (Customers may have many accounts in many banks).



Ternary relationship

A simultaneous relationship among the instances of three entity types.



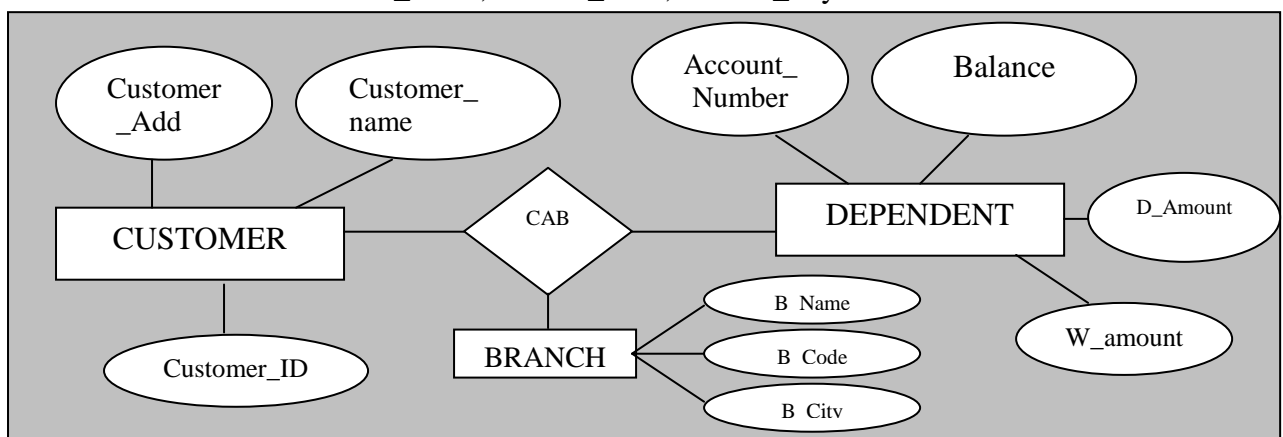
Ternary relationship

Example: Consider the following entity sets with possible attribute is

CUSTOMER: customer_ID, customer_add, customer_name

ACCOUNT: Account_number, Dep_Account, W_Amount, Balance

BRANCH: Branch_name, Branch_code, Branch_city.



Cardinality constrains

Cardinality constrain specifies the number of instances of one entity that can (or must) be associated with each instances of another entity.

The coordinately constrain is divided into two types. They are

1. Minimum Cardinality
2. Maximum Cardinality

Minimum Cardinality

The minimum Cardinality of a relationship is the “minimum number of instances of one entity that may be associated with each instances of another entity.

Maximum Cardinality

The maximum Cardinality of a relationship is the “maximum number of instances of one entity that may be associated with each instances of another entity.

DATABASE DESIGN CHALLENGES (CONFLICTING GOALS)



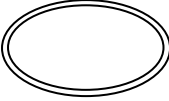

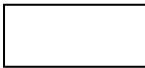
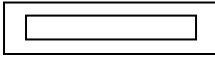

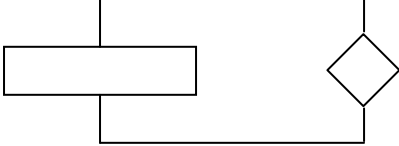
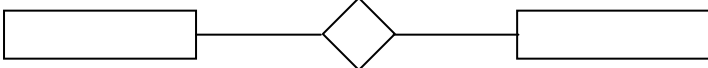
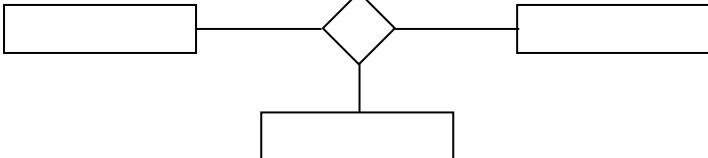
Database designers often must make design compromises that are triggered by conflicting goals, such as design standards (design elegance), processing speed, and information requirements.

- ❖ The database design must conform to design standards. Such standards have guided you in developing logical structures that minimize data redundancies, thereby minimizing the likelihood that destructive data anomalies will occur. Without design standards, it is nearly impossible to formulate a proper design proves, to evaluate an existing design, or to trace the likely logical impact of changes in design.
- ❖ In many organizations, particularly those generating large numbers of transactions, high processing speeds are often a top priority in database design. High processing speed means minimal access time, which may be achieved by minimizing the number and complexity of logically desirable relationships.
- ❖ The quest for timely information might be the focus of database design. Complex information requirements may dictate data transformations, and they may expand the number of entities and attribute4s within the design.

Therefore, the database may have to sacrifice some of its “clean” design structures and/or some of its high transaction speed to ensure maximum information generation.

- ❖ A design that meets all logical requirements and design conventions is an important goal
- ❖ Even as the designer focuses on the entities, attributes, relationships, and constraints, He/She should begin thinking about end-user requirements such as performance, security, shared access, and data integrity, the designer must consider processing requirements and verify that all update, retrieval, and deletion options are available. Finally, a design is of little value unless the end product is capable of delivering all specified query and reporting requirements.

Basic symbol used in ER-diagram

<u>SYMBOL</u>	<u>MEANING</u>
	Simple attribute
	Composite attribute
	Multivalued attribute
	Derived attribute
	Strong entity
	Weak entity
	Associative entity
	Unary Relationship
	Binary Relationship
	Ternary Relationship

EER Model (Enhanced Entity Relationship Model)

1. Introduction to EER Model
2. Super type
3. Sub type
4. Generalization
5. Specialization
6. Constrains in super type & subtype relation
7. Total specialization rule
8. Partial specialization rule
9. Disjoin rule
- 10.Overlap rule

Introduction to EER Model

The EER(**Enhanced Entity Relationship Model**) model is used to identify the model that has resulted from extending the original ER model with these new modeling constructs.

“The most important new modeling construct in incorporated in the EER model is super type/subtype relationships”.

Representing super type and subtype

Super type:

“A generic entity type that has a relationship with one or more subtypes.”

Example:

The following example shows a representation of EMPLOYEE super type with its three sub types using EER notation .consider the following entity sets with attributes.

1. Employee: number, name, address

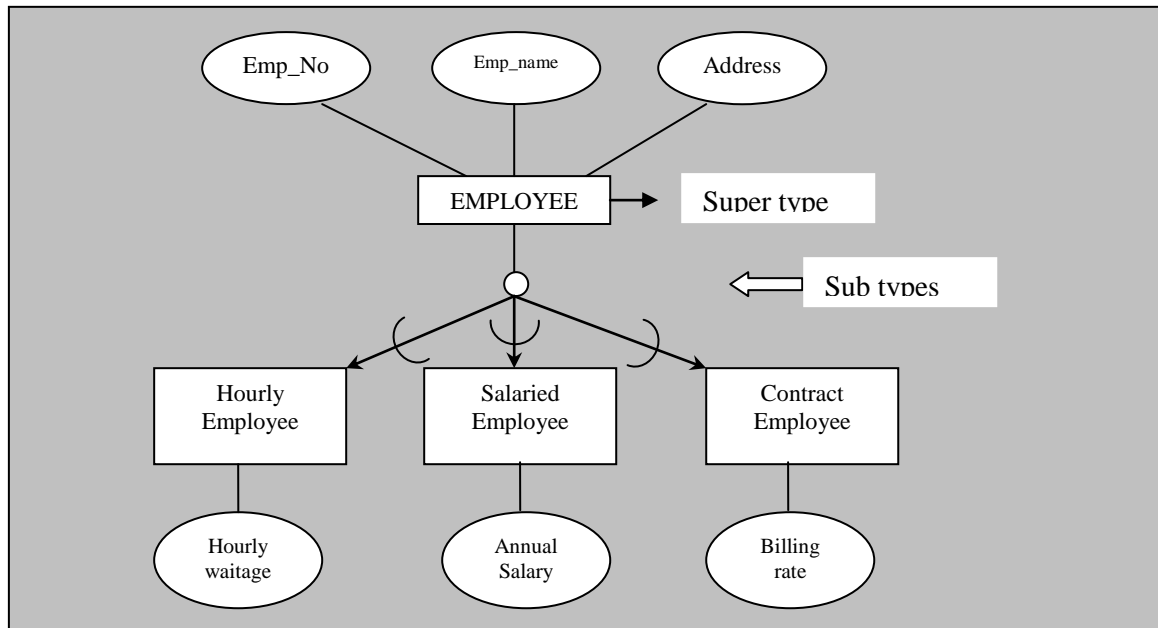
Suppose that an organization has three basic types of employees namely

- i) Hourly employee
 - ii) Salaried employee
 - iii) Contract employee

Some of important attribute for each of these type of employees are the following


- i) *Hourly employees:* Emp_no, Emp_name, Emp_add, Hourly_Rate
- ii) *Salaried employees:* Emp_no, Emp_name, Emp_add, Annual_Salary
- iii) *Contract employees:* Emp_no, Emp_name, Emp_add, Billing_Rate

In all above employee types several, attributes in common namely "Emp_no, Emp_name, Emp_add. In addition, each type has one or more attributes distinct from the attributes of the other types the following is the EER diagram with super type and sub types.



The above figure shows a representation of the EMPLOYEE super type with its three subtypes, using EER relation. Attribute shared by all employees are associated with the EMPLOYEE entity type. Attributes that are peculiar to each subtype are included with subtype only.

Super Type

The super type is connected with a line to a circle,  which in turn is connected by a line to each subtype that has been defined.

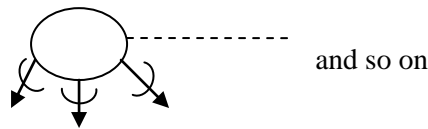
Subtype

A sub grouping of the entities is an entity type that is meaningful to the organization and that "shares common attributes or relationships, distinct from other subgroups."

Example: STUDENT is an entity type in a university. Two sub types of STUDENT are GRADUATE STUDENT and UNDERGRADUATE STUDENT. In this example STUDENT is super type where as GRADUATE STUDENT and UNDERGRADUATE STUDENT is two subtypes.

Basic notation

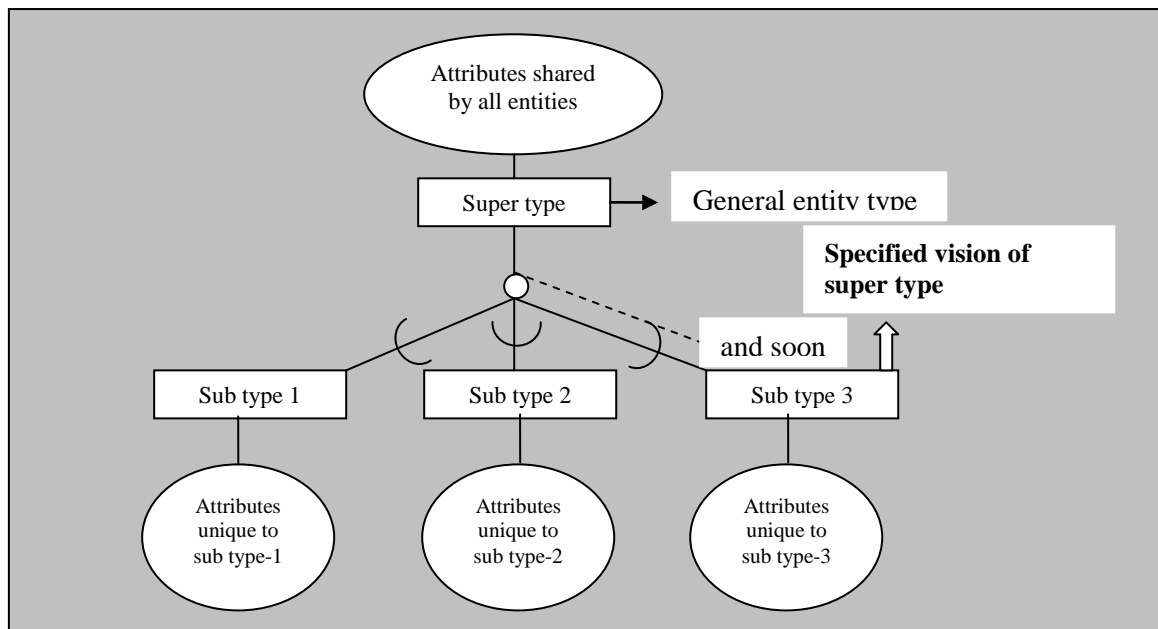
The U-shaped symbol on each line connecting a subtype to the circle indicates that the subtype is a subset of the super type.



Basic concepts and notation per super type and sub types relationship

The basic notation that we use for sub type/ super type relationships is shown in the following figure.

Basic concepts and notation per sub type and super type relationship:



In the above figure the super type is connected with a line to a circle which in turn is connected by a line to each sub type that has been defined.

The "U" shaped symbol on each line connecting a subtype to the circle indicates that the subtype is a subset of the super type.

"Attributes that are shared by all entries" are associated with the **super type**. **Attributes that are unique** to a particular **"subtype"** are associated with that sub type."

When to use super type/sub type relationships:

Use the super type/sub type relationships in the following situations.

- 1) There are attributes that apply to some (but not all) of the instance of an entity type.
- 2) The instances of a subtype participate in a relationship unique to that sub type.

Representing generalization and specialization:

There are two processes namely

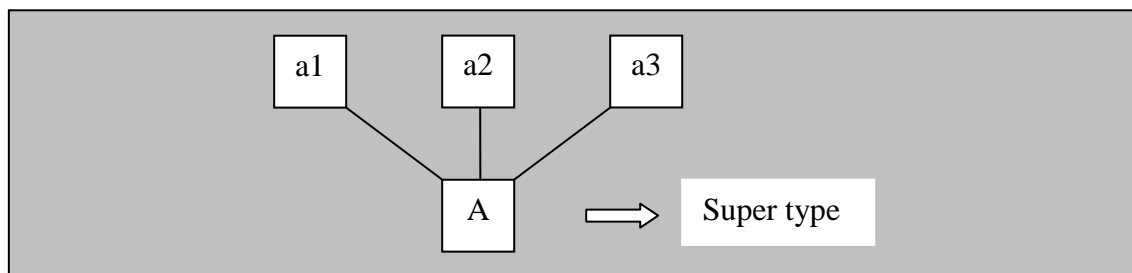
1. Generalization
2. Specialization

-That are serve as mental models in developing super/sub types relationships.

1. Generalization

Definition: "The process of defining a more general entity type from a set of more specialized entity types."

"The generalization is a bottom up process," (deriving super type from its subtypes.)



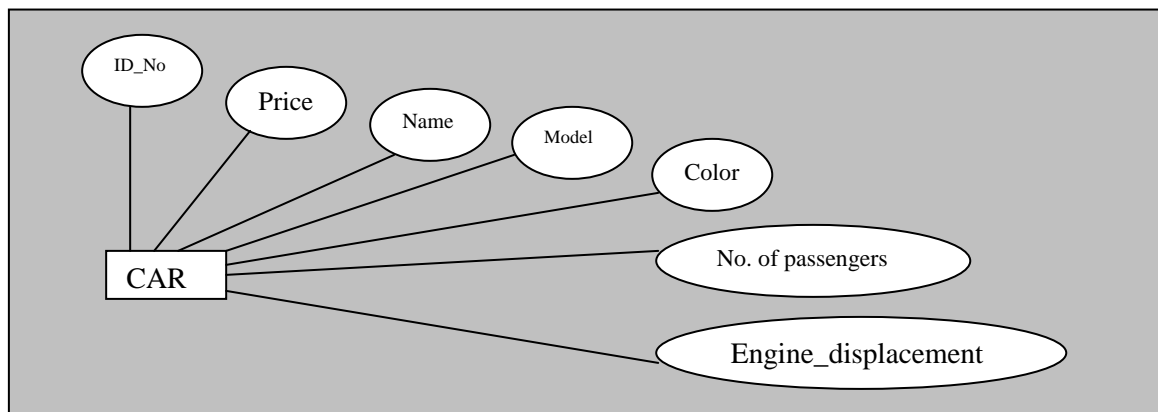
Example: consider three entity types having the attributes is as follows

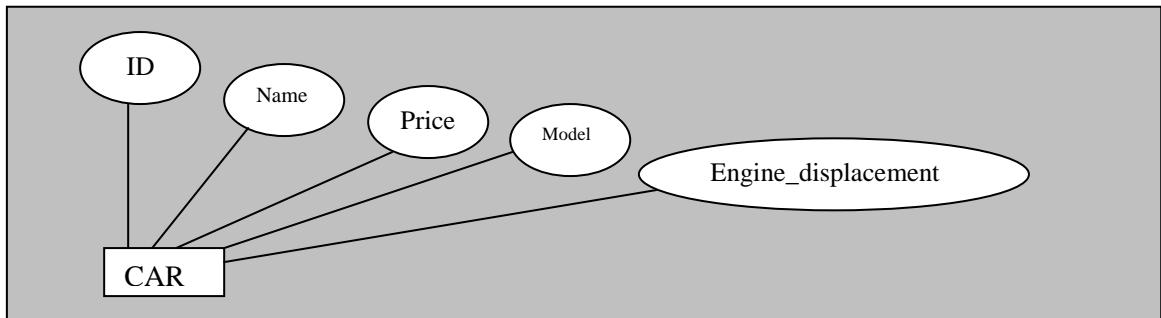
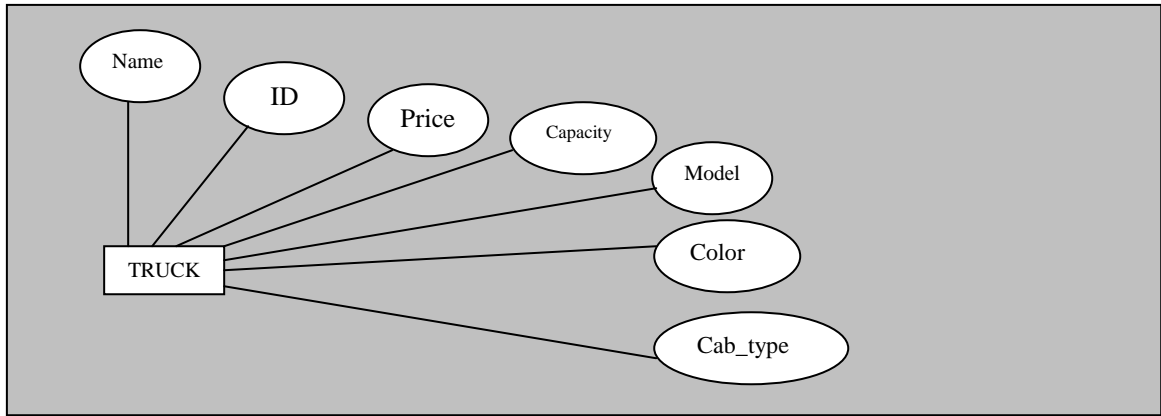
CAR: Vehicle_ID, Price, No_of_Passengers, Vehicle_Name, Engine_Displacement, Model, Make, Color

TRUCK: Vehicle_ID, Vehicle_Name, Capacity, Price, cab_type, Model, Engine_displacement, Make, Color

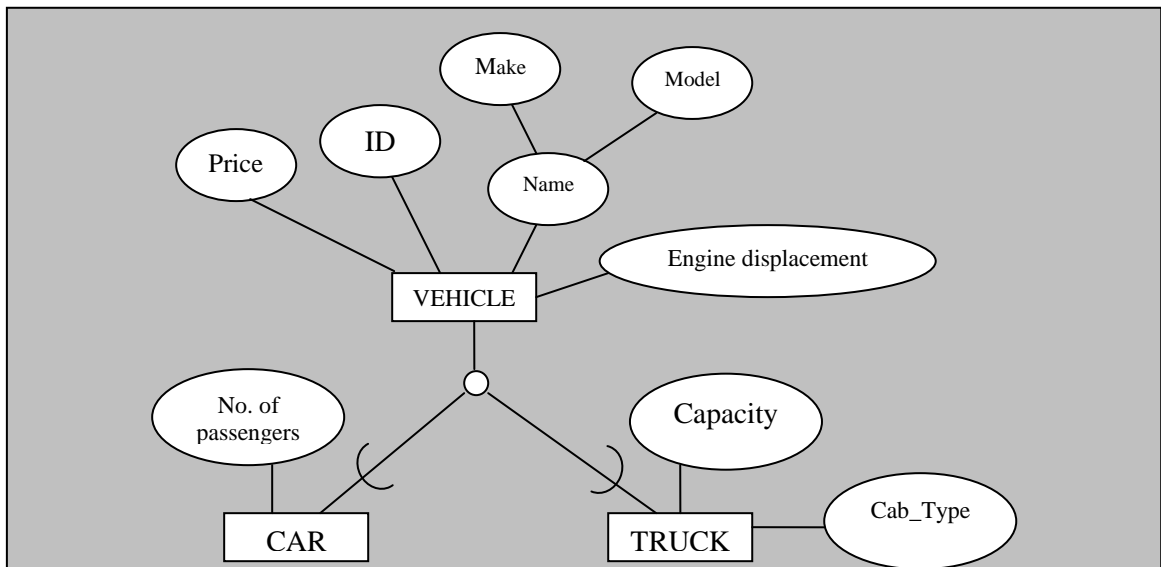
MOTORCYCLE: Vehicle_ID, Vehicle_Name, Make, Price, Model, Engine_displacement

Three entity types CAR, TRUCK, MOTORCYCLE At the stage the data modeler intends to represent these separately on an ER diagram in fig A..





This more general entity type (named VEHICLE) together with the resulting super type/subtype relationships is shown in figure(B).



Generalization to vehicle super type

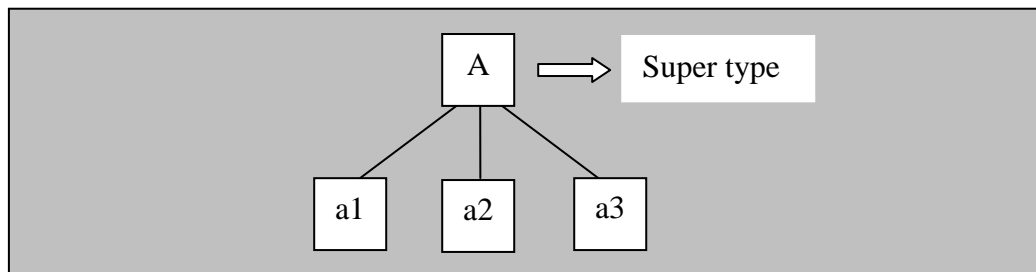
The entity CAR has the specific attribute number of passengers, while TRUCK has two specific attributes cab_type, capacity. Thus generalization has allowed us to group entity types along with their common attributes and at the same time preserves specific attributes that are peculiar to each sub type.

“Notice that the entity type MOTORCYCLE is not included in the relationship because it does not satisfy the conditions for sub type.”

We will notice that only attributes of MOTORCYCLE are those that are common to all vehicles. There are no specific attributes to MOTORCYCLE.

Specialization: *Generalization is bottom up process. Specialization is a top down process the direct reverse of generalization.*

Definition: *“Specialization is the process of defining one or more subtypes of the super type and forming sub type/super type relationships”.*

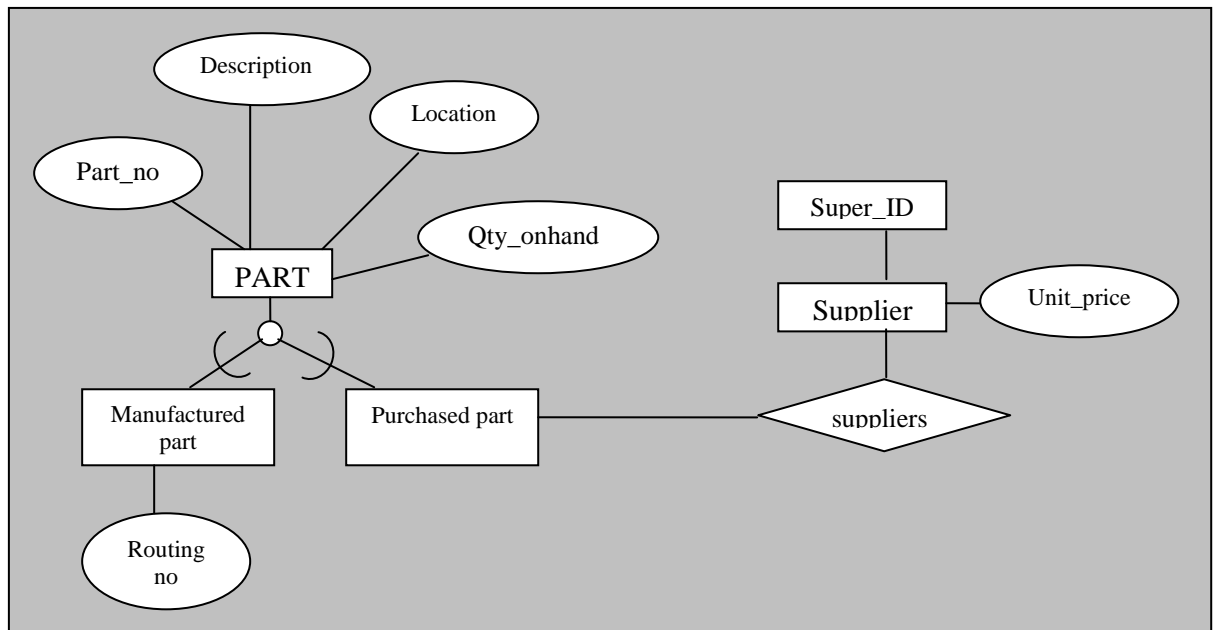


Example: suppose an entity type PART having the attributes.

PART: part_Number, Description, Location, Quantity_on_Hand, Unit_Price, Supplier_ID, Routing_Number.

There are two possible sources for parts: some are manufactured internally, while others are purchased from outside supplies. Further we discovered that some parts are obtained from both sources.

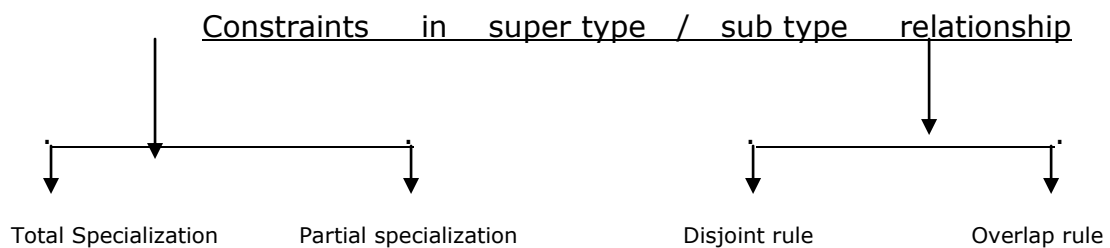
These factors suggest that PART should be specialized by defining the sub types MANUFACTURED PART and PURCHASED PART.



Difference between Generalization and Specialization

Generalization	Specialization
1. Generalization is valuable technique for developing super type/sub type relationship.	1. Specialization is also valuable technique for developing sub type/super type relationship.
2. Deriving super type from sub type.	2. Deriving sub types from super type.
3. Bottom up process	3. Top down process.

Specifying constraints in super type/sub type relationships:

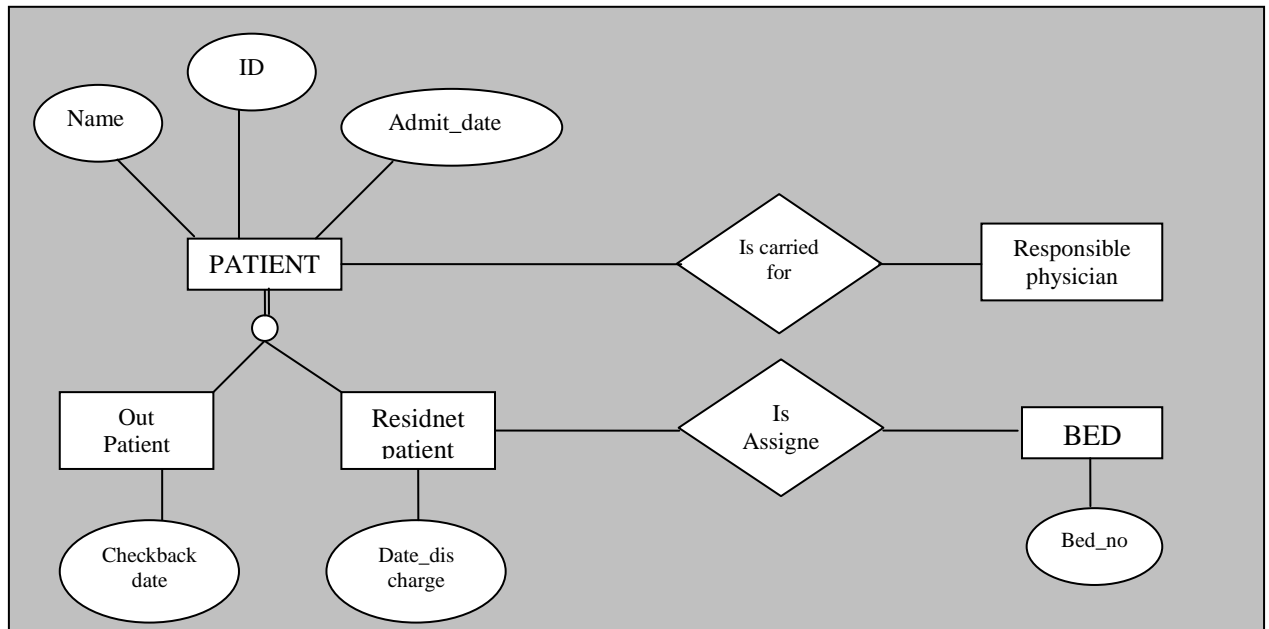


1.Total specialization:

Total specialization rule specifies that each entity instance of the super type must be a member of some sub type in the relationship.

Example: consider the example of PATIENT and introduces the notation for total specialization.

In this example the business rule is the following: A patient must be either out patient or a resident patient. There are no other types of patients in this hospital. Total specialization is indicated by the double line.



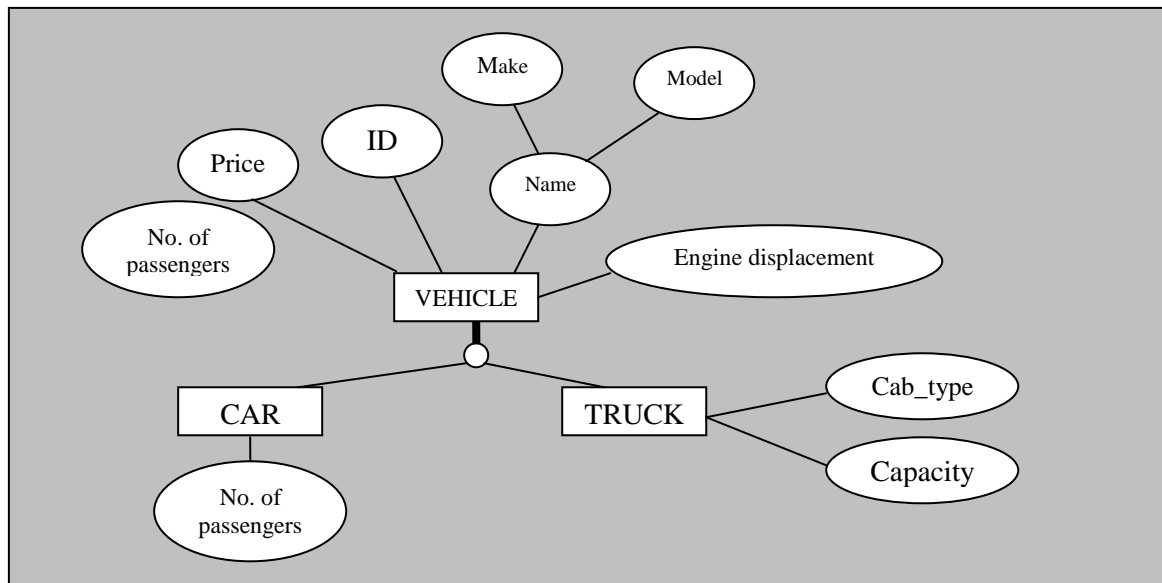
Total specialization rule to patient

2. Partial specialization rule

The partial specialization rule specifies that an entity instance of the super type is allowed not to be belonging to any sub type. The partial specialization is specified by the single line.

Example: consider the example of VEHICLE and is sub type CAR and TRUCK, MOTORCYCLE.

In this example MOTORCYCLE is a type of VEHICLE but that is not represented as a sub type in the data model.



Partial specialization rule to vehicle

3. Disjoint rule

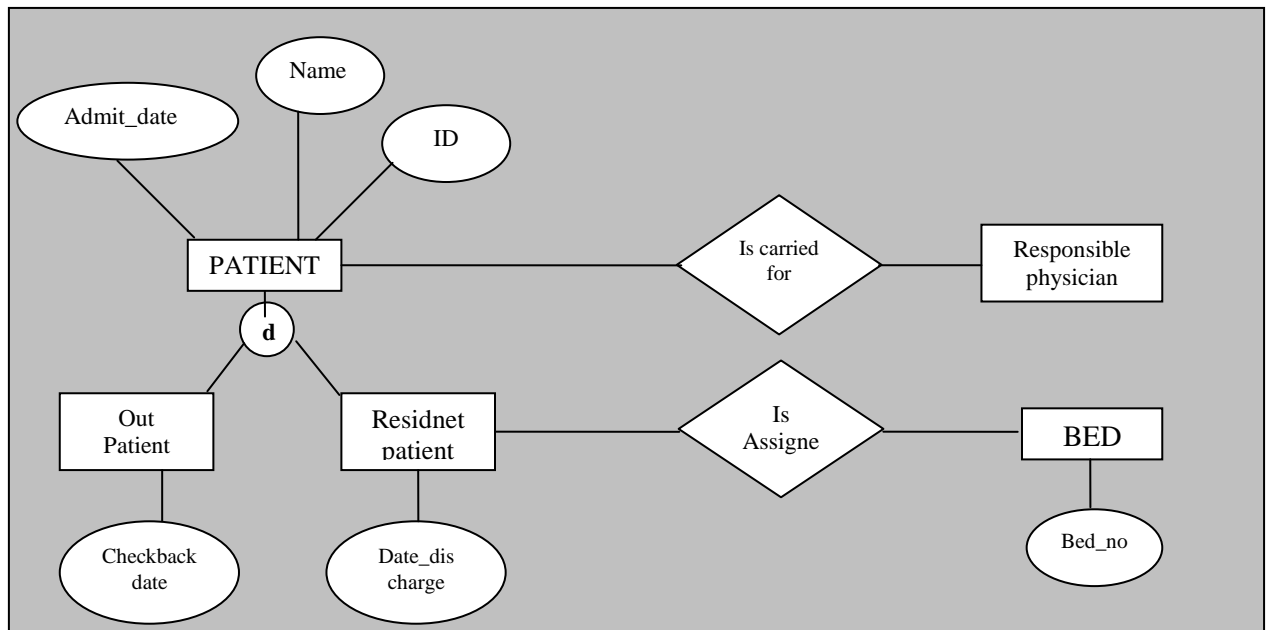
The disjoint rule specifies that if an entity instance of the super type is a member of one sub type, it cannot simultaneously be a member of any other sub types.

The disjoint rule as specified by the letter 'd' in the circle joining the super type and its sub types

Example: consider the example of patient.

The business rule in this case in this case is the following:

At any given time, a patient must be either an out patient or a resident patient but cannot be both.



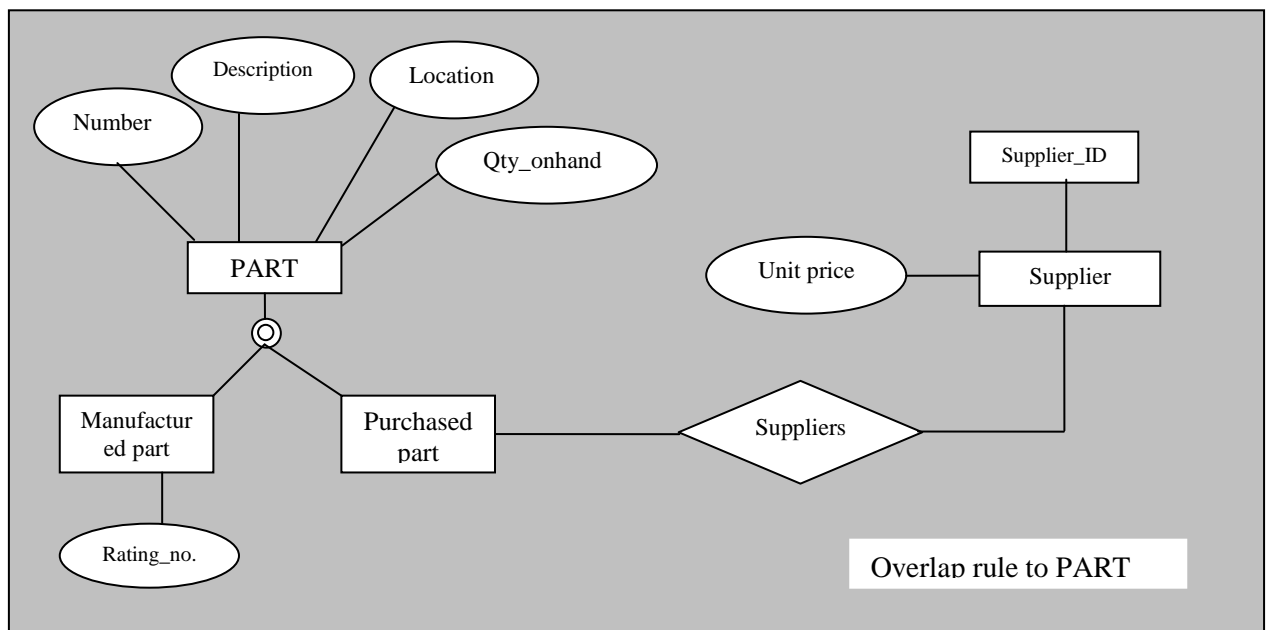
Disjoint rule to patient

4. Overlap rule

The overlap rule specifies that an entity instance can simultaneously be a member of two or more sub types.

The overlap rule is specified by placing the letter 'o' in the circle.

Example: consider the example entity type PART with its sub types MANUFACTURED PART and PURCHASED PART. In this example some parts are both manufactured and purchased.



Overlap rule to PART

Entity clustering

E-R diagrams can become large and complex including hundreds of entities. Many users and managers do not need to see all the entities, relationships and attributes to understand the part of the database with which they

Entity clustering is a useful way to represent a data model for a large and complex organization.

Definition:

"An entity cluster is a set of one or more entity types and associated relationships grouped into a single abstract entity type."

Because an entity cluster behaves like an entity type, entity clusters and entity types can be further grouped to form a higher level entity cluster.

Entity clustering is a hierarchical decomposition a macro level view of the data model into finer and finer views, eventually resulting in the full, detailed data model.

NORMALIZATION

Definition and need of Normalization

“Normalization is the process of **decomposing** relations with anomalies to **produce smaller, well structured relations.**”

Normalization is primarily a tool to validate and improve a logical design, so that it satisfies certain constraints that “**avoid unnecessary duplication of data.**”

Steps in Normalization

BASIC NORMAL FORMS

- **First Normal Form (1NF):** Any multi-valued attributes have been removed, so there is a single (possibly null) at the intersection of each row and column of the table.
- **Second Normal Form (2NF):** Any partial functional dependencies have been removed.
- **Third Normal Form (3NF):** Any transitive dependencies have been removed.

ADVANCED NORMAL FORMS

- **Boyce/Codd Normal Form (BCNF):** Any remaining anomalies that result from functional dependencies have been removed.
- **Fourth Normal Form (4NF):** Any multi-valued dependencies have been removed.
- **Fifth Normal Form (5NF):** Any remaining anomalies have been removed.

Functional Dependencies

“**Normalization is based on the analysis of functional dependences.**”

Def: A Functional dependency is a constraint between two attributes or two sets of attributes.

“The functional dependency” of B on A is “Represented by an arrow”, as follows : $A \rightarrow B$

Example:

Consider the relation

EMP_COURSE (Emp_ID, Course_Title, Date_Completed)

We represent the functional dependency in this relation as follows.

Emp_ID, Course_Title → Date_Completed.

Common examples of functional dependencies are the following.

1. SSN → Name, Address, Birth date

(A person Name, Address, Birth Date is functionally dependent on that person's social security number)

2. VIN → Make, Model, Color

(The make model and color of a vehicle are functionally dependent on the Vehicle identification number)

Determinants

"The attributes on the left-hand side of the arrow in functional dependencies called a determinant"

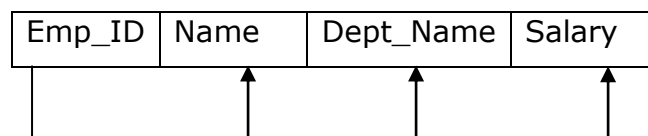
Example: SSN, VIN are called determinants

EMP_ID, Course_Title are also called determinants.

Representation of functional dependencies

We represent the functional dependencies for a relation using the notation shown in the following figure (a).

Example 1: Figure (a) shows the representation for EMPLOYEE 1



Functional Dependencies if Figure(a)

The horizontal line in the figure portrays the functional dependencies.

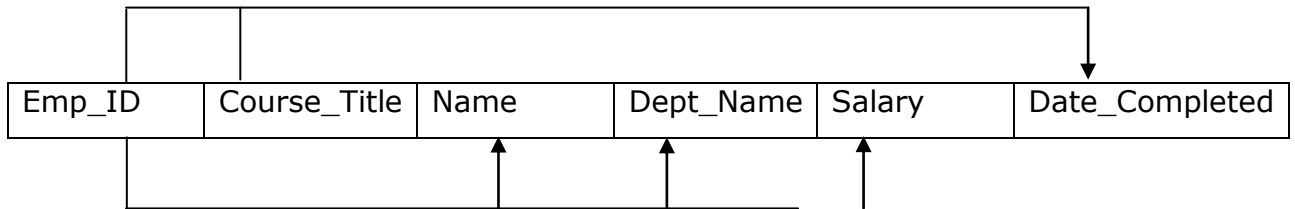
A vertical line drops from the primary key (Emp_ID) and connects to this line. Vertical arrows then point to each of the non-key attributes that are functionally dependent on the primary key.

Example 2: Consider the example EMPLOYEE 2

There are two functional dependencies in this relation.

1. Emp_ID → Name, Dept_Name, Salary
2. Emp_ID, Course_Title, → Date_Completed

The primary key of EMPLOYEE 2 is a composite key neither Emp_ID nor Course_Title uniquely identifies a row in this relation.



Functional dependencies in EMPLOYEE 2

BASIC NORMAL FORMS (1NF, 2NF and 3NF):

First Normal Form (1NF):

A relation is in "First Normal Form" (1NF) it satisfies

- All of the key attributes are defined
- There are no repeating groups in the table. In other words ,each row/column intersection contains one and only value(atomic or single value) ,not a set of values(mutivalued attributes).
- All attributes are dependent on the primary key.

Example: Table 1: Un-normalized Data

PROJ_NUM	PROJ_NAME	EMP_NUM	EMP_NAME	JOB_CLAS	CHG_HOURS	HOURS
15	Evergreen	103	June E.Arboogh	Elect Engineer	84.5	23.8
		101	john	Data base designer	105	19.4
		105	Alice	Data base designer	105	35.8
		106	William	Programmer	35.75	12.6
		102	David	System Analyst	96.75	23.8
18	Amber Wave	114	Annelise	Application Designer	48.1	24.6
		118	James	General Support	18.36	45.3
		104	Anne	System Analyst	96.75	32.4
		112	Darlene	DSS analyst	45.95	44
		105	Alice	Data base designer	105	64.7
22	Rolling tide	104	Anne.k	System Analyst	96.75	48.4
		113	Delbert	Application Designer	48.1	23.6
		111	Geoff	Clerical support	26.87	22
		106	William	Programmer	35.75	12.8
		105	Alice	Data base designer	105	64.7
25	Starflight	107	Maria	Programmer	35.75	24.6
		115	Travis	System Analyst	96.75	45.8
		101	John	Data base designer	105	56.3
		114	Anne jones	Application Designer	48.1	33.1
		108	Ralph	System Analyst	96.75	23.6
		118	James	General Support	18.36	30.5

		112	Darlene	DSS analyst	45.9	41.4
--	--	-----	---------	-------------	------	------

Table 2: **Converting Un-normalized Data into First Normal Form (1NF):**

PROJ_NUM	PROJ_NAME	EMP_NUM	EMP_NAME	JOB_CLAS	CHG_HOURS	HOURS
15	Evergreen	103	June E.Arboh	Elect Engineer	84.5	23.8
15	Evergreen	101	john	Data base designer	105	19.4
15	Evergreen	105	Alice	Data base designer	105	35.8
15	Evergreen	106	William	Programmer	35.75	12.6
15	Evergreen	102	David	System Analyst	96.75	23.8
18	Amber Wave	114	Annelise	Application Designer	48.1	24.6
18	Amber Wave	118	James	General Support	18.36	45.3
18	Amber Wave	104	Anne	System Analyst	96.75	32.4
18	Amber Wave	112	Darlene	DSS analyst	45.95	44
22	Rolling tide	105	Alice	Data base designer	105	64.7
22	Rolling tide	104	Anne.k	System Analyst	96.75	48.4
22	Rolling tide	113	Delbert	Application Designer	48.1	23.6
22	Rolling tide	111	Geoff	Clerical support	26.87	22
22	Rolling tide	106	William	Programmer	35.75	12.8
25	Starflight	107	Maria	Programmer	35.75	24.6
25	Starflight	115	Travis	System Analyst	96.75	45.8
25	Starflight	101	John	Data base designer	105	56.3
25	Starflight	114	Anne jones	Application Designer	48.1	33.1
25	Starflight	108	Ralph	System Analyst	96.75	23.6
25	Starflight	118	James	General Support	18.36	30.5
25	Starflight	112	Darlene	DSS analyst	45.9	41.4

MULTI VALUED ATTRIBUTES ELIMINATED & CONVERTED TO 1NF

Table 2: A TABLE IS IN FIRST NORMAL FORM

Second Normal Form (2 NF):

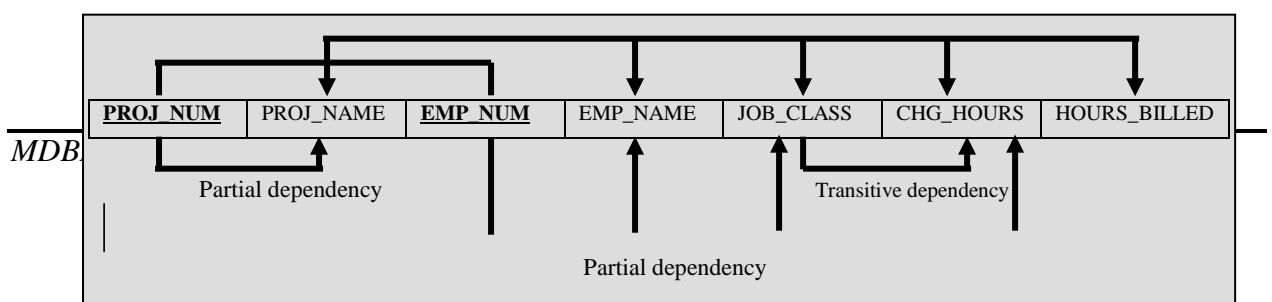
A relation is in "Second normal form (2NF)" it is in **first normal form and every non key attribute is fully functionally dependent on the primary key**".

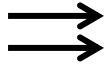
A table is in second normal form(2 NF) when

- It is in first normal form(1NF)
- and
- It includes no partial dependences; that is, no attribute is dependent on only a portion of the primary key.

(But it is still possible for a table in 2NF to exhibit transitive dependency ; that is, one or more attributes may be functionally dependent on non key attributes)

Figure (a): **DEPENDENCY DIAGRAM**



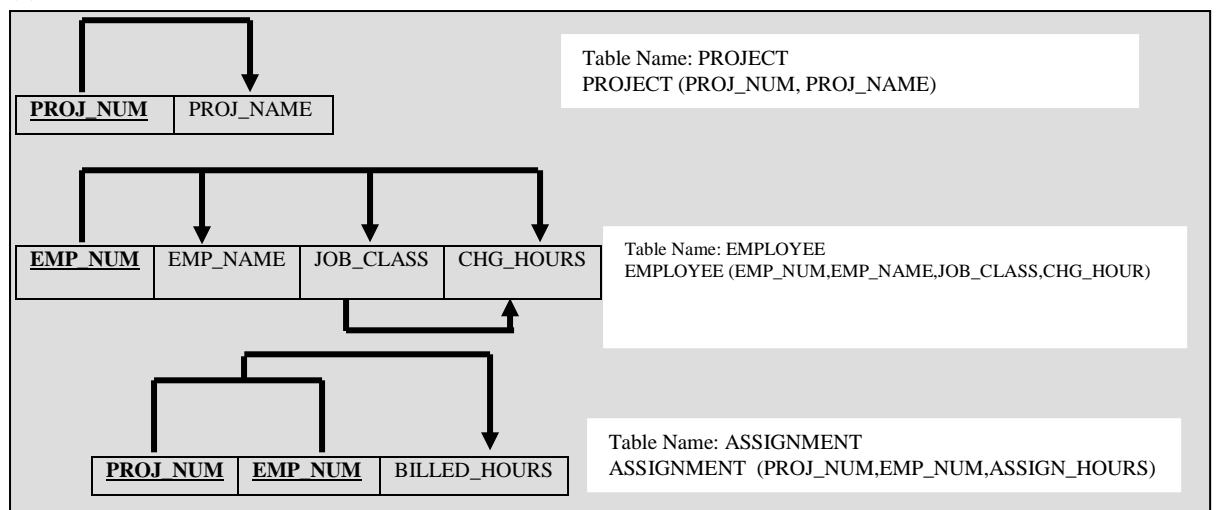


In above dependency diagram two types dependencies exist.

Partial dependencies:

We need to know only the PROJ_NUM to determine the PROJ_NAME is dependent on only part of the primary key. And you need to know only the EMP_NUM to find the EMP_NAME, the JOB_CLASS and the CHG_HOUR. A dependency based on only a part of a composite primary key is called primary key is called a **Partial dependency**.

Figure (b):



SECOND NORMAL FORM 2NF CONVERSION

SECOND NORMAL FORM 2NF CONVERSION RESULTS

PROJ_NUM	PROJ_NAME	EMP_NUM	EMP_NAME	JOB_CLAS	CHG_HOURS	HOURS	TO 1NF
15	Evergreen	103	June	Elect Engineer	84.5	23.8	

			E.Arboogh			
15	Evergreen	101	john	Data base designer	105	19.4
15	Evergreen	105	Alice	Data base designer	105	35.8
15	Evergreen	106	William	Programmer	35.75	12.6
15	Evergreen	102	David	System Analyst	96.75	23.8
18	Amber Wave	114	Annelise	Application Designer	48.1	24.6
18	Amber Wave	118	James	General Support	18.36	45.3
18	Amber Wave	104	Anne	System Analyst	96.75	32.4
18	Amber Wave	112	Darlene	DSS analyst	45.95	44
22	Rolling tide	105	Alice	Data base designer	105	64.7
22	Rolling tide	104	Anne.k	System Analyst	96.75	48.4
22	Rolling tide	113	Delbert	Application Designer	48.1	23.6
22	Rolling tide	111	Geoff	Clerical support	26.87	22
22	Rolling tide	106	William	Programmer	35.75	12.8
25	Starflight	107	Maria	Programmer	35.75	24.6
25	Starflight	115	Travis	System Analyst	96.75	45.8
25	Starflight	101	John	Data base designer	105	56.3
25	Starflight	114	Anne jones	Application Designer	48.1	33.1
25	Starflight	108	Ralph	System Analyst	96.75	23.6
25	Starflight	118	James	General Support	18.36	30.5
25	Starflight	112	Darlene	DSS analyst	45.9	41.4

Table 1: A TABLE IS IN FIRST NORMAL FORM

The above table 1 is in 1NF but not 2NF now the 1NF Table is **decomposed into 3 tables namely PROJECT, EMPLOYEE, ASSIGNMENT is as follows then it satisfies the 2NF.**

PROJ_NUM	PROJ_NAME
15	Evergreen
18	Amber Wave
22	Rolling tide
25	Starflight

Table 2: 2NF:
PROJECT(PROJ_NUM, PROJ_NAME)

EMP_NUM	EMP_NAME	JOB_CLAS	CHG_HOURS
103	June E.Arboogh	Elect Engineer	84.5
101	john	Data base designer	105.0
105	Alice	Data base designer	105.0
106	William	Programmer	35.75
102	David	System Analyst	96.75
114	Annelise	Application Designer	48.10
118	James	General Support	18.36
104	Anne	System Analyst	96.75
112	Darlene	DSS analyst	45.9
113	Delbert	Application Designer	48.1
111	Geoff	Clerical support	26.87
107	Maria	Programmer	35.75

115	Travis	System Analyst	96.75
108	Ralph	System Analyst	96.75

Table 3: 2NF:

EMPLOYEE(EMP_NUM, EMP_NAME, JOB_CLASS, CHG_HOURS)

PROJ_NUM	EMP_NUM	HOURS_Billed
15	103	23.8
15	101	19.4
15	105	35.8
15	106	12.6
15	102	23.8
18	114	24.6
18	118	45.3
18	104	32.4
18	112	44.0
22	105	64.7
22	104	48.4
22	113	23.6
22	111	22
22	106	12.8
25	107	24.6
25	115	45.8
25	101	56.3
25	114	33.1
25	108	23.6
25	118	30.5
25	112	41.4

Table 4: 2NF

ASSIGNMENT(PROJ_NUM, EMP_NUM, ASSIGN_HOURS_BILLED)

Third Normal Form (3 NF):

"A Relation is in third normal form (3 NF) when:

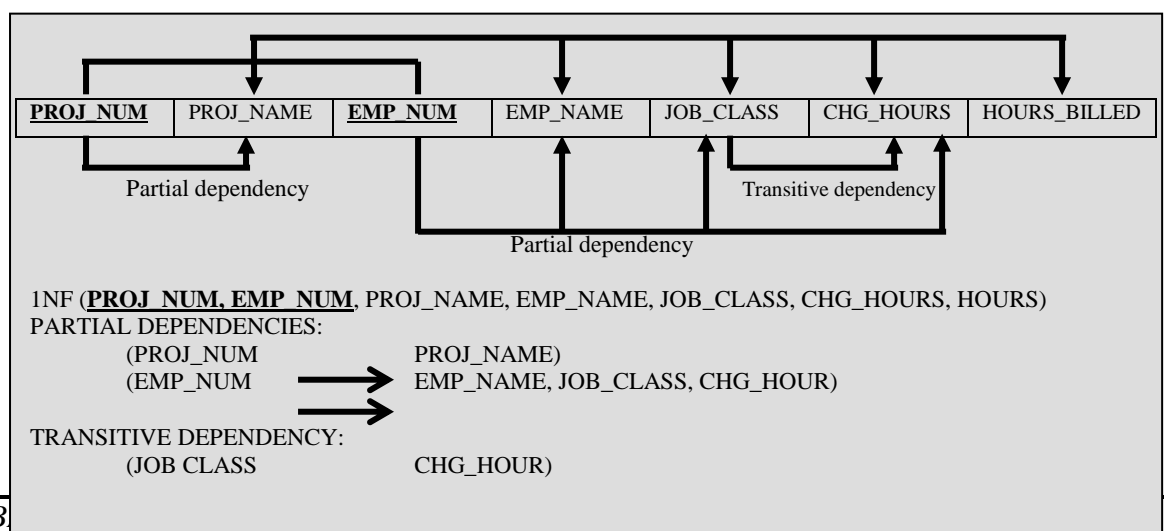
It is in second normal form

and

It contains no transitive dependencies exist".

Transitive dependency:

A transitive dependency in a relation is a functional dependency between two (or more) non key attributes.

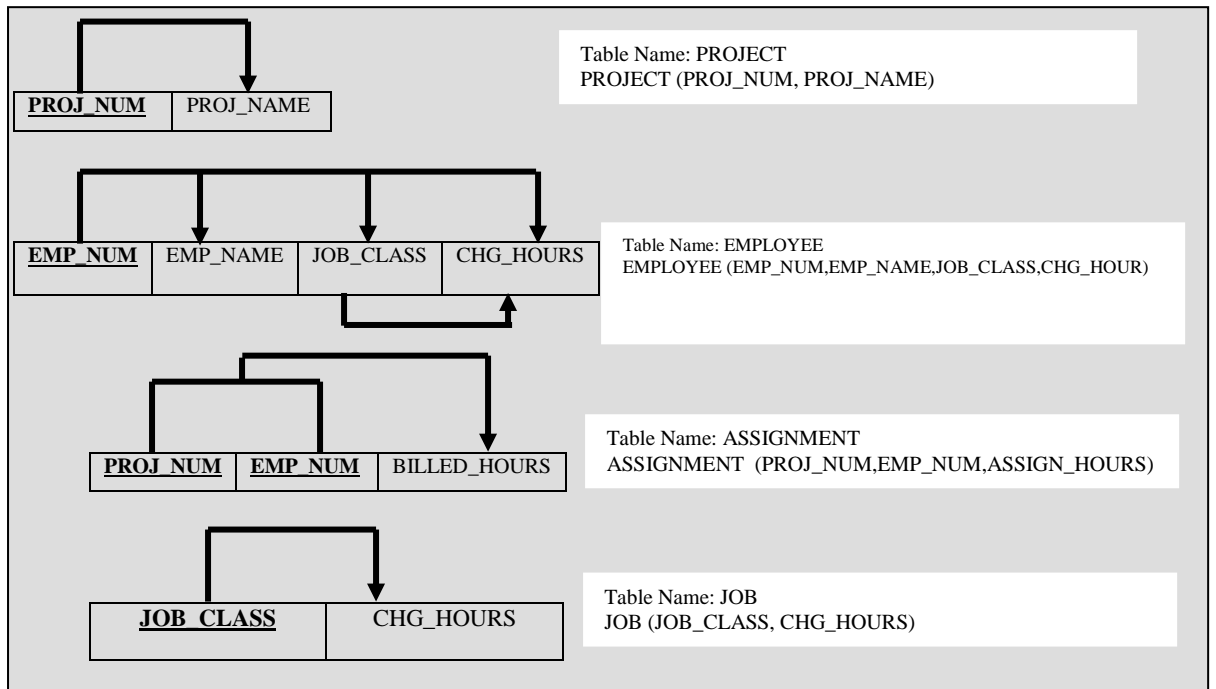




Dependency Diagram

In above dependency diagram CHG_HOURS is functionally dependent on JOB_CLASS and JOB_CLASS is already depended on EMP_NUMBER. So this type of dependencies is known as transitive dependence. To eliminate transitive dependence the table decompose into 4 tables namely:

- PROJECT (PROJ_NUM, PROJ_NAME)
- EMPLOYEE (EMP_NUM,EMP_NAME,JOB_CLASS,CHG_HOUR)
- ASSIGNMENT (PROJ_NUM,EMP_NUM,ASSIGN_HOURS)
- JOB (JOB_CLASS, CHG_HOURS)



THIRD NORMAL FORM 3NF CONVERSION

THIRD NORMAL FORM 3NF CONVERSION RESULTS

PROJ_NUM	PROJ_NAME
15	Evergreen
18	Amber Wave
22	Rolling tide

25	Starflight
----	------------

Table 1: 3NF:
PROJECT(PROJ_NUM, PROJ_NAME)

EMP_NUM	EMP_NAME	JOB_CLAS	CHG_HOURS
103	June E.Arboogh	Elect Engineer	84.5
101	john	Data base designer	105.0
105	Alice	Data base designer	105.0
106	William	Programmer	35.75
102	David	System Analyst	96.75
114	Annelise	Application Designer	48.10
118	James	General Support	18.36
104	Anne	System Analyst	96.75
112	Darlene	DSS analyst	45.9
113	Delbert	Application Designer	48.1
111	Geoff	Clerical support	26.87
107	Maria	Programmer	35.75
115	Travis	System Analyst	96.75
108	Ralph	System Analyst	96.75

Table 2: 3NF: **EMPLOYEE** (EMP_NUM, EMP_NAME, JOB_CLASS, CHG_HOURS)

PROJ_NUM	EMP_NUM	HOURS_Billed
15	103	23.8
15	101	19.4
15	105	35.8
15	106	12.6
15	102	23.8
18	114	24.6
18	118	45.3
18	104	32.4
18	112	44.0
22	105	64.7
22	104	48.4
22	113	23.6
22	111	22
22	106	12.8
25	107	24.6
25	115	45.8
25	101	56.3
25	114	33.1
25	108	23.6
25	118	30.5
25	112	41.4

Table 3: 3NF
ASSIGNMENT (PROJ_NUM, EMP_NUM, ASSIGN_HOURS_BILLED)

JOB_CLASS	CHG_HOURS
Elect Engineer	84.5
Data base designer	105
Programmer	35.75
System Analyst	96.75
Application Designer	48.1

General Support	18.36
DSS analyst	45.95
Clerical support	26.87

Table 4: 3NF
JOB(*JOB_CLASS*,CHG_HOUR)

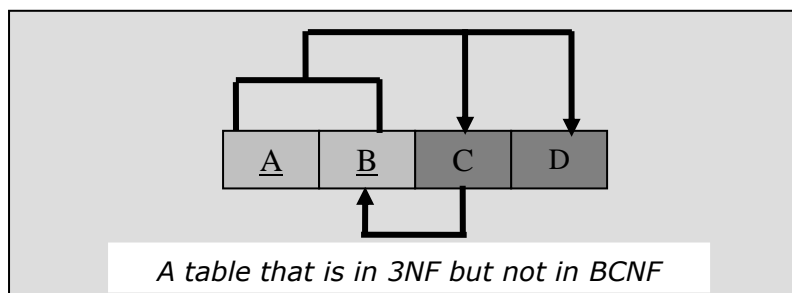
ADVANCED NORMAL FORMS (BCNF, 4NF and 5NF)

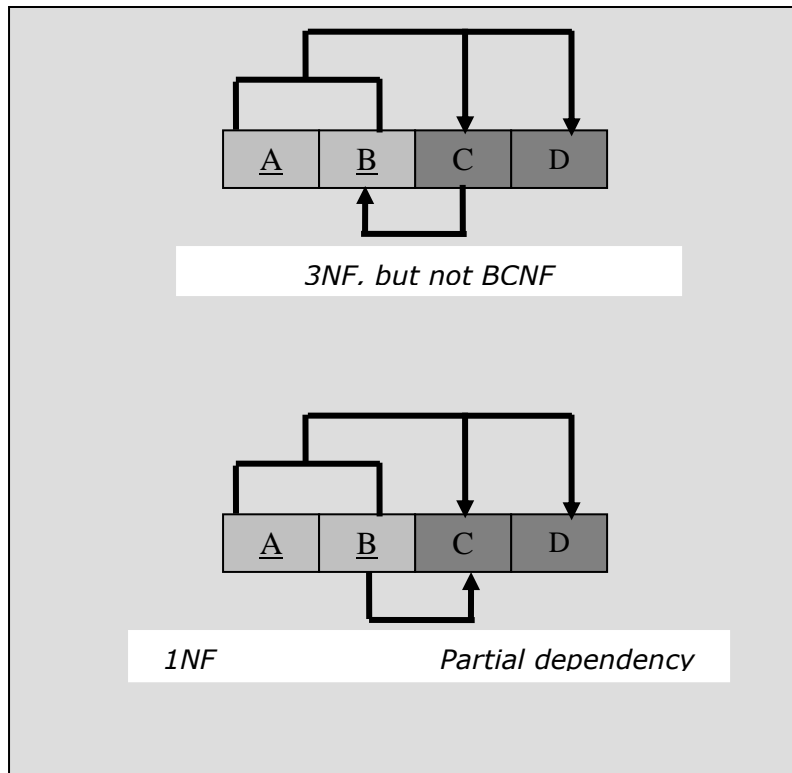
THE BOYCE-CODD NORMAL FORM (BCNF)

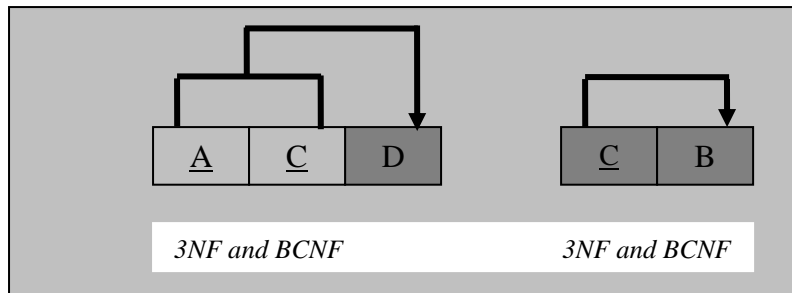
A table is in Boyce-Codd normal form (BCNF) when every determinant in the table is a candidate key. When a table contains only one candidate key, the 3NF and the BCNF are equivalent.

$$\begin{array}{l}
 A + B \longrightarrow C, D \\
 C \longrightarrow B
 \end{array}$$

The table structure shown in below figure has no partial dependencies, nor does it contain transitive dependencies. (the condition $C \rightarrow B$ indicates that a non-key attribute determines part of transitive).







Decomposition to BCNF

The **difference between 3NF and BCNF** is that for a functional dependency $A \rightarrow B$ the third normal form allows this dependency in a relation if B is a Primary key attribute and 'A' is not a candidate key where as BCNF insist that for this dependency to remain in a relation 'A' must be a candidate key. Therefore BCNF is a stronger form of the 3NF such that every relation in BCNF is also in the 3NF. However a relation in the 3NF is not necessarily in BCNF.

Fourth Normal Form (4NF):

Multi-valued dependencies (MVDs) are removed it satisfies the 4NF.

No, table should contain two or more one-to-many or many-to-many relationships that are not directly related to the key. These kinds of relationships are called multi-valued dependencies (MVDs).

Consider a table (relation) EMPLOYEE that has the attributes Name, Project and Hobby. A row in the EMPLOYEE table represents the fact that an employee works for a project and has a hobby. But an employee can work in more than one project and can have more than one hobby. The employee's projects and hobbies are independent of one another. To keep the relation state consistent we must have a separate tuple to represent every combination of an employee's project and an employee's hobbies. This constraint is specified as a multi-valued dependency on the EMPLOYEE relation. The multi-valued dependency can be avoided using the fourth normal form. **EMPLOYEE TABLE**

EMPLOYEE		
NAME	PROJECT	HOBBY
Alexis	Microsoft	Reading

Alexis	Oracle	Music
Alexis	Microsoft	Music
Alexis	Oracle	Reading
Mathews	Intel	Movies
Mathews	Sybase	Riding
Mathews	Intel	Riding
Mathews	Sybase	Movies

We resolve this by decomposing the EMPLOYEE table into two tables that satisfy the fourth normal form as follows.

PROJECT	
Alexis	Microsoft
Alexis	Oracle
Mathews	Intel
Mathews	Sybase

HOBBY	
Alexis	Reading
Alexis	Music
Mathews	Movies
Mathews	Riding

Fifth Normal Form (5NF): Any remaining anomalies have been removed.

UNIT - III

Interaction with Databases and Construction of Information System

SQL (STRUCTURED QUERY LANGUAGE)

SQL INTRODUCTION

SQL was first introduced by Dr CODD. SQL allows users of the database to communicate with the database. Data in a relational database retrieved using a standard language like SQL. It is English like computer language. This makes interaction between user and data base very simple.

Advantages of SQL

- ❖ Non procedural language because more than one record can be accessed rather than one record at a time.
- ❖ It is the common language for all relational data bases. In other words it is portable and it requires very few modifications that it can work on other databases.
- ❖ Very simple commands for quarrying inserting, deleting and modifying data.
- ❖ Easy to learn can handle complex situations.
- ❖ The results to be expected are well defined.
- ❖ The language has sound theoretical bases and there is no ambiguities about the way of query.

Types of SQL

The SQL is of two types, they are

1. Interactive SQL
2. Embedded SQL

While both operate exactly the same way, their usage differs.

1. Interactive SQL

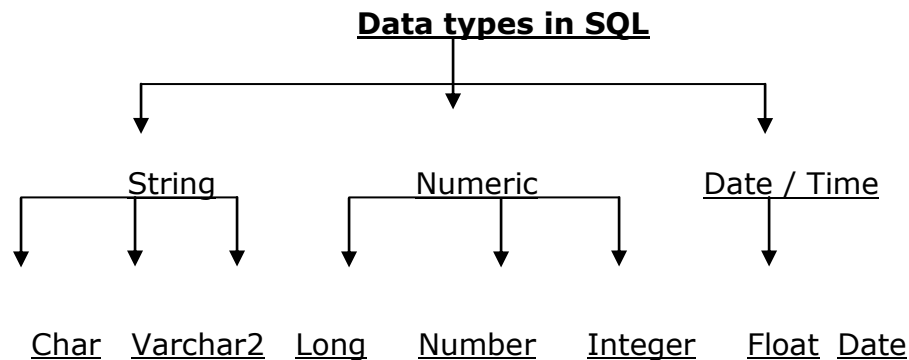
Interactive SQL is used for interacting directly with the database, where the out put of the operation is used for human consumption. As the term suggests, the commands are interactive. Once the command is specified, it is executed and the output is immediately viewed by the user.

2. Embedded SQL

In the case of embedded SQL, commands are SQL commands that are written in some other language, such as COBOL or Pascal. This makes the programs very fast and powerful.

DATA TYPES

Oracle supports the following data types of achieve the above requirements.



String data types: The following are the string or character data types supported by oracle.

1. Char
2. Varchar2
3. Long

Char: The char data type is used, when fixed length character string is required, *it can store alphanumeric values*. The column length of such a data typed can vary b/w 1-2000 bytes. By default it is one byte.

If the user enters a value shorter than the specified length, then the Database blank pads to the fixed length. In case if the user enters a value larger than the specified length, then the Database would return an error.

Varchar2: The varchar2 data type supports a variable length character string. *It also stores alphanumeric values*. The size for this data type ranges from 1-4000 bytes. While defining this data type we should specify the size. Using varchar2 saves disk space when compare to char.

Long:- This data type is used to store variable character length. The maximum size is 2GB, *it is also used to store alphanumeric values*. Long data type has same characteristics similar to varchar2 data type.

Note: In general most commonly data type is varchar2.

Numeric data type:

The following are the various data type related to numeric that are.

1. Number
2. Integer
3. Float

Number: The number data type can store the +ve numbers, -ve numbers, zero's, fixed point numbers and floating point numbers.

Integer: Integer represents for signed integers only.

Float: The float represents for a floating point numbers.

Date: Date data type is used to store date and time in a table. Oracle data base makes use of its own format to store date and time. The data consists of day, month, year.

Time consists of HH, MM, SS
 The format of the date is dd-mm-yy
 The format of the time is hh-mm-ss

To view systems data and time we can use the SQL function called Sysdate().

Types of SQL Commands: SQL can be classified based on its functionality. SQL provides a comprehensive set of commands for variety of tasks of including the following.

DDL(Data Definition Language)	=	Create, Alter and Drop
DML(Data Manipulation Language)	=	Insert, Update, Delete
DQL(Data Query Language)	=	Select
DCL(Data Control Language)	=	Grant, Revoke
DAS(Data Administration Statements)	=	Start audit, Stop audit
TCS(Transaction Control Statements)	=	Commit, Rollback, Save point, Set traction

DDL: The DDL is used to *create, alter and delete data base objects*, the commands used are *create, alter and drop*.

DML: The DML commands allow the user to *manipulating data*. Manipulation data refers commands used in DML is *Insert, Update and Delete*.

DQL: This is one of the most commonly used SQL statement. To *retrieve or access* data from the table we use data query language. The SQL has only one DQL statement is *select*.

DCL: The data control language consists of commands that control the user access in the Data Base accepts *DCL is mainly related to the security issues*, i.e. determining who has access in the database objects and what operations they can

perform on them. The task of the DCL to base administrators DBA is the responsible for this language. The DCL commands are *grant and revoke*.

DAS: The DAS allow the user to perform *audits and analysis* on operations with in the Data Base. They are also used to *analyze the performance of the system*. Two data administration commands are *start audit and stop audit*.

TCS: The TCS are statements which are manage all the changes make by the DML statements some of the transaction Commands are *commit, rollback, save point and set transaction*.

SQL operators: Operators conditions are used to perform operations, such as addition, subtraction or comparison on the data items in SQL statements. There are two types of operators they are

1. Unary operator
2. Binary operator

Unary operator: The Unary operator operates on only one operand.

Ex: To indicate that 10 is a -ve number we use the unary operator (-) and write -10.

The unary operator are '+' and '-' it will act as binary operators also.

Binary operator: The binary operator operates on two operands.

The examples are multiplication and division etc.

Ex: $10*2$, $10/2$ =Binary operations.

The SQL provides a variety of operators to perform calculations, comparisons etc.

1. Arithmetic operators.
2. Comparison or relational operators.
3. Logical operators.
4. Set operators.

Arithmetic operators: Arithmetic operators are used in SQL expressions to add, sub, mul, div and negate data values. The result of this expression is a numeric value.

The following table shows the various type of arithmetic operators.

Arithmetic operators:

Unary operator: +, - denotes +ve or -ve expressions.

Binary operator: '/' division, '*' Multiplication, '+' Addition, '-' subtraction

Comparison operators: These are used to compare one expression with another.

The result of a comparison is true, false and unknown.

The following table shows the various types of comparison operators in SQL.

OPERATOR	DEFINITION
=	equality
!= or <>	not equal or in equality
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
IN	equal to any member of
NOT IN	not equal to any member of
IS NULL	test for nulls
IS NOT NULL	test for anything other than nulls
LIKE	returns true when the first expression matches the pattern
WILD CARD(%)	like allow and is case sensitive
ALL	compares a value to every value in a list
ANY, SOME	compare a value to each value in a list
EXISTS	through if sub query returns at least one row
>=X AND <=Y	between X and Y

Logical operators: The logical operators are used to produce a single result from combining the two separate conditions. The following table shows the various logical operators and their definitions used in SQL.

OPERATOR	DEFINITION
AND	Returns true if both component conditions are true, other wise returns false
OR	Returns true if either component conditions is true or both conditions are true. Other wise the terms are false.
NOT	Returns true if the condition is false other wise returns false.

When more than two search conditions are combined with AND, OR, NOT. According the standard NOT has the higher precedence, followed by AND followed by OR.

The **truth table** for evaluating the condition expressions is given below.

C1	C2	C1 AND C2	C1 OR C2
T	T	T	T

T	F	F	T
F	T	F	T
F	F	F	F

NOT T	F
NOT F	T

Set operators: Set operators combine the results of two separate queries in a single result. Not all implementations support intersects and minus. So, check your implementation support these features before using them union all is supported by all SQL based products. The following table shows the various set operators used in SQL.

OPERATOR	DEFINITION
Union	Returns all distinct rows from both queries
UNION ALL	Returns all rows from both queries
INTERSECT	Returns all rows selected by both queries
MINUS	Returns all distinct rows that are in the first query, but not in the second query

What is query? And writing a query

"A query is a question" written in the Structured Query Language (SQL). As its name denotes, SQL is best at writing queries. Anything that is a part of in RDBMS table can be retrieved (access) at the users request using SQL.

A SQL query has five basic parts they are:

1. Select
2. From
3. Where
4. Order by
5. Group by

Select: This comprises of the list of columns that have to be displayed in the query result. If you want all the columns of a table to be displayed, instead of writing down all the column names you could simply use and "*". This is a DQL statement.

From: This part identifies the source table of the columns. These could be a single table or more than one table.

Where: This is an optional clause of a query. This part specifies, the limits that the results have displayed according to users condition. If a query does not have a where clause all the rows are selected.

Ex: if the librarian wants to see the list of all members who had borrowed books on the date 3-9-08 the query that has to be return as select mem_code from member where issue date=3-9-08.

Order by: This is an optional clause, that controls the order in which the rows to be displayed by the query are ordered.

Ex: if the librarian wants the results to be ordered by mem_code to be written as Select mem_code from member where issued date =3-9-08 order by mem-code.

Group by: This is an optional part of a query. It is used only when the results of the query have to be grouped based on criteria. Expressions can be specified as criteria for this clause.

Eg: If average salary of the employee in tent is to be found, the following query can be written as select deptno, avg(pay) from employee group by deptno;

Starting SQL:

- I. Click on the start button.
- II. Pointing on programs group icon.
- III. In the corresponding list point on oracle for windows-98 group icon and within it click on SQL plus 8.0

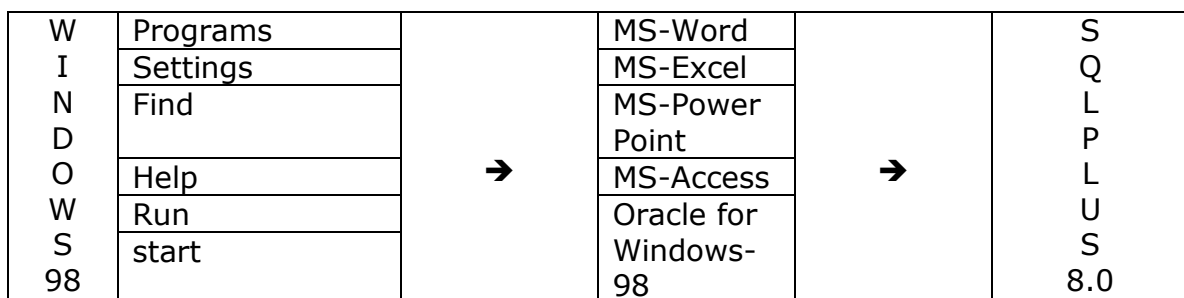


Figure - 1

A screen as shown below after we have start the SQL.

Login	
User name:	<input type="text" value="Scott"/>
Pass Word:	<input type="password" value="*****"/>
Host String:	<input type="text" value="Oracle"/>
<input type="button" value="OK"/> <input type="button" value="Cancle"/>	

Figure - 2

In fig-2 we enter user name as Scott and password is tiger.

- IV. Click on OK button. The screen will appear as shown below.
- V. Displaying the information about the product.

```
Oracle SQL
Plus
File Edit Search Option Help
Copy right
Connected to Oracle 8.0
SQL>|
```

```
Oracle SQL Plus
File Edit Search Option Help
Copy right
Connected to Oracle 8.0
SQL>|
```

Figure – 3

In SQL prompt we can type all executable commands.

How to create a Table:

CREATE statement is used to create a Table. It is a 'DDL' command.

The syntax is as follows:

```
Create table <table name> (Column1 data type (size),
                           Column2 data type (size),
                           ..... Column data type (size));
```

Ex: The following example explains how to create a branch table with the fields branch_code, address1, address2, city.

```
SQL> Create table branch (branch_code varchar2(20),
                          Address1 varchar(10),
                          Address2 varchar(10),
                          City varchar2(10));
```

Table Created.

Table with constraints syntax:

```
Create table <table name>
(Column name1 data type (size),. . . . .
Column name n data type (size),
Constraint <constraint name> primary key (column name1), constraint
<constraint name> foreign key (foreign_column_name) references
referenced_table [(primary_column_Name of referenced table)],
Constraint <constraint name> check (<conditions>));
```

How to view table structure:

To see the structure of the table the giving DESC statement.

The syntax as follows.

```
Desc <table name>;
```

Ex: To view the structure of the branch table the command is

```
SQL> Desc Branch;
```

Output:

<u>Name</u>	<u>Null</u>	<u>Type</u>
Branch_code		varchar2(10)
Address1		varchar2(15)
Address2		varchar2(20)
City		varchar2(10)

How to insert data:

To add records into the table INSERT statement is used. It is a DML command.

The syntaxes of the insert statement are as follows:

1. ADDING A ROW TO A TABLE:

The syntax is

```
INSERT into<table_name>values(value1 for column1,value2 for column 2,.....);
```

Ex:

Let us insert data into the branch table by using insert statement as shown below

```
SQL>insert into branch values('b1','avenue','xroad','piler');
```

```
1 ROW CREATED
```

2. INSERTING RECORDS THROUGH USER INTERACTION

Inserting records through the INSERT statement is very tedious when we have to insert a large number of records. We would have to give as many INSERT statements as the number of records to be inserted. An alternative this is to prompt the user for entering data and repeat the same command.

```
Insert into <table name> values ('&column name1', '&column name2'  
..... .. '&column name n');
```

The '&' would prompt the user to enter data. Single quotes are required if the data type of the column is character or varchar data. If the data to be entered is the numeric than the column name is not to be include with in quotes.

Ex: To insert the values to a branch table the statement is as follows.

SQL> Insert into branch values ('&branch_code', '&address1', '&address2', '&city');

Enter value for branch_code : 001
Enter value for address1 : LBS Road
Enter value for address2 : Padmavathi Nagar
Enter value for city : Piler

SQL>/

Enter value for branch_code : 001
Enter value for address1 : LBS Road
Enter value for address2 : CR.Colony
Enter value for city : Piler

To repeat the command for entering more rows enter back ward slash(/) at the SQL prompt (SQL>).

3. INSERTING A PARTIALLY FILLED RECORD INTO A TABLE:

If we do not want to insert data for all the columns a variant of INSERT can be used.

The syntax is as follows:

Insert into <table_name> (column1,column3,...)values (value1 for column1,value3 for column3,.....);

How to display data:

Data entered into the table can be seen by using SELECT statement. This is a DQL statement.

A) To view all the columns: To view al the columns in the table the syntax is follows:

Select * from <table name>;
--

Here * indicates that the all columns will be displayed.

Ex: To display all the columns in the branch table the statement is

SQL> Select * from branch;

Output: branch code address1 address city

001	LBS Road	Padmavathi Nagar	Piler
002	LBS Road	CR Colony	Piler

2 rows selected.

B) To view selected columns: To view selected columns enter the column names, separated by coma (,) instead of *.

The syntax as follows:

Select <selective column name1, selective column name2> from <table name>;

Ex: To display only branch_code, city from the branch table.

SQL> select branch_code, city from branch;

<u>branch code</u>	<u>City</u>
001	Piler
002	TPT

2 rows selected.

C) Selecting Rows with Conditional Restrictions:

To select partial table contents by placing restrictions on the rows to be included in the output. This is done by adding conditional restrictions to the SELECT statement, using the WHERE clause. The following syntax enables to specify which rows to select.

Select<columnlist>from <tablename> where <condition>

Ex:

SQL> select branch_code ,city from branch where branch_code=001;

How to view existing tables:

Tab is a view which gives us the names of the table created by the current user.

The syntax is as follows:

Select Tname, Tabtype from tab;

SQL> Select Tname, Tab type from tab;

Output: Tname Tab type

Branch	Table
Student	Table
Transaction	View

Where Tname is the column which display the name of table, view, index etc.
Tab type displays whether that object is a table, index, view etc.

How to modify table structure:

All changes in the table structure are made by using the ALTER TABLE command, followed by a keyword that produces the specific change we want to make. Three options are available

1. ADD 2. MODIFY 3. DROP

a) Adding a new column: Management decides to store telephone numbers of the branch office. *We need to add a column to the branch table.*

The syntax is as follows:

Alter table <table name> add (new column name Data type (size));

Ex: To add a telephone column in a branch table the statement is

SQL> Alter table branch add (telephone number (20));

Table altered

SQL> desc branch;

<u>Name</u>	<u>Null</u>	<u>Data type</u>
Branch_code		varchar2(10)
Address1		varchar2(15)
Address2		varchar2(20)
City		varchar2(10)
Telephone		number(10)

b) Modifying existing columns: It is found that the column city may not be sufficient to hold large values. It's size needs to be increased.

The syntax is as follows:

Alter table <table name> modify (existing column name data type(size));

Ex: suppose we want to increase the size of the city column (15)

SQL> Alter table branch modify (city varchar(15));

Table altered.

SQL> desc branch;

<u>Name</u>	<u>Null</u>	<u>Data type</u>
Branch_code		varchar2(10)
Address1		varchar2(15)
Address2		varchar2(20)
City		varchar2(15)
Telephone		number(10)

c) To remove column in the table:

DROP allows to delete a column from a table.

The syntax would be as follows

Alter table <tablename> drop <column name>;

Ex: suppose we want to delete the address1 in the branch table

SQL>alter table branch drop address1;

How to Edit data (or) Advanced data updates:

We need to enter the telephone numbers of all the branch offices for this we need to *modify the existing records through the update statement* the syntax is as follows:

Update <table name>
set <column name1=value1>,
<Column name2=value2>,.(where<condition>);

There fore enter telephone number for branch offices, whose branch code is '001' the statement is as follows:

SQL>Update branch set telephone=9909 where branch_code='001';

One row updated.

SQL> select * from branch;

<u>branch_code</u>	<u>address1</u>	<u>address2</u>	<u>city</u>	<u>Telephone</u>
001	LBS Road	Padmavathi Nagar	Piler	9909
002	LBS Road	CR Colony	Piler	

For enter telephone number to the second record the statement.

SQL> Update branch set telephone=9903 where branch_code='002';

One row updated.

SQL> select * from branch;

branch code	address1	address2	city	Telephone
--------------------	-----------------	-----------------	-------------	------------------

001	LBS Road	Padmavathi Nagar	Piler	9909
002	LBS Road	CR Colony	Piler	9903

Deleting records: It has been decided to delete branch_code 001 record from the table "delete" statement is used. The syntax is as follows:

DELETE <table name> (where <condition>;

Ex: To delete 001 record from branch table the statement is as follows.

SQL> Delete branch where branch_code = '001';

One row deleted.

SQL> Select * from branch;

branch code	address1	address2	city	Telephone
--------------------	-----------------	-----------------	-------------	------------------

002	LBS Road	CR Colony	Piler	9903
-----	----------	-----------	-------	------

Duplicating table:

An entire table with its structure and can be duplicated by the following statement. The syntax is as follows:

Create table <new table name> as <select statement>;

Ex: To duplicate branch table as branch, give the statement is as follows.

SQL> Create table branch1 as select * from branch;

Table created.

SQL> select * from branch;

Removing tables:

Tables which are no larger needed can be deleted from the data base by "drop" statement. The syntax is as follows:

Drop table <table name>;

Ex: The remove branch1 table from the DB the statement is as follows:

SQL> drop table dranch1;

Table dropped.

SQL> Select Tname, tab type from tab;

Relational operators used in a Table:

1. **Equal to:** Equal to operator is used to compare two values for equality.

Ex: To list all the first class passengers travel in a flight, the statement is as follows:

SQL> Select pass_name, class reservation where class='F';

<u>Pass name</u>	<u>Class</u>
Kailash	F
Tarun	F

2. **Not equal (<>):** Not equal operator is used for inequality comparison.

Ex: To view the passengers "who have not booked for first class." Give the statement as shown below.

SQL> Select pass_name, class from reservation. Where class <> 'F';

<u>Pass name</u>	<u>Class</u>
Kumar	B
Pinky	E

3. **In:** To retrieve data which matches a specific list of values? In operator is used.

Ex: to view the passenger "Who have booked for either first or business class." Give the statement as shown below.

SQL> Select pass_name, class from reservation where class in ('F','B');

<u>Pass name</u>	<u>Class</u>
Kailash	F
Tarun	F
Kumar	B

4. **Between and:** When data needs to be retrieved for a "Specific range of values." Between and operator is used.

Ex: to view the

passengers list. Who traveled from 01-Aug-07 to 05-Aug-07 give statement as shown below.

SQL> Select pass_name, class from reservation where date-Journey between '01-Aug-07' and '05-Aug-07';

<u>Pass name</u>	<u>Date of Journey</u>
Kailash	01-Aug-07
Tarun	02-Aug-07
Kumar	05-Aug-07

LIKE Special Operator:

The LIKE special operator is used in conjunction with wildcards to find patterns within string attributes. SQL allows the **Wild cards characters**

- 1) Like %
- 2) Like -

To make matches when the entire string is not known.

1) Like %:

When data needs to be retrieved which matches any sequence of zero or more characters wild card % is used.

Ex: To view all the passengers whose name beginning with 'K'. Give the statement as shown below.

SQL> Select pass_name, class from reservation where pass_name like 'K%';

<u>Pass name</u>	<u>Class</u>
Kailash	F
Kumaran	B

2) Like _:

To match a single character in a pattern ,wild card "_" is used.

Ex: Suppose, we want to find out whether a passenger named 'Jerry' or 'Jenny' or someone having a name beginning with 'Je' and fifth character 'y' ,has made any reservations, then the statement as shown below.

SQL>select pass_name from reservation where rtrim(pass_name)like 'je_y';

Output: PASS_NAME

```
-----  
      Jerry  
      Jenny
```

Here we are not sure about what the 3rd and 4th character are. It could be anything.

Rtrim() is a function which removes trailing and extra spaces in the string. It is required when the length of the pattern that we are matching is not the same length or size of the column.

Logical operators:

Logical operators AND, OR and NOT are used to produce a single result by combining two separate conditions.

AND: When we want that the rows to be retrieved should satisfy both the conditions then AND operator can be used.

Ex: To view that passengers of first class and who have avail for special service for wheel chair (WC) give the statement as shown below.

```
SQL> Select pass_name, class, ss_code from reser where class='F' and ss_code='WC';
```

OR: When we want that the rows to be retrieve should satisfy either of the conditions. Then OR operator can be used.

Ex: To view the passengers of business class OR who have avail for special service for wheel chair give the statement as follows:

```
SQL> Select pass_name, class, ss_code from reser where class='B' OR ss_code="WC";
```

NOT: Some times we need to **negate a condition** for this not operator can be used. NOT can be used with all the operators such as is NOT NULL, NOT BETWEEN and NOTIN, NOTLIKE etc.

Ex: To view passengers who are not flying in 01-Oct-07 and 03-Oct-07 give the statement as shown below.

```
SQL> select pass_name, date-of-jou, class from reser where date-of-jou NOTIN('01-Oct-07', '03-Oct-07');
```

ADVANCED SELECT QUERIES:

Ordering Data:

Data that is displayed using select statement can be arranged in a particular order. I.e. ascending or descending.

Ordering can be done in the ascending order or descending order. By specifying the ASC or DESC parameter. If no parameter is specified the default is the ascending order. The ordering can be done on signal column or multiple columns.

Single column ordering: The order by option can operate on a single column.

Ex: To view the passengers names according to the class give the following statement.

```
SQL> Select pass_name, flight no, date-of-jou, reser date, class from reservation order by class;
```

Multiple column ordering: The order by option can operate on multiple columns also. Ex: To view the passengers names branch wise, the following statement is given below.

```
SQL> select pass_name, flight no, branch_name, flight_code from reser order by branch_code, pass_name.
```

Advanced SQL

SQL JOINS

SQL JOIN OPERATORS

INTRODUCTION:

A join is one of the most powerful features in RDBMS. ***The SQL Plus allows, retrieving data from more than one table at a time. This retrieve is done with the help of join.*** This powerful feature makes possible in retrieving data from any number of tables.

Definition of joins: A join is a query that enables retrieval of rows from more than one table having a common column b/w both the tables.

Types of Joins:

Depending upon the type of join condition used to make the query. Join can be classified in to three types. They are...

1. Equi Join
2. Outer Join
3. Self Join

1) Equi Join (Natural join):

- ❖ This is simplest type of join that can be made. It is also called a natural join.
- ❖ The equi join has a predicate based on equality.
- ❖ This being the simplest type of joins that most commonly used.
- ❖ The join that are based on equal to operator (=) are called equi join.
- ❖ A Natural join returns all rows with matching values in the matching columns and eliminates duplicate columns.
- ❖ The syntax of equi join is as follows.

Select <columns list>
 From <tables name list>
 Where table1.common column=table2.common column
 And table3.common column=table n.common column;

Ex: Consider the following tables.

Table1:(student)

Regno	Name	College
101	A	SHP
102	B	GDC
103	A	GDC
104	D	SHP
105	E	SHP
106	F	GDC

Table 2:(Marks)

Regno	Marks
101	56
102	65
103	75
104	85
105	95
106	75
107	60

Suppose we want to view the student details like regno, name, college, marks. For viewing the above mentioned details, we have to refer to two tables namely student, marks. For retrieving data from both tables we have to join those two tables with one column i.e., common to both the tables.

Equi Join operation:

SQL> Select student.reg,name,college, marks from student, marks where student.regno=marks.regno;

Output:

Regno	Name	College	Marks
101	A	SHP	56
102	B	GDC	65
103	A	GDC	75
104	D	SHP	85
105	E	SHP	95
106	F	GDC	75

In above table equi join select the values that are common in the tables. Where as it does not select the values that are not common in the tables.

2) Outer Join: Outer join is a join it is similar to equi join with a small difference. An outer join is different from all other types of joins as it returns.

1. All the rows that are returned by a simple join and
2. All the rows of one table that do not match with the rows in the other table.

The equi-join does not select the values that are not common in the tables, but in outer join does select the values that are not common in the tables. These values can be selected by using (+) operator.

Syntax:

```
Select <columns list>
from <tables list>
where table1.common column=table2.common column(+);
```

Ex:

SQL> Select student, Regno,Name,College,Marks
 from student, marks where
 student.regno=marks.regno(+);

Output:

Regno	Name	College	Marks
101	A	CNR	56
102	B	GDC	65
103	A	GDC	75
104	D	CNR	85
105	E	CNR	95
106	F	GDC	75
107			60

3) Self Join: (Joining a table itself): In some situations, we may find out it necessary to join a table to itself, as though you were joining two separate tables. This is refer to as a self join.

“In a self join, two rows from the same table combine to form a result row.”

Ex: Retrieve the names of the employees and the names of their respective manages from the employee table.

Table name: Employee

Emppo	Name	Managerno
E001	Vasu	E002
E002	Ravi	E005
E003	Rahul	E004
E004	Raghu	--
E005	Vinita	--

SQL> Select emp.name, mngr.name manager from employee emp, employee mngr
Where emp.mngrno=mngr.empno.

Output:

Name	Manager
Vasu	Ravi
Ravi	Vinita

SET OPERATORS:

SQL plus 8.0 supports the following 4 types of set operators. They are...

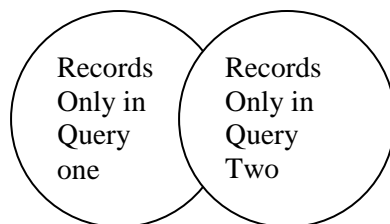
1. UNION
2. UNIONALL
3. INTERSECT
4. MINUS

The above set operators mainly used to combine two or more rows are to display common rows from similar tables to produce results.

1) UNION: Union is a combination of two or more rows from similar tables to produce results. This combination is done using the union operator.

"The Union operator combines two or more rows from similar tables and returns only distinct values from them. It cannot display duplicate values (OR) records."

Output of the union class: (do not display duplicate records)



The syntax of the union operator is as follows:

```
Select<column names> from <table name1>  
UNION  
Select<column names> from<table name2>;
```

MDBM ly

Ex: Consider the following tables:

Employee1			Employee2		
ID	NAME	SALARY	ID	NAME	SALARY
101	A	5000	106	G	9000
102	B	4500	107	H	4000
103	C	6000	108	I	6000
104	D	7000	109	J	3000
105	E	5000	110	K	8000

To display all the employee names from employee-1 and Employee-2 tables.

SQL> Select name from employee1

UNION

select name from employee2;

Output: **Name**

A
B
C
E
F
G
H
C

8 rows selected.

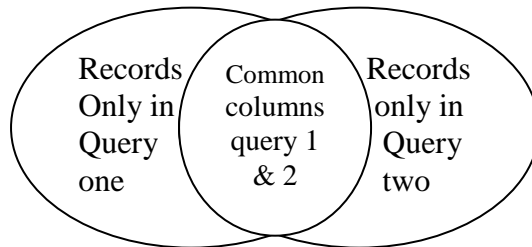
The following restrictions on using a union are as follows.

- ➔ No. of columns in all the queries should be the same
- ➔ The data types of the columns in each query must be same.
- ➔ Aggregate functions can not be used with union class.

2) UNION ALL:

This operator is used to display all records from both the tables including "duplicate, records." So, it is very useful to find out the total number of records in two tables. It is represented by a keyword "UNION ALL."

Output of Union All:



(It displays duplicate records also)

The syntax is as follows:

```
Select <column names> from <table name1>  
UNION ALL  
Select <column names> from <table name2>;
```

Ex: Display all the employee names including duplicate records from both the tables.

```
SQL> select name from employee1  
UNION ALL  
Select name from employee2;
```

Output: Name

A
B
C
E
F
G
H
C
I
A

10 rows selected.

Difference between Union and Union All:

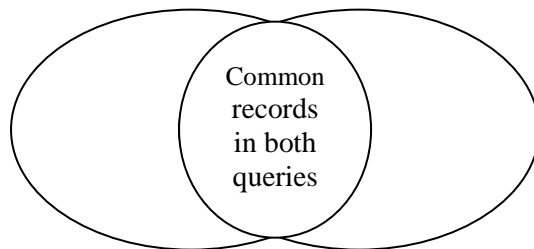
UNION	UNION ALL
--------------	------------------

It displays distinct rows from both tables. It cannot display duplicate records.	It can display distinct rows and duplicate records also.
Using the keyword UNION	Using the keyword UNION ALL
We cannot find out the total number of records in a table	We can find out the table numbers of records in a table.

3) INTERSECT:

The intersect operator returns the rows that are "common" between the tables.

Output:



Syntax: The syntax Intersect class is as follows:

```
Select<column names> from <table name1>
INTERSECT
Select<column names> from <table name2>;
```

Query: Display the common employee names from employee1 and employee2 tables.

```
SQL> Select name from employee1
      INTERSECT
      select name from employee2;
```

Output: **Name**

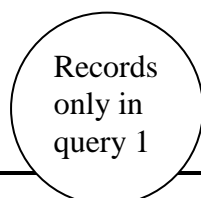
A
C

Two rows selected.

4) MINUS:

The Minus operator combines the results of two queries and returns only those values selected by the "First query and not the second."

Output:



Syntax:

```
Select <column names> from <table name1>  
MINUS  
Select <column names> from <table name2>;
```

table.

SQL> Select name from employee_1 MINUS select name from employee_2;

Ex: List the employee names in employee1 table but not in employee2 table.

SQL> Select name from employee1

MINUS

Select name from employee2

Output: **Name**

B
E
F

3 rows selected.

Ex: List the employee names in employee2 table but not in employee1 table.

SQL> select name from employee2

MINUS

select name from employee1;

Output: **Name**

G
H
I

IEWS:

1. Introduction
2. Definition of View
3. Types of views and their syntaxes
4. Advantages of views
5. Disadvantages of views
6. Dropping views.

Introduction: After a table is created and populated with data, it may become necessary to “prevent all users from accessing all columns of a table” for data security reasons. “To reduce redundancy of data to the minimum possible,” Oracle allows the creation as an object called a VIEW.”

Definition:

- "A view is a virtual table the content of which is taken from table with the help of query, so view is an imaginary table.
- It is derived only from base table.
- The changes made in the based tables are automatically reflected in the view.
- We can create any number of views to a base table.
- If the base table is dropped, automatically all of its views will be dropped.

Types of views: The views can be classified into two types. They are....

1. Simple view
2. Complex (OR) composite view.

1. **Simple view:** A "View" is said to be simple if it is created only on one base table. The syntax of the simple view is as follows...

Create view <view name> as select <statement> from <base table name>;

2. **Composite view:** A view is created "based on multiple tables." It is said to be composite view. The composite view is not updatable view. The syntax of the composite view is as follows. . . .

Create view <view name> as select <statement> from <table1, table2>
where <condition>;

Advantages of Views: Some of the major advantages are listed below:

1. **Data Security:** Views allow setup different security levels. For the same base table, thus protecting certain data from people who do not have proper authority.
Ex: For the use of the data entry clerk in the personal department a view of the employee master table can be created, which exclude confidential details about a person like his salary etc.
2. The views allow "The same data to be seen by different users in different ways" at the same time.
3. Views can be used to hide complex queries.
4. It provide row and column level security.

5. To ensure efficient access paths.
6. To ensure proper data derivation.
7. To provide domain support.
8. To rename columns.

Disadvantages of views:

- If the definition of the view involves either a group by or a having clause and the outer most level, then the views are not updatable.
- The order by option cannot be used with a view.
- If a column of a view is derived from an aggregate function then the "view is not updatable."
- If the from clause in the view definition involves multiple range variables, then it is not updatable.
- If the definition of the view involves distinct at the outermost level, then that view is not updatable.
- The updating is possible for simple views, but more complex views cannot be updated, there are "read only."
- A view is defined on a non updatable view is not updatable.

Dropping a view: A view is defined by the create view statement there is no alter view statement as in the case of base tables.

If we want to delete (or) remove an existing view we can do, by using the drop view statement. The syntax of the drop view statement is as follows.

```
Drop view <view name>;
```

CONSTRAINTS:

How to create constraints on the table:

Constraints are used to define certain validations or restrictions. That has to be followed while entering data into the tables. We have to create constraints on two tables. Namely

1. Creating constraints on a new table
2. Creating constraints on existing table.

Creating constraints on a new table: Constraints can be defined at two levels they are

1. Column level
2. Table level

Column Constrains: The syntax of column constrains is as follows. . . .

```
Create table <table name>
(<column name1 data type(size) constraint <constraint name> references
reference_table[(primary_colume_name of referenced table)],
Column name3 data type(size) constraint <constraint_name> check [<condition>],
column name4 data type (size) not null);
```

Table constraint: The syntax is as follows:

```
Create table <table name>
(Column name1 data type (size),. . . . .
Column name n data type (size),
Constraint <constraint name> primary key (column name1), constraint <constraint
name> foreign key (foreign_column_name) references referenced_table
[(primary_column_Name of referenced table)],
Constraint <constrant name> check (<conditions>));
```

Ex: Let us create a table fare with certain data enter restrictions.

Route-code should be a primary key

Route-dese, source and destination should not be empty.

First class fare should be greater than zero.

Business class fare should be less than first class.

Economy class fare should be less than business class.

```
SQL> Create table fare (route_code varchar2(10) constraint route_pk primary key,
route_dese varchar2(15) not null, source varchar2(12) not null, destination
varchar2(10) not null, first_fare number(5) constraint first_fare_zero check
(first_fare>0), bus_fare number(5),Eco_fare number(5), constraint
bus_fare_greater_first check(bus_fare<first_fare), constraint
eco_fare_greater_business check (eco_fare<bus_fare));
```

Creating constrains and Existing tables: Constrains can also be enclosed on the existing tables. The syntax is as follows:

```
Alter table <table name> add constraint <constraint name> <constraint>;
```

Ex: Let as make branch_code of branch table a primary key.

```
SQL> Alter table branch add constraint branch_pk primary key (branch_code);
Table altered.
```

SQL> Desc branch

<u>Name</u>	<u>Null</u>	<u>Type</u>
Branch_code	Not null	Varchar2(10)

Grouping data from tables in SQL:

In SQL for grouping data the following classes used.

1. Grouping clause.
2. Having clause.

Group by clause:

The group by clause is another section of the select statement. This is optional clause. This tells oracle to group rows based on distinct values that exist for specified columns. That is it "creates a data set containing several sets of records grouped together based on a condition.

Syntax: the syntax group by clause is

Select columns list from <table name> group by <column>;

Ex: Retrieve the product numbers and the total quantity ordered for each product from the sales order details table.

Sales_order_details:

SQL>

<u>Order No</u>	<u>Product No</u>	<u>Qtyordered</u>	<u>Qtydisp</u>
01001	P00001	10	10
01001	P00004	3	3
01001	P00006	7	7
01002	P00002	4	4
01002	P00005	10	10
01003	P00003	2	2
01004	P00001	6	6
01005	P00006	4	4
01005	P00005	1	1
01006	P00006	8	8

Select product no, sum (qtyordered) from sales_order_details groupby productno;

Output:

<u>Productno</u>	<u>Qtyorder</u>
P00001	16
P00002	4
P00003	2
P00004	4
P00005	10
P00006	19

Note: The aggregate functions are also used in group by option.

Having clause:

The having clause can be used in conjunction with the group by clause. Having imposes a condition on the group by clause. Which further filters the groups created by the group by clause.

Syntax: The syntax of group by with having clause is. . . .

```
Select <column list> from <table name> groupby column having <condition>;
```

Ex: Retrieve the product number and the total quantity ordered for products, P00001, P00004 from the sales order details table.

SQL> Select product, sum(qtyorder) from sales_order details groupby product no having product no='P00001' or 'P00004';

Output:	<u>Productno</u>	<u>Qtyorder</u>
	P00001	16
	P00004	4

PROGRAMING LANGUAGE STRUCTURED QUERY LANGUAGE (PL/SQL)

INTRODUCTION

1. SQL does not support programming. It is used to write single line statements called Quarries.

2. To support programming constructs, additional features are added to SQL and name it as PL/SQL.
3. PL/SQL is very useful to develop programming.
4. PL/SQL supports almost all programming constructs like variables, conditional statements, looping, etc.

PL/SQL BLOCK STRUCTURE

The programming area in PL/SQL is called PL/SQL block. This block mainly contains three parts. They are.

1. Declarative part
2. Executable part
3. Exception handling part

It is shown in the following structure

```
SQL>Declare
.....
..... } variable declaration
.....
Begin
Statement 1;
Statement 2;} Executable part
Statement 3;
End ;
Exception . . . . . Error handling part
```

The above syntax shows that every PL/SQL block declarative part should be indicated with a key word called "declare". Executable part must be enclosed with in begin and end key words. Exception handling part can be indicated with a key ward called it "Exception".

Input and Output statement in PL/SQL:

Input statement

The input statement in PL/SQL is & operator.

& Operator :-

It is to give the input values to the variables.

Ex:- a:= &a;
x:= &x; etc. . . .

Output statement

The out put statement in PL/SQL is

```
dbms_output.put_line ('message' / variable);
```

Ex:- SQL> **dbms_output.put_line** ('Welcome to Oracle');

Set server output on

This command is used to open server. RDBMS can display the result of any program, only when the server is on.

Syntax:- Set server output on; <enter>

PL/SQL Data types

PL/SQL supports and allows all SQL data types like number(n), Int, float, Varchar, varchar2, etc.

PL/SQL Operators

PL/SQL allows all SQL operators like arithmetic operators, relational operators, logical operators, etc.

Example programs:

Q Write a PL/SQL program to add two numbers.

```
SQL> Declare
2   a number(3);
3   b number(3);
4   c number(3);
5   Begin
6   a:=&a;
7   b:=&b;
8   c:=a + b;
9   dbms_output.put_line('result'||c);
10  End;
```

/ <enter>

PL/SQL procedure successfully completed

SQL> / <enter>

Enter value for a=10 <enter>

Enter value for b=20 <enter>

Result=30

PL/SQL procedure successfully completed.

SQL>

Q Write PL/SQL program to calculate area of a rectangle.

```
SQL> Declare
2   Len number(3);
3   B number(3);
4   A number(3);
```

```

5 Begin
6 Len:=&len;
7 B:=&b;
8 A:=len*b;
9 Dbms_output.put_line ('Area='||a);
10 End;
/ <enter>

```

Enter value for len=20
Enter value for b=5
Area=100
PL/SQL procedure successfully completed;

Q Write PL/SQL program to calculate simple interest and total amount.

```

SQL> Declare
2   p number(5);
3   t float;
4   r float;
5   I float;
6   A float;
7   Begin
8   P:=&p;
9   T:=&t;
10  R:=&r;
11  I:=(P*T*R)/100;
12  A:= P+ I;
13  dbms_output.put_line ('Simple Interest='||i);
14  dbms_output.put_line ('Total Amount='||a);
15  End;
/ <enter>

```

Enter value for p=500
Enter value for t=5
Enter value for r=5
Simple Interest=125
Total Amount=625
PL/SQL procedure successfully completed
SQL>

BRANCHING

Branching is a process in which the control will be jumped from one statement to another statement. In a program, they jumping may be based on condition or without condition.

Branching with condition is called conditional branching and the branching without condition called unconditional branching.

For conditional branching SQL provides the following statements.

1. If
2. If then else
3. If then else if

1. If Statement

It is used for conditional branching purpose in SQL.

If:

It is used to test only one condition. It returns when if the condition is true. Otherwise the condition will exit. Every if statement should be ended with key word "End if".

Syntax:- If<Condition> then
Statement 1;
End if;

Ex: If (Num>0) then
dbms_output.put_line ('Positive number='||num);
End if;

2. If then else

It is used to test a condition it returns true if the condition is true. It returns false if the condition is false.

Syntax:- If <condition> then
Statement 1;
Else;
Statement 2;
End if;

In the above syntax statement1 will be executed when the condition is true. Otherwise statement2 will be executed.

Ex:- If (a>b) then
dbms_output.put_line ('a is big'||a);
Else
dbms_output.put_line ('b is big'||b);
End if;

3. If then Else if

This statement is used to test multiple conditions one by one sequentially.

Syntax:- If <condition1> then
Statement1;
Else if <condition2> then
Statement2;
Else if <condition3> then
Statement3;
.....
Else
Statement n;

End if;

In the above syntax if the condition1 is true. Statement 1 will be executed.

If condition 1 is false, it will go for checking condition2 and so on.

```
Ex:- If((a>b) and (a>c)) then
      dbms_output.put_line ('a is big'||a);
Else if (b>c) then
      dbms_output.put_line('b is big'||b);
Else
      dbms_output.put_line('c is big'||c);
End if;
```

Q Write a PL/SQL program to test whether the given number is positive or not.

```
SQL> Declare
2   Int a;
3   begin
4   a:=&a;
5   if (a>0) then
6   dbms_output.put_line('positive='||a);
7   end if;
8   end;
```

SQL>/ <enter>

Enter value for a=11 <enter>

Results= positive 11

PL/SQL procedure successfully completed.

Q Write PL/SQL program to find biggest number between two numbers.

```
SQL> Declare
2   Int a;
3   Int b;
4   begin
5   a:=&a;
6   b:=&b;
7   if (a>b) then
8   dbms_output.put_line('a is big:='||a);
9   else
10  dbms_output.put_line('b is big:='||b);
11  end if;
12  end;
```

Q Write PL/SQL program to test whether it is even number or odd number.

```
SQL> Declare
2   Int a;
3   begin
4   a:=&a;
5   if (a mod 2=0) then
6   dbms_output.put_line('Even number:='||a);
7   else
8   dbms_output.put_line('odd number:='||a);
9   end if;
10  end;
```

SQL> / <enter>

Q Write PL/SQL program to find biggest among three numbers.

```
SQL> Declare
2   Int a;
3   Int b;
4   int c;
5   begin
6   a:=&a;
7   b:=&b;
8   c:=&c;
9   if ((a>b) and (a>c)) then
10  dbms_output.put_line('a is big:='||a);
11  else if (b>c) then
12  dbms_output.put_line('b is big:='||b);
13  else
14  dbms_output.put_line('c is big:='||c);
15  end if;
16  end;
```

SQL> / <enter>

Enter value for a=10

Enter value for b=20

Enter value for c=15

Results = b is big = 20

Q Write a PL/SQL program to find whether the given number is positive or negative or zero.

```
SQL> Declare
2   Int a;
3   begin
4   a:=&a;
5   if (a>0) then
6   dbms_output.put_line('positive' || a);
7   else if (a<0) then
8   dbms_output.put_line('negative' || a);
9   else
10  dbms_output.put_line('zero' || a);
11  end if;
12  end;
```

SQL> / <enter>

Enter value for a =0

Results = zero

PL/SQL procedure successfully completed.

Q Write a PL/SQL program to find smallest number among three values.

```
SQL> Declare
2   Int a;
3   Int b;
4   Int c;
5   begin
6   a:=&a;
```

```

7      b:=&b;
8      c:=&c;
9      if ((a<b) and (a<c)) then
10     dbms_output.put_line('a is small='||a);
11     else if (b<c) then
12     dbms_output.put_line('b is small='||b);
13     else
14     dbms_output.put_line('c is small='||c);
15     end if;
16     end;

```

SQL> / <enter>

Enter value for a =10

Enter value for b =5

Enter value for c = 25

Results = b is small = 5 PL/SQL procedure successfully completed.

LOOPING

Looping is a process in which a block of statements will be executed repeatedly until given condition is true. PL/SQL provides two types of looping statement they are:

1. For loop Statement
2. While loop Statement

A) For loop Statement

This looping statement will execute loop body repeatedly initial value to final value for every execution. The for loop counter will be incremented by one. So, by using this statement we can also find the number of iterations. This statement also called **recursive looping** statement.

Syntax:- for counter variable in [reverse] initial vale.. final value
 Loop

 } loop body

 End loop;

Here reverse is used to print the given ranged values in reversing order.

Ex:- for I in 1 .. 10	<i>Results:</i>
Loop	<i>1 to 10</i>
dbms_output.put_line(i);	
End loop	
For I in reverse 1 .. 10	<i>10 to 1</i>
Loop	
dbms_output.put_line (i);	

End loop

Note: Every for loop statement should be ended with "End loop" statement.

Q Write PL/SQL program to print natural numbers from 1 to 10.

```
SQL> Declare
2   Int i;
3   Begin
4   For I in 1..10
5   Loop
6   dbms_output.put_line (i);
7   End loop;
8   End;
SQL> / <enter>
```

Q Write a PL/SQL program to display numbers in reversing order from 10 to 1

```
SQL> Declare
9   int i;
10  Begin
11  For I in reverse 1..10
12  Loop
13  dbms_output.put_line (i);
14  End loop;
15  End;
SQL> / <enter>
```

Q Write a PL/SQL program to print natural number up to given limit.

```
SQL> Declare
2   I number(3);
3   N number(3);
4   Begin
5   N:=&n;
6   For I in 1..n;
7   Loop
8   dbms_output.put_line (i);
9   End loop;
10  End;
SQL> / <enter>
```

Enter value for n=15

Results = 1 2 3 4 5 6 7.. . 15.

PL/SQL procedure successfully completed.

Q Write a PL/SQL program to print natural numbers from 20 to given limit in reversing order.

```
SQL> Declare
2   I number(3);
3   N number(3);
4   Begin
5   N:=&n;
6   For I in reverse 20..n
7   Loop
8   Dbms-output. Put-line(i);
```



```

9      End loop;
10     End;
SQL> / <enter>
Enter value for n= 30
Results = 30 29 28 27 . . . .1
PL/SQL procedure successfully completed.

```

B) While Loop

This looping statement can execute loop body based on a condition. This statement with check the condition first if the condition is true loop body will be executed. Otherwise loop will be terminated.

Syntax:- While <condition>
 Loop
 Statement1
 Statement2

 End loop;

} Loop body

Ex: While (i<10)
 Loop
 dbms_output.put_line (i);
 I:=I + 1;
 End loop;

Q Write a PL/SQL program to print natural number from 1 to 10.

```

SQL> Declare
2      I number(5);
3      Begin
4      I:=1;
5      While (i<=10)
6      Loop
7      dbms_output.put_line (i);
8      I:=I + 1;
9      End loop;
10     End;

```

Q Write PL/SQL code to print even numbers up to 20.

```

SQL> Declare
2      I number(5);
3      Begin
4      I:=0;
5      While (i<20)
6      Loop
7      dbms_output.put_line (i);
8      I:= I + 2;
9      End loop;

```

10 End;

Q Write a PLSQL program to print a number in reverse order:

```
SQL> Declare
      N number(3):=&n;
      Rem number(5);
      Rev number(5):=0;
      Begin
      While (n<>0);
      Loop
      Rem:=mod(n,10);
      Rev:=rev*10 + rem;
      N:=float (n/10);
      End loop;
      Dbms-output. Put-line ('reverse number='||rev)
      End;
/ <enter>
```

Enter value for n= 786

Reverse number= 687

PLSQL procedure successfully completed

Q Write a PL/SQL program to print odd number up to given limit.

```
SQL> Declare
2     I number(5);
3     N number(3);
4     Begin
5     N:=&n;
6     I:=1;
7     While (i<=n)
8     Loop
9     dbms_output.put_line (i);
10    I:= I + 2;
11    End loop;
12    End;
```

Q Write a PL/SQL program to swap to numbers.

```
SQL> Declare
2     Int a;
3     Int b;
4     temp int;
5     begin
6     a:=&a;
7     b:=&b;
8     temp:=a;
```

```
9    a:=b;
10   b:=temp;
11   dbms_output.put_line ('a='||a);
12   dbms_output.put_line ('b='||b);
13   end;
```

Q

Swapping without using third variable or temporary variable.

```
SQL> Declare
2    Int a;
3    int b;
4    begin
5    a:=&a;
6    b:=&b;
7    a:=a + b;
8    b:=a - b;
9    a:=a - b;
10   dbms_output.put_line ('after swapping');
11   dbms_output.put_line ('a='||a);
12   dbms_output.put_line ('b='||b);
13   end;
```

CURSORS

Cursor is a private working area or a cursor is a temporary working area. **It is mainly used for retrieving multiple rows based on multiple attributes at a time. Cursors are very useful to update big structure of a table at a time.** Cursors can calculate calculation according to given conditions and fetching rows into base tables. Cursors are mainly divided into two types in SQL. They are:

1. Implicit cursors
2. Explicit cursors

1. **Implicit cursors:** These are automatically executed by SQL engine internally.
Ex: Create table, Alter table, Create view, Update etc.
2. **Explicit cursors:** The cursors which are coded by data base programmers are called Explicit cursors.

The execution of explicit cursors is depending on the data base programmers code all the users created cursor are explicit cursors.

Cursors operations:

Every explicit cursor can be operated by using the following cursor operation.

1. Declaring a cursor
2. Opening a cursor
3. Fetching rows in to cursor
4. Closing a cursor

1. Declaring a cursor

Before using any cursor that should be declared first. In one PL/SQL program we may use many cursors; all cursors must be defined under declarative part of the PL/SQL block. A cursor is a data base object, it is declared by using the following syntax:

Syntax: Cursor <cursor name> is <select statement>;

Ex: Cursor C1 is select * from Emp;

2. Opining a cursor

In order to work with cursor, we must open it after creation. To open a cursor open key word is used.

Syntax: Open cursor name;

Ex: Open C1;

3. Fetching rows in to cursor

To calculate values host variables are useful after calculations. The values must be fetched to host variables. To do so fetching cursor is used.

Syntax: Fetch <cursor name> into host variables,

Ex: Fetch C1 into TA DA HRA;

Fetching must be performing to all rows successfully. So it must be included with in looping statement.

4. Closing cursor

After working successfully with a cursor, it must be closed when we close a cursor. It will releases all resources. To close a cursor close key word is used.

Syntax: Close<cursor name>;

Ex: Close C1;

CURSOR ATTRIBUTES

Cursor attributes are used to know the status of the cursor at a particular time. Cursor attributes are,,,,,,,,,

1. % is open
2. % row count
3. % row type
4. % is found
5. % not found

1. %is open: This attribute is used to know whether a cursor is opened or not. It returns true if the cursor is opened, otherwise it returns false.

Syntax: <cursor name>%is open;

Ex: C1 %is open

2. %row count: It provides the information about number of rows successfully fetched into tables currently.

Syntax: <cursor name>%row count;

Ex: C1%row count

3. %row type: It can equates the data types of table attributes and host variables, this attribute always be used while defining a cursor.

Syntax: <cursor name>% row type;

Ex: cursor C2 is select *From Emp C2% row type;

4. %is found: It provides the information about resent fetch operation. If the fetch statement is successfully completed it returns true other wise it returns false.

Syntax: <cursor name>% found;

Ex: if C2 is % found then

Update Emp set net Sal:=TA + HRA + DA

5. %not found: This attribute is used to find whether record are available or not in the cursor. Cursor operations can be perform until %not found is false.

Syntax: <cursor name> %not found;

Ex: Loop

Fetch C1 into TA, DA, BASIC, HRA

Exit when C1% not found;

Net sal:= basic + TA + DA

End loop

PROGRAMS BASED ON CURSORS

Q Write a PL/SQL program to process 10th class result. Creation of 10th class result table.

```
SQL> create tenth(htno number(10) primary key, name varchar2(10), Tel
number(3), Eng number(3), Hin number(3), Maths number(3), Sci
number(3), SS number(3), Total number(4), Result varchar2(10));
```

Table Created:

```
SQL> Insert into tenth(htno, name, Tel, Eng, Hin, Maths, Sci, SS)
values(205156730,'rani',50, 60, 70, 70, 50, 70); <enter>
```

1 Row created

Similarly enter 10 rows

Q PL/SQL program for updating total and results.

```
SQL> Declare
```

```
Cursor ten_cur is select htno, Tel, Eng, Hin, Maths, Sci, SS from tenth;
```

```
Hno number(10);
```

```
Nme varchar2(10);
```

```
Tel number(3);
```

```
Eng number(3);
```

```
Hin number(3);
```

```
Mat number(3);
```

```
Sci number(3);
```

```
SS number(3);
```

```
TT number(4);
```

```
Res number(10);
```

```
Begin
```

```
Open ten_cur;
```

```
Loop
```

```
Fetch ten_cur into hno, Tel, Eng, Hin, Maths, Sci, SS;
```

```
Exit when ten_cur %not found;
```

```
TT:= Tel + Eng + Hin + Maths + Sci + SS;
```

```
Update tenth set total=TT where, htno=hno;
```

```
If ((Tel<35) or (Eng<35) or (Hin<35) or (Maths<35) or (Sci<35) or
(SS<35)) then
```

```
Update tenth set result='fail' where htno=hno;
```

```
Else if (TT>=360) then
```

```
Update tenth set result='first' where, htno=hno;
```

```
Else if ((TT.=300) and (TT<360)) then
```

```
Updae tenth set result='second' where, htno=hno;
```

```
Else
```

```
Update tenth set result='pass' where, htno=hno;
```

```
End if
```

```
End loop
```

```
Close ten_cur
```

```
End;
```

```
SQL> / <enter>
```

PL/SQL procedure successfully completed

```
SQL> select *from tenth;
```

TRIGGERS

- ❖ Use of database triggers
- ❖ Basic parts in triggers
- ❖ Types of triggers
- ❖ Dropping a trigger

Triggers: A trigger PL/SQL block associated with a table. Trigger can be fire implicitly (internally) when a particular event has occurred.

By using “triggers we can restrict the end user in performing invalid DML statements.” The DML statements are insert, delete and update.

Uses of database Triggers:

- A trigger can permit 'DML' statements against a table.
- The triggers can be used to prevent invalid transactions.
- Enforce complex security authorizations.

Basic parts in Triggers:

Triggers have three parts namely:

- The Event
- The Condition and
- The Action

These parts are reflected in the coding structure of triggers.

Types of Triggers:

There are twelve basic types of triggers. A trigger's type is defined by the type of triggering transaction and by the level at which the trigger is executed. The following describe these classifications:

- **Row-level triggers**
- **Statement-level triggers**
- **Before and after triggers**

Row-level triggers:

Row-level trigger, trigger once for each row in a transaction. These types of triggers are very useful in cases like audit trails, where you want to track the modification made to the data in a table. Different RDBMS have implemented the row-level triggers in different ways. For example, in ORACLE's PL/SQL, you can create a row-level trigger by using the FOR EACH ROW clause in the CREATE TRIGGER command.

Statement-level triggers:

Statement-level triggers execute once for each transaction. For example, if a single transaction inserted 700 rows into a table then a statement-level trigger on that table will be executed only once. Statement-level triggers therefore are not often used for data related activities. They are normally used to enforce additional security measures on types of transactions that may be performed on a table. Statement-level triggers are the default type of triggers created using CREATE TRIGGER command.

Before and after triggers:

Since triggers occur because of events, they may set to occur immediately before or after those events. Since the events that execute triggers are database

transactions, triggers can be executed immediately before or after INSERT s, UPDATE s and DELETE s.

Within a trigger, you will be able to reference the **old** and **new** values involved in the transaction. The access required for the old and new data may determine which type of trigger you need. **Old** refers to the data, as it existed prior to the transaction. Updates and deletes usually reference old values. *New values are the data values that the transaction creates. They are referred by the keyword **NEW**.*

If you need to set a column value in an inserted row via your trigger, then you will need to use a BEFORE INSERT trigger in order to access the NEW values. Using an AFTER INSERT trigger would not allow you to set the inserted value, since the row will already have been inserted into the table.

AFTER row-level triggers are frequently used auditing applications, since they do not fire until the row has been modified. Since the row has been successfully modified this implies that it has successfully passed the referential integrity constraints defined for that table.

The syntax for the CREATE TRIGGER

```
CREATE [or REPLACE] TRIGGER trigger_name
[BEFORE|AFTER]
[DELETE|INSERT|UPDATE[OF column_name]]
ON[user.]table_name
[FOR EACH ROW][WHEN condition]
[PL/SQL block];
```

Clearly there is a great deal of flexibility in the design of a trigger. The BEFORE and AFTER keywords indicate whether the trigger should be executed before or after the triggering transaction. The DELETE, INSERT and UPDATE keywords indicate the type of data manipulation that will constitute the triggering event.

When the FOR EACH ROW clause is used, the trigger will be row-level-trigger, otherwise, it will be a statement-level trigger. The WHEN clause is used to further restrict when the trigger is executed. The restrictions enforced in the WHEN clause may include checks of old and new data values.

The CREATE TRIGGER command is used to create or replace the Database triggers. For example, Suppose we want to monitor any changes to the amount

that is greater than 40%. The following row-level BEFORE UPDATE trigger will be executed only if the new value of the amount column is more than 40% its old value.

```
CREATE TRIGGER ledger_bef_updrow
BEFORE UPDATE ON ledger
FOR EACH ROW
WHEN (NEW.amount/OLD.amount>1.4)
BEGIN
    INSERT INTO Ledger_audit
VALUES(:OLD.Action_date,:OLD.action,:OLD.amount,:OLD.item)
END;
```

UNIT-4

Transaction Management in DBMS Environment

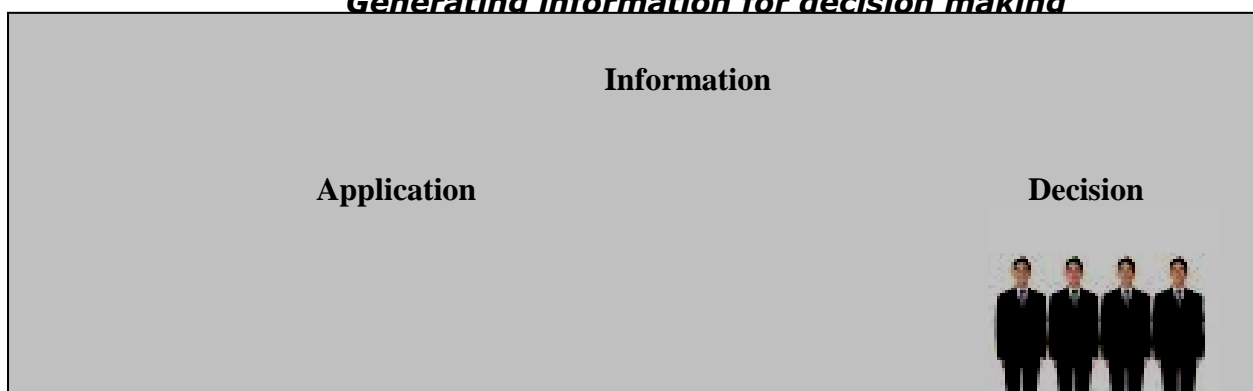
The Information system

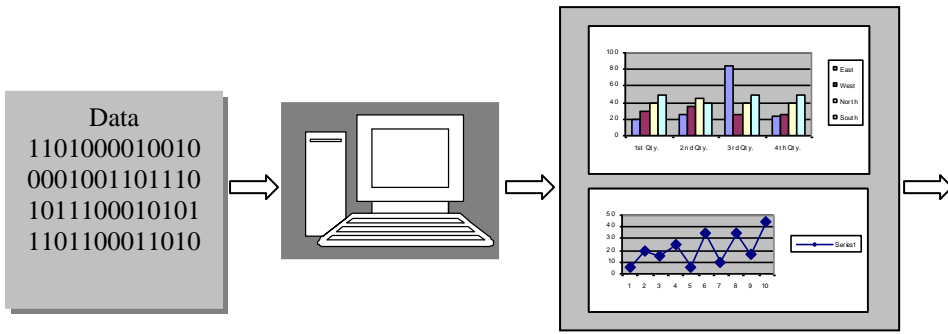
Basically, a database is a carefully designed and constructed repository of facts. The fact repository is a part of a larger whole known as an information system.

An **information system** provides for data collection, storage, and retrieval. It also facilitates the transformation of data into information and the management of both data and information. Thus a complete information system is composed of people, hardware, software, the database(s), application programs, and procedures. Systems analysis is the process that establishes the need for and the extent of an information system. The process of creating an information system is known as systems development.

Within the framework of systems development, applications transform data into the information that forms the basis for decision making. Applications usually produce formal reports, tabulations, and graphic displays designed to procedure insight. The below figure illustrates that every application is composed of two parts: the data and the code (program instructions) by which the data are transformed into information. Data and code work together to represent real-world business functions and activities.

Generating information for decision making





The performance of an information system depends on a triad of factors:

- ❖ Database design and implementation.
- ❖ Application design and implementation.
- ❖ Administrative procedures.

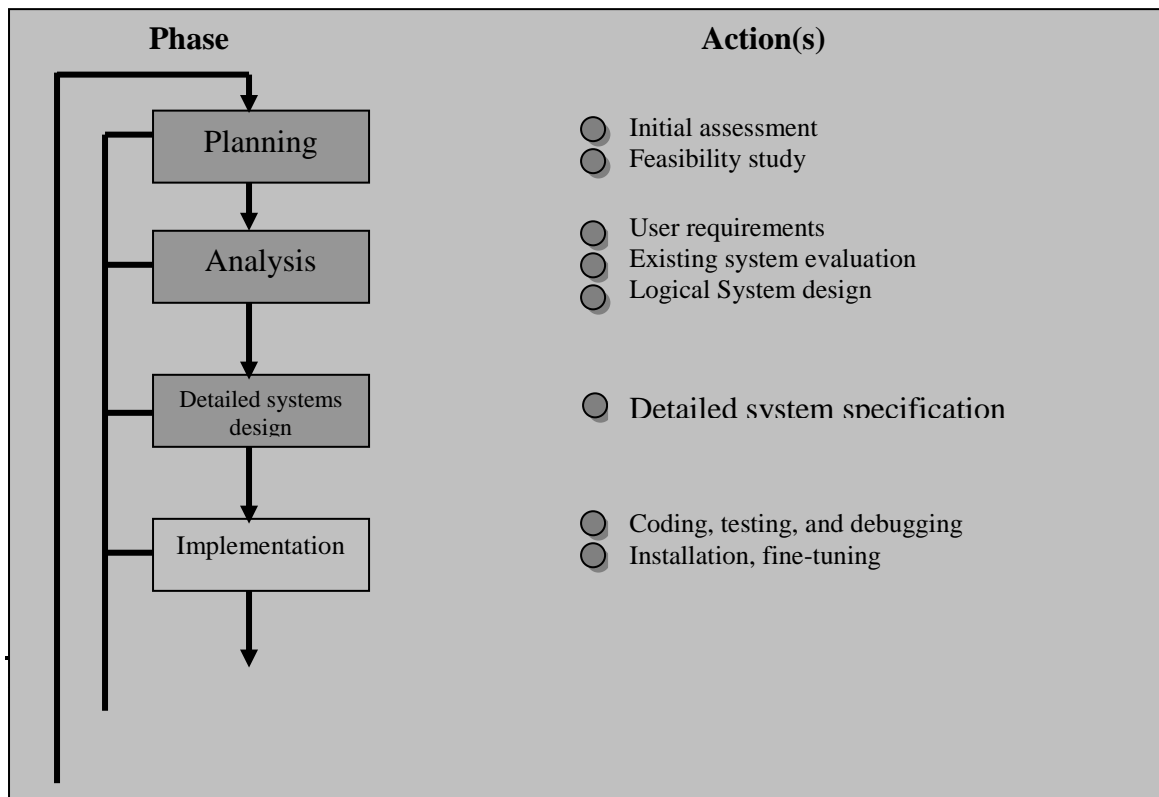
The information that is generated is accurate, timely and relevant, then these systems will go a long way in helping the organization realizing its goals.

The Systems Development Life Cycle (SDLC)

The SDLC traces the history (life cycle) of an information system. Perhaps more important to the system designer, the SDLC provides the big picture within which the database design and application development can be mapped out and evaluated.

The traditional SDLC is divided into five phases: planning, analysis, detailed systems design, implementation, and maintenance. The SDLC is an iterative rather than a sequential process.

The Systems Development Life Cycle (SDLC)





1. PLANNING

The SDLC planning phase yields a general overview of the company and its objectives. An initial assessment of the information-flow-and-extent requirements must be made during this discovery portion of the SDLC. Such an assessment should answer some important questions:

- ❖ Should the existing system be continued?
- ❖ Should the existing system be modified?
- ❖ Should the existing system be replaced?

If it is decided that a new system is necessary, the next question is whether it is feasible. The feasibility study must address the following:

- The technical aspects of hardware and software requirements. The decisions might not be vendor-specific, but they must address the nature of the hardware requirements (PC, midrange, or mainframe) and the software requirements (single or multi-user operating systems, database type and software, programming languages to be used by the applications, and so on).
- The system cost.

2. ANALYSIS

A microanalysis must be made of both individual needs and organizational needs, addressing questions such as:

- ❖ What are the requirements of the current system's end users?
- ❖ Do those requirements fit into the overall information requirements?

The analysis phase of the SDLC is, in effect, a thorough audit of user requirements.

The existing hardware and software systems are also studied during the analysis phase. The result of analysis should be a better understanding of the system's functional areas, actual and potential problems, and opportunities.

Along with a study of user requirements and the existing systems, the analysis phase also includes the creation of a *logical systems design*. The *logical design* must specify the appropriate conceptual data model, inputs, processes, and expected output requirements.

When creating a logical design, the designer might use tools such as data flow diagrams (DFDs), hierarchical input process output (HIPO) diagrams, and entity relationship (ER) diagrams.

3. DETAILED SYSTEMS DESIGN

In the detailed systems design phase, the *designer completes the design of the system's processes*. The design includes all necessary technical specifications for the screens, menus, reports, and other devices that might be used to help make the system a more efficient information generator.

4. IMPLEMENTATION

During the implementation phase, the hardware, DBMS software, and application programs are installed and the database design is implemented. During the initial stages of the implementation phase, the system enters into a cycle of coding, testing, and debugging *until it is ready to be delivered*.

5. MAINTENANCE

Almost as soon as the system is operational, end users begin to request changes in it. Those changes generate system maintenance activities, which can be grouped into three types:

- ❖ *Corrective maintenance* in response to systems errors.
- ❖ *Adaptive maintenance* due to changes in the business environment.
- ❖ *Perfective maintenance* to enhance the system.

THE DATABASE LIFE CYCLE (DBLC)

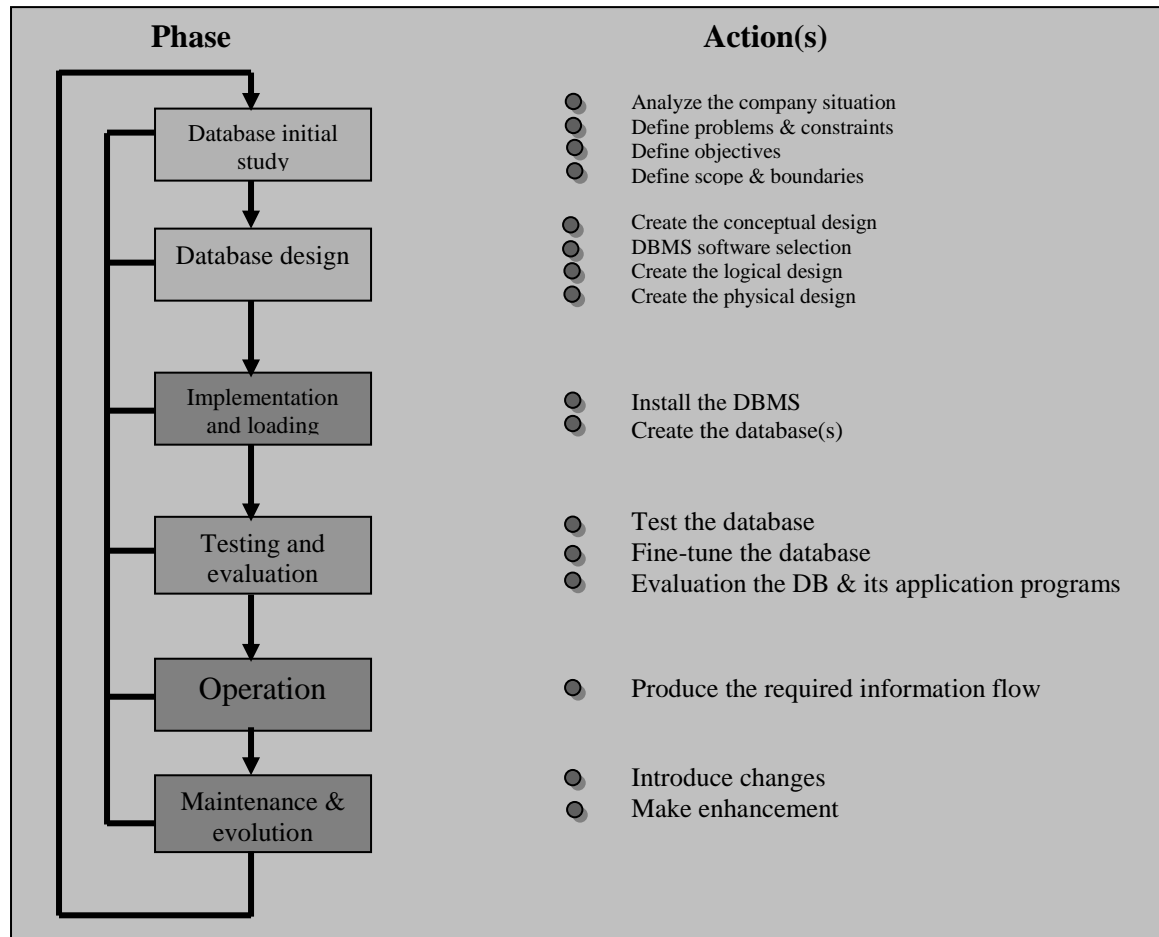
Within the larger information system, the database, too, is subject to a life cycle. **The DBLC contains six phases: database initial study, database design, implementation and loading, testing and evaluation, operation, and maintenance and evolution.**

1. THE DATABASE INITIAL STUDY

Examining the current system's operation within the company, the designer must determine how and why the current system fails. That means spending a lot of time talking with (but mostly listening to) end users.

Depending on the complexity and scope of the database environment, the database designer might be a lone operator or part of a systems envelopment team composed of a project leader, one or more senior systems analysis, and one or more junior systems analysts.

The Database Life Cycle (DBLC)



The overall purpose of the database initial study is to:

- ❖ Analyze the company situation.
- ❖ Define problems and constraints.
- ❖ Define objectives.
- ❖ Define scope and boundaries.

ANALYZE THE COMPANY SITUATION

To analyze the company situation, the database designer must discover what the company's operational components are, how they functions, and how they interact.

These issues must be resolved:

- What is the organization's general operating environment, and what is its mission within that environment?
- What is the organization's structure?

DEFINE PROBLEMS AND CONSTRAINTS

How does the existing system function? What input does the system require? What documents does the system generate? How is the system output used? By whom? Studying the paper trail can be very informative.

During the initial problem definition process, the designer is likely to collect very broad problem descriptions.

DEFINE OBJECTIVES

A proposed database system must be designed to help solve at least the major problems identified during the problem discovery process. If the designer can create a database that sets the stage for more efficient parts management, both departments gain. The initial objective, therefore, might be to create an efficient inventory query and management system.

DEFINE SCOPE AND BOUNDARIES

The designer must recognize the existence of two sets of limits: scope and boundaries. The system's scope defines the extent of the design according to operational requirements. Will the database design encompass the entire organization, one or more departments within the organization, or one or more functions of a single department?

The proposed system is also subject to limits known as boundaries, which are external to the system.

2. DATABASE DESIGN

The second phase focuses on the design of the database model that will support company operations and objectives. This is arguably the most critical DBLC phase: making sure that the final product meets user and system requirements. In the process of database design you must concentrate on the data characteristics required to build the database model.

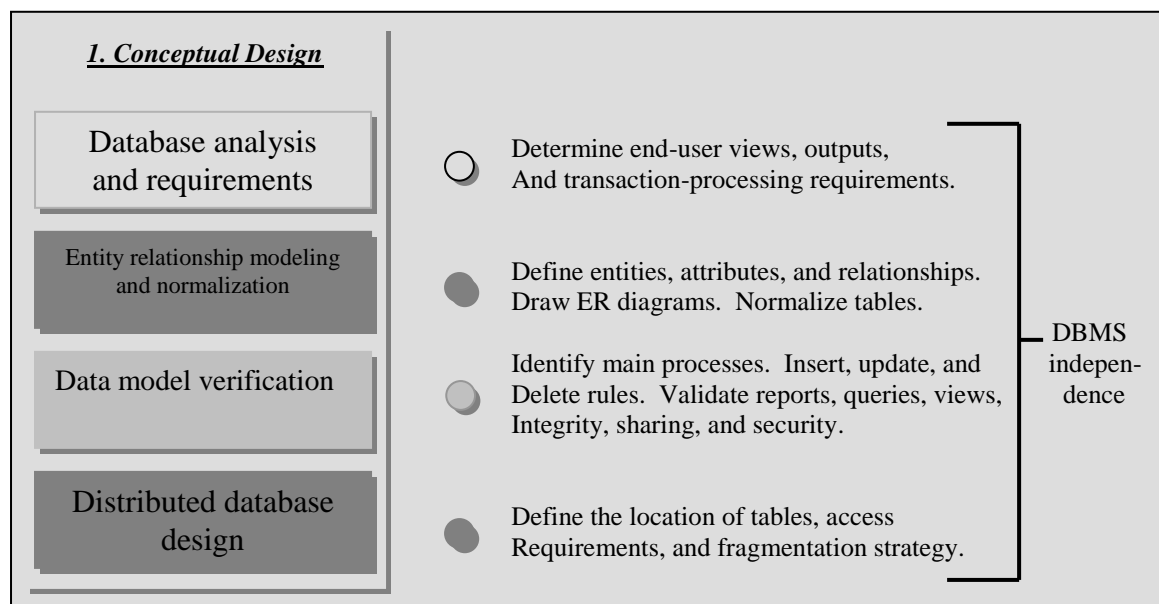
CONCEPTUAL DESIGN

At this level of abstraction, the type of hardware and/or database model to be used might not yet have been identified. Therefore, the design must be software

and hardware independent so the system can be set up within any hardware and software platform chosen later.

- ❖ Determine end-user views, outputs, and transaction-processing requirements.
- ❖ Define entities, attributes, and relationships. Draw ER diagrams. Normalize tables.
- ❖ Identify main processes. Insert, update, and delete rules. Validate reports, queries, view, integrity sharing and security.
- ❖ Define the location of tables, access requirements, and fragmentation strategy.

PROCEDURE FLOW IN THE DATABASE DESIGN



DBMS SOFTWARE SELECTION

The selection of DBMS software is critical to the information system's smooth operation. Consequently, the advantages and disadvantages of the proposed DBMS software should be carefully studied. To avoid false expectations, the end user must be made aware of the limitations of both the DBMS and the database.

LOGICAL DESIGN

Logical design translates the conceptual design into the internal model for a selected database management system (DBMS) such as DB2, SQL Server, Oracle, and Access. Therefore, the logical design is software-dependent.

Logical design requires that all objects in the model be mapped to the specific constructs used by the selected database software. For example, the logical design

for a relational DBMS includes the specifications for the tables, indexes, views, transactions, access authorizations, and so on.

PHYSICAL DESIGN

Physical design is the process of selecting the data storage and data access characteristics of the database. The storage characteristics are a function of the types of devices supported by the hardware, the type of data access methods supported by the system, and the DBMS. Physical design affects not only the location of the data in the storage devise(s), but also the performance of the system.

3. IMPLEMENTATION AND LODING

In most modern relational DBMSs, such as IBM, DB2, Microsoft SQL Server, and Oracle, a new database implementation requires the creation of special storage-related constructs to house the end-user tables. The constructs usually include the storage group, the table space, and the tables. Note a table space may contain more than one table.

During the implementation and loading phase, you also must address performance, security, backup and recovery integrity, and company standards.

PERFORMANCE

Database performance is one of the most important factors in certain database implementations. Performance varies according to the hardware and software environment used. Important factors in database performance also include system and database configuration parameters, such as data placement, access path definition, use of indexes, and buffer size.

SECURITY

Data stored in the company database must be protected from access by unauthorized users.

- ❖ *Physical security* allows only authorized personnel physical access to specific areas.
- ❖ *Password security* allows the assignment of access rights to specific authorized users.
- ❖ *Access rights* can be established through the use of database software.
- ❖ *Audit trails* are usually provided by the DBMS to check for access violations.

- ❖ *Data encryption* can be used to render data useless to unauthorized users who might have violated some of the database security layers.
- ❖ *Diskless workstations* allow end users to access the database without being able to download and information from their workstations.

BACKUP AND RECOVERY

Timely data availability is crucial in the information game. The database can be subject to data loss through unintended data deletion, power outages, and so on. Data backup and recovery procedures create a safety value, allowing the database administrator to ensure the availability of consistent data.

INTEGRITY

Data integrity is enforced through the proper use of primary and foreign key rules.

COMPANY STANDARDS

Database standards may be partially defined by specific company requirements. The database administrator must implement and enforce such standards.

4. TESTING AND EVALUATION

Once the data have been loaded into the database, the DBA tests and fine-tunes the database for performance, integrity, concurrent access, and security constraints. The testing and evaluation phase occurs in parallel with applications programming.

If the database implementation fails to meet some of the system's evaluation criteria, several options may be considered to enhance the system:

- ❖ For performance-related issues, the designer must consider fine-tuning specific system and DBMS configuration parameters.
- ❖ Modify the physical design.
- ❖ Modify the logical design.
- ❖ Upgrade or change the DBMS software and/or the hardware platform.

5. OPERATION

Once the database has passed the evaluation stage, it is considered to be operational. At that point, the database, its management, its users, and its application programs constitute a complete information system.

6. MAINTENANCE AND EVOLUTION

The database administrator must be prepared to perform routine maintenance activities within the database. Some of the required periodic maintenance activities include:

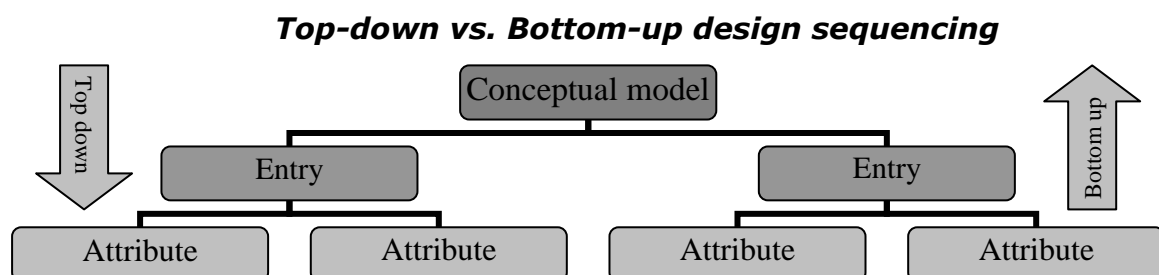
- ❖ Preventive maintenance (backup).
- ❖ Corrective maintenance (recovery).
- ❖ Adaptive maintenance (enhancing performance, adding entities and attributes, and so on).
- ❖ Assignment of access permissions and their maintenance for new and old users.
- ❖ Generation of database access statistics to improve the efficiency and usefulness of system audits and to monitor system performance.
- ❖ Periodic security audits based on the system-generated statistics.
- ❖ Periodic (monthly, quarterly, or yearly) system-usage summaries for internal billing or budgeting purposes.

DATABASE DESIGN STRATEGIES

There are two classical approaches to database design:

1. **Top-down design** starts by identifying the data sets, then defines the data elements for each of those sets. This process involves the identification of different entity types and the definition of each entity's attributes.
2. **Bottom-up design** first identifies the data elements (items), and then groups them together in data sets. In other words, it first defines attributes, and then groups them to form entities.

The two approaches are illustrated in below figure. The selection of a primary emphasis on top-down or bottom-up procedures often depends on the scope of the problem or on personal preferences.



CENTRALIZED VS. DECENTRALIZED DESIGN

The two general approaches (bottom-up and top-down) to database design can be influenced by factors such as the scope and size of the system. Depending on such factors, the **database design may be based on two very different design philosophies: centralized and decentralized.**

Centralized design is productive when the data component is composed of a relatively small number of objects and procedures. The design can be carried out and represented in a fairly simple database. Centralized design is typical of relatively simple and / or small databases and can be successfully done by a single person (database administrator or by a small, informal design team).

Decentralized design might be based when the data component of the system has a considerable number of entities and complex relations on which very complex operations are performed. Decentralized design is also likely to be employed when the problem itself is spread across several operational sites and each element is a subset of the entire data set. In large and complex projects, the database design typically cannot be done by only one person. Within the decentralized design framework, the database design task is divided into several modules. Once the design criteria have been established, the lead designer assigns design subsets or modules to design groups within the team.

TRANSACTION MANAGEMENT AND CONCURRENCY CONTROL

INTRODUCTION

Transaction management is the ability of a database management system to manage the various transactions that occur within the system. Concurrency control is the activities of coordinating the actions of processes that operate in parallel and access shared data, and therefore potentially interfere with each other.

Transaction management and concurrency control issues are in the design of hardware, operating system, real time systems, communications systems and database systems among others.

TRANSACTION PROPERTIES

To ensure data integrity, the database management system should maintain the following **transaction properties – atomicity, consistency, isolation and durability.** These properties are often referred to as **ACID properties** – an acronym derived from the first letter of the properties.

- ❖ **Atomicity** requires that all operations (SQL requests) of a transaction be completed; if not, the transaction is aborted. If a transaction T1 has four SQL requests, all four requests must be successfully completed; otherwise, the entire transaction is aborted. In other words, a transaction is treated as a single, indivisible, logical unit of work.
- ❖ **Consistency** indicates the permanence of the database's consistent state. When a transaction is completed, the database reaches a consistent state.
- ❖ **Isolation** means that the data used during the execution of a transaction cannot be used by a second transaction until the first one is completed. In other words, if a transaction T1 is being executed and is using the data item X, that data item cannot be accessed by any other transaction (T2. . . .Tn) until T1 ends. This property is particularly useful in multi-user database environments because several different users can access and update the database at the same time.
- ❖ **Durability** ensures that once transaction changes are done (committed), they cannot be undone or lost, even in the event of a system failure.
- ❖ **Serializability** ensures that the concurrent execution of several transactions yields consistent results. More specifically, the concurrent execution of transactions T1, T2 and T3 yields results that appear to have been executed in serial order (one after another). This property is important in multi-user and distributed databases, where multiple transactions are likely to be executed concurrently.

TRANSACTION MANAGEMENT WITH SQL (OR) TRANSACTION STATES

If there are no failures, the transactions complete successfully. A transaction that completes its execution is said to be committed. A committed transaction that has performed updates transforms the database into a new consistent state.

A transaction may not always successfully complete its execution. When a transaction has not successfully completed its execution we say

that it has aborted. If we are to ensure the atomicity property, an aborted transaction should not have any effect on the state of database. So any changes made to the database by an aborted transaction should be reversed or undone.

Once all the changes caused by an aborted transaction have been undone we say that the transaction has rolled back. The rolling back of a transaction is handled by the recovery-management component of the database.

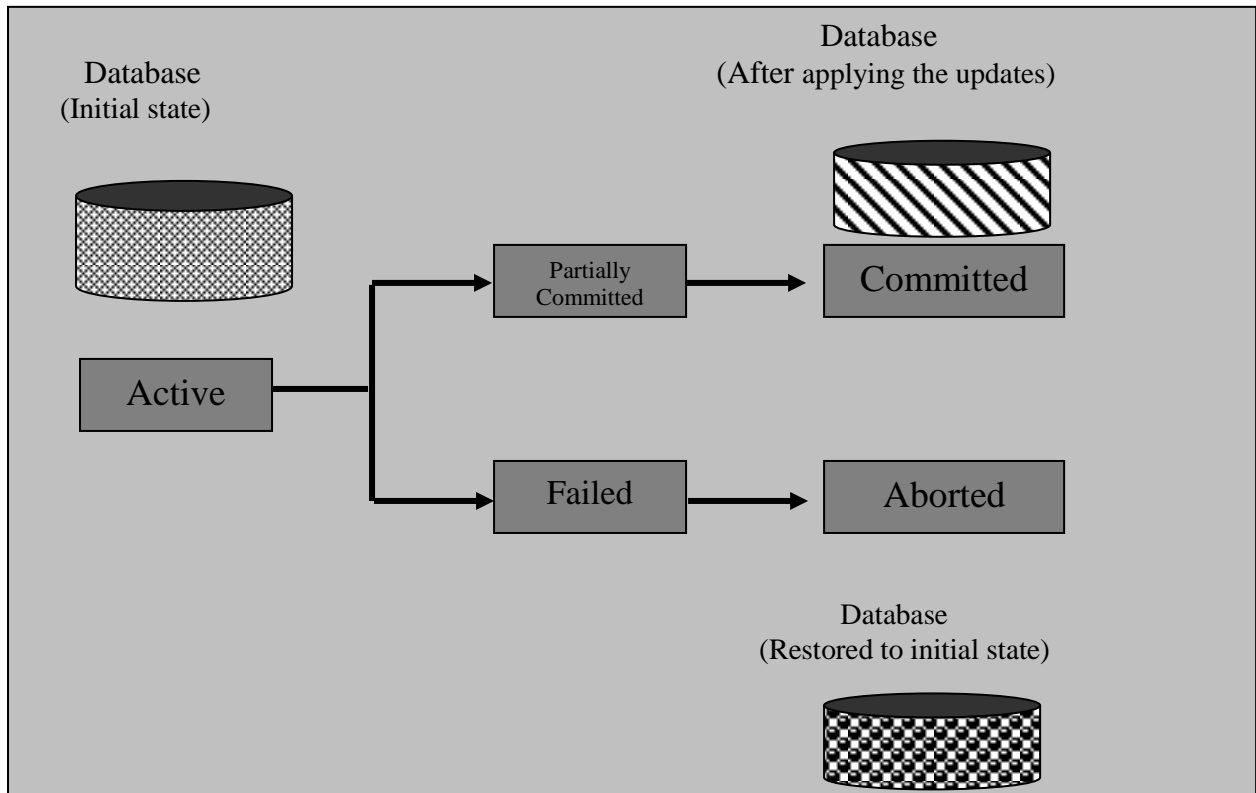
Once a transaction is committed, we cannot undo the changes made by the transaction by rolling back the transaction. A transaction must be in one of the following states:

- ❖ **Active** – This is the initial state, the transaction stays in this state while it is executing.
- ❖ **Partially committed** – A transaction is in this state when it has executed the final statement.
- ❖ **Failed** – A transaction is in this state once the normal execution of the transaction cannot proceed.
- ❖ **Aborted** – A transaction is said to be aborted when the transaction has rolled back and the database is being restored to the consistent state prior to the start of the transaction.
- ❖ **Committed** – A transaction is in the committed state once it has been successfully executed and the database is transformed into a new consistent state.

We say a transaction has committed only after it has entered the committed state. Similarly we say a transaction has aborted only after it has entered the aborted state (i.e. after the transaction is rolled back and the database is restored to the consistent state prior to the start of the transaction). **A transaction that is either committed or aborted is said to be terminated.**

A transaction starts in the active state. **A transaction contains a group of statements that form a logical unit of work.** When the transaction has finished executing the last statement, it enters the partially committed state. At this point the transaction has completed execution, but it is still possible that it may have to be aborted. This is because the actual output may still be in the main memory and a hardware failure can still prevent the successful completion. The database system

then writes enough information to the disk-either updates the database or writes enough information (enough to recover the database to the new consistent state in the event of a failure) into the log files. When the last of this information is written, the transaction enters the committed state.



Transaction support is provided by two SQL statements:

COMMIT and ROLLBACK. When a transaction sequence is initiated by a user or an application program, the sequence must continue through all succeeding SQL statements until one of the following four events occurs:

1. A COMMIT statement is reached, in which case all changes are permanently recorded within the database. The COMMIT statement automatically ends the SQL transaction.
2. A ROLLBACK statement is reached, in which case all changes are aborted and the database is rolled back to its previous consistent state.
3. The end of a program is successfully reached, in which case all changes are permanently recorded within the database. This action is equivalent to COMMIT.

The program is abnormally terminates, in which case the changes made in the database are aborted and the database is rolled back to its previous consistent state. This action is equivalent to ROLLBACK

THE TRANSACTION LOG

A DBMS uses a **transaction log to keep track of all transactions that update the database**. The information stored in this log is used by the DBMS for a recovery requirement triggered by a ROLLBACK statement, a program's abnormal termination, or a system, failure such as a network discrepancy or a disk crash.

Some RDBMSs use the transaction log to recover a database *forward* to a currently consistent state. After a server failure, for example, Oracle automatically rolls back uncommitted transactions and rolls forward transactions that were committed but not yet written to the physical database.

While the DBMS executes transactions that modify the database, it also automatically updates the transaction long. The transaction log stores:

- ❖ A record for the beginning of the transaction.
- ❖ For each transaction component (SQL statement):
 - The type of operation being performed (update, delete, insert)
 - The name of the objects affected by the transaction (the name of the table).
 - The "before" and "after" values for the fields being updated.
 - Pointers to the previous and next transaction log entries for the same transaction.
- ❖ The ending (COMMIT) of the transaction.

CONCURRENCY CONTROL

The process of managing simultaneous operations against a database so that data integrity is maintained and the operations do not interfere with each other in a multi-user environment.

Database administrators must expect and plan for the likelihood that several users will attempt to access and manipulate data at the same time. With concurrent processing involving updates, a database without concurrency control will be compromised due to interference between users. There are two basic approaches to concurrency control: a pessimistic approach (involving locking) and an optimistic approach (involving versioning).

Most DBMSs run in a multi-user environment, with the expectation that users will be able to share the data contained in the database. If users are only reading data, no data integrity problems will be encountered, because no changes will be made in the database. However, if one or more users are updating data, then potential problems with maintaining data integrity arise. When more than one transaction is being processed against a database at the same time, the transactions are considered to be concurrent.

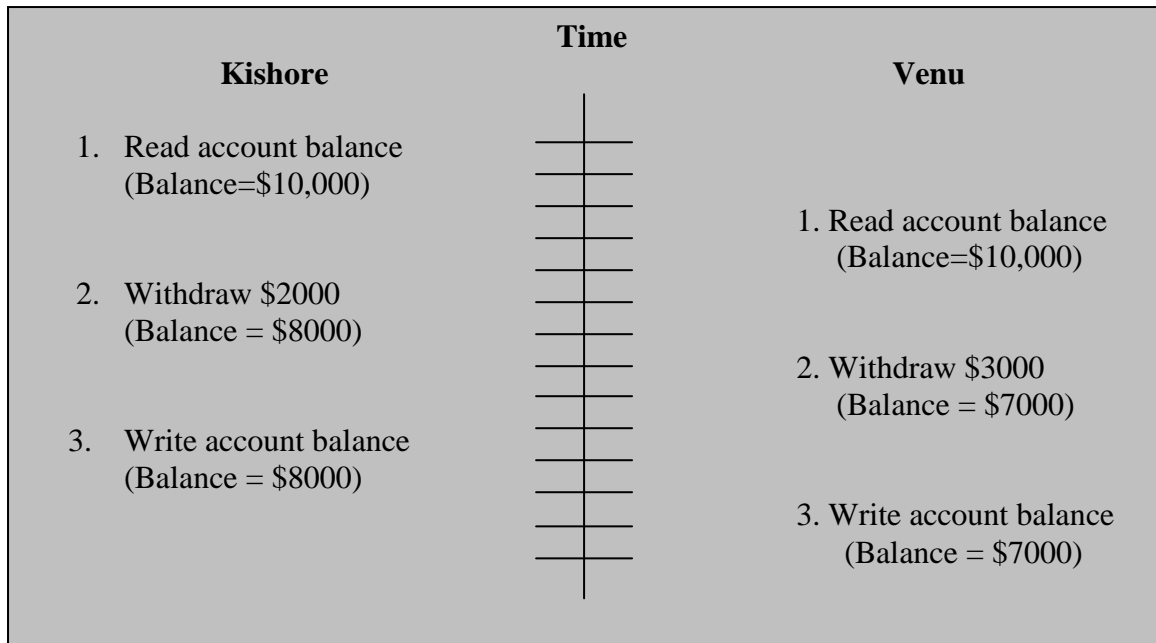
The actions that must be taken to **ensure that data integrity is maintained are called concurrency control actions**. Remember that the CPU can process only one instruction at a time. As new transactions are submitted while other processing is occurring against the database, the transactions are usually interleaved.

The coordination of the simultaneous execution of transactions in a multi-user database system is known as concurrency control. The objective of concurrency control is to ensure the serializability of transactions in a multi-user database environment. Concurrency control is important because the simultaneous execution of transactions over a shared database can create several data integrity and consistency problems.

The three main problems are lost updates, uncommitted data, and inconsistent retrievals.

THE PROBLEM OF LOST UPDATES

The most common problem encountered **when multiple users attempt to update a database without adequate concurrency control is that of lost updates**. Figure(a) shows a common situation. Kishore and Venu have a joint checking account and both want to withdraw some cash at the same time, each using an ATM terminal in a different location. In the below figure shows the sequence of events that might occur, in the absence of a concurrency control mechanism. Kishore's transaction reads the account balance (which is \$10,000) and he proceeds to withdraw \$2000. Before the transaction writes the new account balance (\$8000), Venu's transaction reads the account balance (which is still \$10,000). He then withdraws \$3000, leaving a balance of \$7000. His transaction then writes this account balance, which replaces the one written by Kishore's transaction. The effect of Kishore's update has been lost due to interference between the transactions, and the bank is unhappy.



UNCOMMITTED DEPENDENCY PROBLEM

The uncommitted dependency problem occurs when one transaction is allowed to see the intermediate results of another transaction before it is committed. An example of such a situation is showing below table. Here T1 starts execution and after the results are written to the database it aborts due to some reason. Since the transaction has aborted, the database should be restored to the original consistent state. But before the roll back is performed, transaction T2 reads the account balance and starts executing. So instead of a balance of 70000 (since only transaction T2 was committed), we now end up with a wrong result of 60000. **This happened because the transaction T2 was permitted to read the intermediate result of transaction T1 i.e., before transaction T1 was terminated (either committed or rolled back).**

Uncommitted Dependency Problem

Sequence	T1	T2	Account Balance
01	Begin transaction		50000
02	Read (CA2090)		50000
03	CA2090:=CA2090 - 10000		50000
04	Write (CA2090)	Begin transaction	40000
05		Read (CA2090)	40000
06	Roll back	CA2090=CA2090+20000	50000

07		Write (CA2090)	60000
08		commit	60000

This problem is avoided by preventing T1 from reading the account balance until the transaction T1 is terminated – i.e. either committed or rolled back.

INCORRECT ANALYSIS PROBLEM

The problems arising when concurrent transactions are updating the database. But problems could arise even when a transaction is not updating the database transactions that read the database can also produce wrong results, if they are allowed to read the database when the database is in an inconsistent state. This problem is often referred to as **dirty read** or **unrepeatable read**. The problem of dirty read occurs when a transaction reads several values from the database while other transactions are updating those values.

SERIALIZABILITY

Some of the problems associated with the concurrent execution of transactions. The objective of concurrency control is to schedule or arrange to transactions in such a way as to avoid any interference. One obvious (but very inefficient) way of avoiding the interference of the transactions is to execute them one at a time – one transaction is committed before another is allowed to begin. But in a multi-user environment, where there are hundreds of users and thousands of transactions the serial execution of the transactions is not a viable option. The DBMS will have to find ways and devise strategies to maximize concurrency in the system, so that many transactions can execute in parallel without interfering with one another.

A **non-serial schedule** is a schedule where the operations from a group of concurrent transactions are interleaved. In the case of a non-serial schedule, the problems that we have seen earlier (multiple update, uncommitted dependency and incorrect analysis) can arise, if the schedule is not proper. Serial execution prevents the above-mentioned problems.

The objective of serializability is to find non-serial schedules that allow transactions to execute concurrently without interfering with one another and thereby producing a database state that could be produced by a serial execution. In serializability, the order of the read and write operations are important and the serializability rules are given below:

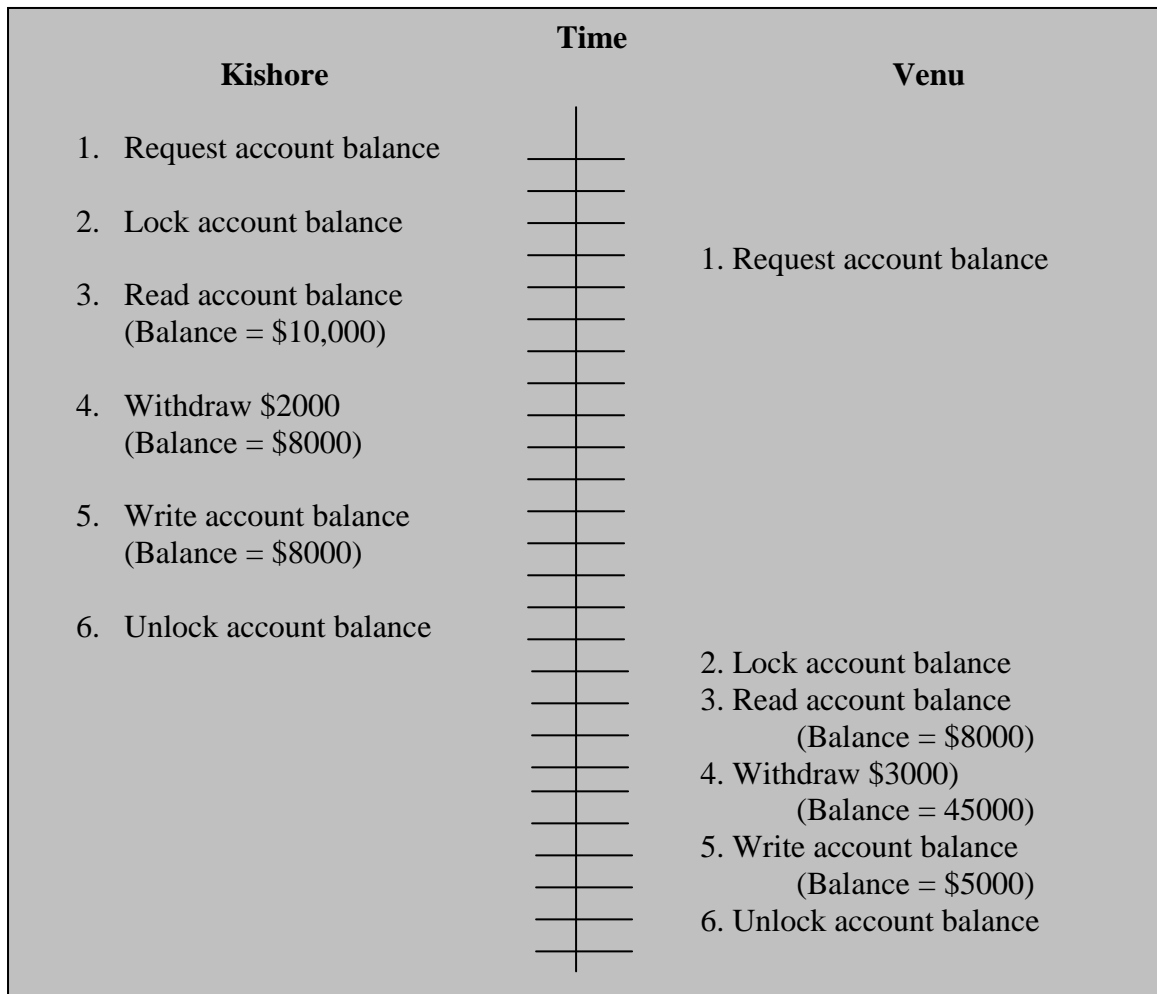
- ❖ If two transactions only read a data item, they do not conflict and the order is not important.
- ❖ If two transactions either read or write completely separate data items, they do not conflict and the execution order is not important.
- ❖ If one transaction writes a data item and another either reads or writes the same data item, the order of execution is important.

LOCKING MECHANISMS (or) concurrency control with locking Mechanisms (or) Granularity:

Transactions that request data from different tables in a database will not conflict with each other and can be run concurrently without causing data integrity problems. Serializability is achieved by different means, but locking mechanisms are the most common type of concurrency control mechanism. With locking, any data that are retrieved by a user for updating must be locked, or denied to other users, until the update is complete or aborted.

Locking: *Any data that are retrieved by a user for updating must be locked, or denied to other users, until the update is completed or aborted*

Following fig shows the use of record locks to maintain data integrity. Kishore initiates a withdrawal transaction from an ATM. Since Kishore's transaction will update this record, the application program locks this record before reading it into main memory. Kishore proceeds to withdraw \$2000, and the new balance (\$8000) is computed. Venu has initiated a withdrawal transaction shortly after Kishore, but his transaction cannot access the account record until Kishore's transaction has returned the updated record to the database and unlocked the record. The locking mechanism thus enforces a sequential updating process that prevents erroneous updates.



Locking Level:

An important consideration in implementing concurrency control is choosing the locking level. The locking level (also called granularity) is the extent of the database resource that is included with each lock. Most commercial products implement locks at one of the following levels:

1. **Database:** The entire database is locked and becomes unavailable to other users. This level has limited application, such as during a backup of the entire database.

2. **Table:** The entire table containing a requested record is locked. This level is appropriate mainly for bulk updates that will update the entire table, such as giving all employees a 5 percent raise.

3. **Block or page:** The physical storage block (or page) containing a requested record is locked. This level is the most commonly implemented locking level. A page will be a fixed size (4K, 8K, etc.) and may contain records of more than one type.

4. **Record level:** Only the requested (or row) is locked. All other records, even within a table, are available to other users.

5. **Field level:** Only the particular field (or column) in a requested record is locked. This level may be appropriate when most updates affect only one or two fields in a record.

Types of Locks

In reality, the database administrator can generally choose between **two types of locks: shared and exclusive.**

1. Shared locks:

Shared locks (also called S locks, or read locks) allow other transactions to read (but not update) a record or other resource. A transaction should place a shared lock on a record or data resource when it will only read but not update that record. Placing a shared lock on a record *prevents* another user from placing an exclusive lock, but not a shared lock, on that record.

2. Exclusive locks:

Exclusive locks (also called X locks, or write locks) prevent another transaction from reading (and therefore updating) a record until it is unlocked. A transaction should place an exclusive lock on a record when it is about to update that record. Placing an exclusive lock on a record prevents another user from placing any type of lock on that record.

DEADLOCKS

A deadlock occurs when two transactions wait for each other to unlock data. For example a deadlock occurs when two transactions, T1 and T2, exist in the following mode:

T1 = access data items X and Y
T2 = access data items Y and X

If T1 has not unlocked data item Y, T2 cannot begin; if T2 has not unlocked data item X, T1 cannot continue. Consequently, T1 and T2 wait indefinitely, each waiting for the other to unlock the required data item. Such a deadlock is also known as a **deadly embrace**.

In a real-world DBMS, many more transactions can be executed simultaneously, thereby increasing the probability of generating deadlocks.

The two basic techniques to control deadlocks are:

- ❖ **Deadlock prevention:** User programs must lock all records they require at the beginning of a transaction (rather than one at a time)
- ❖ **Deadlock detection:** The DBMS periodically tests the database for deadlocks. If a deadlock is found, one of the transactions (the "victim") is aborted (rolled back and restarted) and the other transaction continues.

TWO-PHASE LOCKING TO ENSURE SERIALIZABILITY

Two-phase locking defines how transactions acquire and relinquish locks.

Two-phase locking guarantees serializability, but it does not prevent deadlocks.

The two phases are:

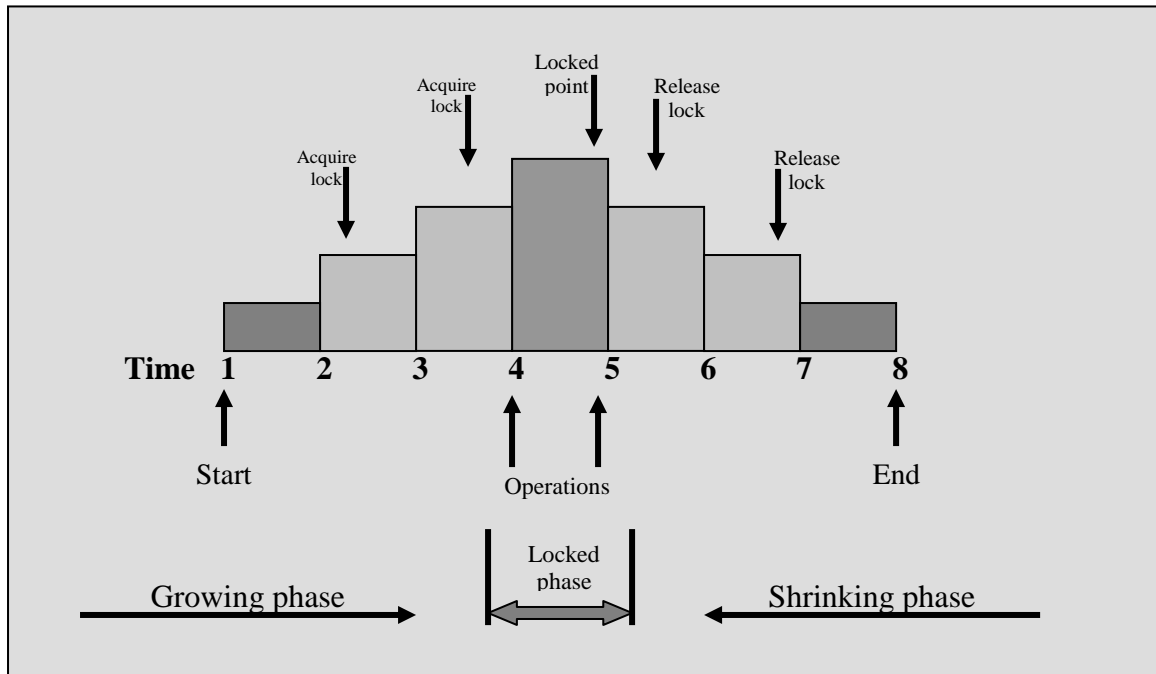
1. A **growing phase**, in which a transaction acquires all required locks without unlocking any data. Once all locks have been acquired, the transaction is in its locked point.
2. A **shrinking phase**, in which a transaction releases all locks and cannot obtain any new lock.

The two-phase locking protocol is governed by the following rules:

- ❖ Two transactions cannot have conflicting locks.

- ❖ No unlock operation can precede a lock operation in the same transaction.
- ❖ No data are affected until all locks are obtained—that is, until the transaction is in its locked point.

Two-phase locking protocol



CONCURRENCY CONTROL WITH TIME TAMPING METHODS

The **time stamping approach to scheduling concurrent transactions assigns a global, unique time stamp to each transaction.** The time stamp value produces an explicit order in which transactions are submitted to the DBMS. **Time stamps must have two properties: uniqueness and monotonicity.**

Uniqueness endures that no equal time stamp values can exist, and **monotonicity** ensures that time stamp values always increase.

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

WAIT / DIE AND WOUND / WAIT SCHEMES

Time stamping methods are used to manage concurrent transaction execution. Two schemes used to decide which transaction is rolled back and which

continues executing: the wait / die scheme and the wound / wait scheme. As example illustrates the difference. Assume that you have two conflicting transactions T1 and T2, each with a unique time stamp. Suppose T1 has a time stamp of 11548789 and T2 has a time stamp of 19562545. You can deduce from the time stamps that T1 is the older transaction (the lower time stamp value) and T2 is the newer transaction. Given that scenario, the four possible outcomes are shown in below table.

Wait / die and Wound / Wait Concurrency Control Schemes

TRANSACTION REQUESTING LOCK	TRANSACTION OWNING LOCK	WAIT / DIE SCHEME	WOUNDE / WAIT SCHEME
T1 (11548789)	T2 (19562545)	<ul style="list-style-type: none"> • T1 waits until T2 is completed and T2 releases its locks. 	<ul style="list-style-type: none"> • T1 preempts (rolls back) T2. • T2 is rescheduled using the same time stamp.
T2 (19562545)	T1 (11548789)	<ul style="list-style-type: none"> • T2 dies (rolls back). • T2 is rescheduled using the same time stamp. 	<ul style="list-style-type: none"> • T2 waits until T1 is completed and T1 releases its locks.

- ❖ **In the wit / die scheme, the older transaction waits and the younger is rolled back and rescheduled.**
- ❖ **In the wound / wait scheme, the older transaction rolls back the younger transaction and reschedules it.**

CONCURRENCY CONTROL WITH OPTIMISTIC METHODS

The **optimistic approach is based on the assumption that the majority of the database operations do not conflict.** The optimistic approach does not require locking or time stamping techniques. Instead, a transaction is executed without restrictions until it is committed. Using an optimistic approach, each transaction moves through two or three phases. The phases are read, validation, and write.

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

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

DATABASE RECOVERY MANAGEMENT

Database recovery restores a database from a given state, usually inconsistent, to a previously consistent state. Recovery techniques are based on the atomic transaction property: all portions of the transaction must be treated as a single, logical unit of work in which all operations are applied and completed to produce a consistent database. If, for some reason, any transaction operation cannot be completed, the transaction must be aborted and any changes to the database must be rolled back (undone). In short, transaction recovery reverses all of the changes that the transaction made to the database before it was aborted.

Backup and recovery functions constitute a very important component of today's DBMSs. Some DBMSs provide functions that allow the database administrator to schedule automatic database backups to permanent secondary storage devices such as disks and tapes.

The level of backup varies:

- ❖ A **full backup** of the database, or dump of the database.
- ❖ A **differential backup** of the database, in which only the last modifications to the database (when compared with a previous backup copy) are copied.
- ❖ A **transaction log backup**, which backs up only the transaction log operations that are not reflected in a previous backup copy of the database.

The database backup is stored in a secure place, usually in a different building and is protected against dangers such as fire, theft, flood, and other potential calamities.

Failures that plague databases and systems are generally induced by software, hardware, programming exemptions, transactions, or external factors.

SOFTWARE

Software-included failures may be traceable to the operating system, the DBMS software, application programs, or viruses.

HARDWARE

Hardware-induced failures may include memory chip errors, disk crashes, bad disk sectors, and disk full errors.

PROGRAMMING EXEMPTIONS

Application programs or end users may roll back transactions when certain conditions are defined. For example, a recovery procedure may be initiated when a withdrawal of funds is made when the customer balance is zero or when an end user has initiated an unintended keyboard error, such as pressing Ctrl+C while running UNIX application that updates a database at the shell (command prompt) level.

TRANSACTIONS

The system detects deadlocks and aborts one of the transactions.,

EXTERNAL FACTORS

Backups are especially important when a system suffers complete destruction due to fire, earthquake, flood, or other natural disaster.

TRANSACTION RECOVERY

Database transaction recovery focuses on the different methods used to recover a database from an inconsistent to a consistent state by using the data in the transaction log.

Four *important concepts that affect the recovery process*:

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

- ❖ **Redundant transaction logs.** Most DBMSs keep several copies of the transaction log to ensure that a physical is failure will not impair the DBMS's ability to recover data.
- ❖ **Database buffers.** A buffer is a temporary storage area in primary memory used to speed up disk operations. To improve processing time, the DBMS software reads the data from the physical disk and stores a copy of it on a "buffer" in primary memory. When a transaction updates data, it actually updates the copy of the data in the buffer because that process is much faster than accessing the physical disk every time.
- ❖ **Database checkpoints.** A database checkpoint is an operation in which the DBMS writes all of its updated buffers to disk. While this is happening, the DBMS does not execute any other requests. A checkpoint operation is also registered in the transaction log.

The recovery processes will follow these steps:

1. *Identify the last checkpoint in the transaction log.* This is the last time transaction data were physically saved to disk.
2. *For a transaction that started and committed before the last checkpoint,* nothing needs to be done because the data are already saved.
3. *For a transaction that committed after the last checkpoint,* the DBMS redoes the transaction, using the "after" values of the transaction log. Changes are applied in ascending order, from oldest to newest.
4. *For any transaction that had a ROLLBACK operation after the last checkpoint or that was left active (with neither a COMMIT nor a ROLLBACK) before the failure occurred,* the DBMS uses the transaction log records to ROLLBACK or undo the operations, using the "before" values in the transaction log. Changes are applied in reverse order, from newest to oldest.

UNIT-5

DDBMS and Database Administration

THE NEED FOR DATA ANALYSIS

Data analysis can provide information about short-term tactical evaluations and strategies such as these: Are our sales promotions working? What market percentage are we controlling? Are we attracting new customer? Tactical and strategic decisions are also shaped by constant pressure from external and internal forces, including globalization, the cultural and legal environment, and (perhaps most importantly) technology.

DECISION SUPPORT SYSTEMS

Decision support is a methodology (or a series of methodologies) designed to extract information from data and to use such information as a basis for decision making.

A decision support system (DSS) is an arrangement of computerized tools used to assist managerial decision making within a business. A DSS usually requires extensive data "massaging" to produce information.

A DSS is used at all levels within an organization and is often tailored to focus on specific business areas or problems such as finance, insurance, healthcare,

banking, sales, and manufacturing. The *DSS is interactive and provides ad-hoc query tools to retrieve data and to display data in different formats.* The *DSS has an unquestionably important role in modern business operations,* keep in mind that the manager must initiate the decision support process by asking the appropriate questions.

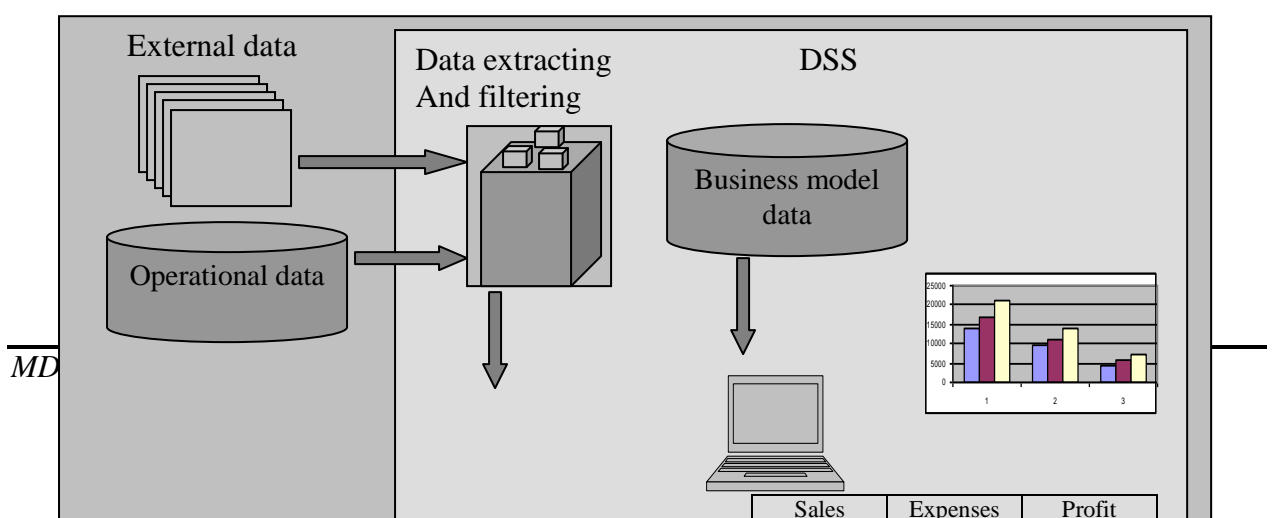
A DSS is usually composed for four main components:

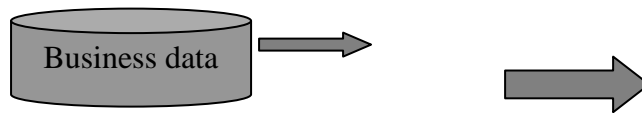
A data store component, a data extraction and data filtering component, an end-user query tool, and an end-user presentation tool.

- ❖ The **data store component** is basically a DSS database. The data store contains two main types of data: business data and business model data. The business data are extracted from the operational database and from external data sources; they represent a snapshot of the company situation. Business models are generated by special algorithms that model the business to identify and enhance the understanding of business situations and problems.
- ❖ The **data extraction and data filtering** component is used to extract and validate the data taken from the operational database and the external data sources.
- ❖ The **end-user query tool** is used by the data analyst to create the queries that access the database.
- ❖ The **end-user presentation tool** is used by the data analyst to organize and present the data. This tool helps the end user select the most appropriate presentation format, such as summary report, map, pie or bar graph, or mixed graphs. The query tool and the presentation tool are the front end to the DSS and are typically a part of the so-called *inference engine*. The inference engine uses the business model data and the business data to draw conclusions on which the decision maker can act.

Each DSS component shows below figure.

Main components of a decision support system (DSS)





DSS data differ from operational data in three main areas:

Time-span, granularity, and dimensionality.

- ❖ **Time-span.** Operational data cover a short time frame. In contrast, DSS data tend to cover a longer time frame.
- ❖ **Granularity** (level of aggregation). DSS data must be presented at different levels of aggregation, from highly summarized to near-atomic. For example, if a manager must analyze sales by region, the person must be able to access data showing the sales by region, by city within the region, by store within the city within the region, and so on.
- ❖ **Dimensionality.** Operational data focus on representing individual transactions rather than on the effects of the transactions over time. In contrast, data analysts tend to include many data dimensions and are interested in how the data related over those dimension.

Transforming operational data into decision support data

Operational Data					
	A	B	C	D	E
3	Year	Region	Agent	Product	Value
4	2004	East	Carlos	Erasers	50
5	2004	East	Tere	Erasers	12
6	2004	North	Carlos	Widgets	120
7	2004	North	Tere	Widgets	100
8	2004	North	Carlos	Widgets	30
9	2004	South	Victor	Balls	145
10	2004	South	Carlos	Balls	34
11	2004	South	Carlos	Balls	80
12	2004	West	Mary	Pencils	89
13	2004	West	Mary	Pencils	56
14	2005	East	Carlos	Pencils	45
15	2005	East	Victor	Balls	55
16	2005	North	Mary	Pencils	60
17	2005	North	Victor	Erasers	20
18	2005	South	Carlos	Widgets	30
19	2005	South	Mary	Widgets	75
20	2005	South	Mary	Widgets	50
21	2005	South	Tere	Balls	70
22	2005	South	Tere	Erasers	90
23	2005	West	Carlos	Widgets	25

	A	B	C	D	E	F
1	Year	2005				
2						
3	Sum of Value	Region				
4	Product	East	North	South	West	Total
5	Balls	55		70	100	225
6	Erasers		20	90		110
7	Pencils	45	60			105
8	Widgets			145	25	180
9	Total	100	80	315	125	620
10						
11						
12	Year	(ALL)				
13	Product	(ALL)				
14						
15	Sum of Value	Region				
16	Agent	East	North	South	West	Total
17	Carlos	95	150	60	25	300
18	Mary		60	25	145	330
19	Tere	12	100	160	100	372
20	Victor	55	20	259		334
21	Total	162	330	574	270	1336

Decision support system (DSS) data focus on a broader time-span, tend to have high levels of granularity, and can be examined in multiple dimensions. For example, note these possible aggregations:

- 1) Sales by product, region, agent, etc.
- 2) Sales for all years or only as few selected years.
- 3) Sales for all products or only a few selected products.

The differences between operational and DSS data are as follows:

- ❖ Operational data represent transactions as they happen, in real time. DSS data are a snapshot of the operational data at a given point in time.
- ❖ Operational and DSS data are different in terms of transaction type and transaction volume.
- ❖ Operational data are commonly stored in many tables, and the stored data represent the information about a given transaction only. DSS data are generally stored in a few tables that store data derived from the operational

data. The DSS data do not include the details of each operational transaction. Instead, DSS data represent transaction summaries; the DSS stores data that are integrated, aggregated, and summarized for decision support purpose.

- ❖ The degree to which DSS data are summarized is very high when contrasted with operational data.
- ❖ Query activity (frequency and complexity) in the operational database tends to be low to allow additional processing cycles for the more crucial update transactions. Queries against DSS data typically are broad in scope, high in complexity, and less speed-critical.
- ❖ DSS data are characterized by large amounts of data.

DSS DATABASE REQUIREMENTS

There are **four main requirements for a DSS database: the database schema, data extraction and loading, the end-user analytical interface, and database size.**

DATABASE SCHEMA

The DSS database schema must support complex (non-normalized) data representations. The DSS database must contain data that are aggregated and summarized. In addition to meeting those requirements, the queries must be able to extract multidimensional time slices. If you are using an RDBMS, the conditions suggest using non-normalized and even duplicated data. To see why this must be true, take a look at the 10 year sales history for a single store containing a single department. At this point, the data are fully normalized within the single table, as shown in below table:

Ten-Year Sales History for a Single Department, Millions of Dollars

YEAR	SALES
1996	8,227
1997	9,109
1998	10,1041
1999	11,553
2000	10,018
2001	11,875
2002	12,699
2003	14,875
2004	16,301
2005	19,986

The DSS database schema must also be optimized for query (read-only) retrievals.

DATA EXTRACTION AND FILTERING

The DSS database is created largely by extracting data from the operational database and by importing additional data from external sources. Thus, the DBMS must support advanced data extraction and filtering tools. The data extraction capabilities should also support different data sources: flat files and hierarchical, network, and relational databases, as well as multiple vendors. Data filtering capabilities must include the ability to check for inconsistent data or data validation rules.

END-USER ANALYTICAL INTERFACE

The end-user analytical interface is one of the most critical DSS DBMS components. When properly implemented, an analytical interface permits the user to navigate through the data to simplify and accelerate the decision making process.

DATABASE SIZE

DSS databases tend to be very large; gigabyte and terabyte ranges are not unusual. The DSS database typically contains redundant and duplicated data to improve data retrieval and simplify information generation. Therefore, the DBMS must be capable of supporting **very large databases (VLDBs)**. To support a VLDB adequately, the DBMS might be required to use advanced hardware, such as multiple disk arrays, and, even more importantly, to support multiple processor technologies, such as a symmetric multiprocessor (SMP) or a massively parallel processor (MPP).

THE DATA WAREHOUSE

Bill Inmon, the "Father of the data warehouse", **defines the term as "an integrated, subject oriented, time variant, nonvolatile collection of data that provides support for decision making."**

- ❖ **Integrated.** The data warehouse is a centralized, consolidated database that integrates data derived from the entire organization and from multiple sources with diverse formats. Data integration implies that all business entities, data elements, data characteristics, and business metrics are described in the same way throughout the enterprises. For instance, the status of an order might be indicated with text labels such as "open", "received", "cancel", and "closed" in one department and as "1", "2", "3" and "4" in another department. A student's status might be defined as

"freshman", "sophomore", "junior", or "senior" in the accounting department and as "FR", "SO", "JR" or "SR" in the computer information systems department. To avoid the potential format tangle, the data in the data warehouse must conform to a common format acceptable throughout the organization.

- ❖ **Subject oriented.** Data warehouse data are arranged and optimized to provide answers to questions coming from diverse functional areas within a company. Data warehouse data are organized and summarized by topic, such as sales, marketing, finance, distribution, and transportation, for each topic, the data warehouse contains specific subjects of interest-products, customers, departments, regions, promotions, and so on. This form of data organization is quite different from the more functional or process-oriented organization of typical transaction system. For example, an invoicing system designer concentrates on designing normalized data structures (relational tables) to support the business process by storing invoice components in two tables: INVOICE and INVLIN. In contrast, the data warehouse has a subject orientation. Data warehouse designers focus specifically on the data rather than on the processes that modify the data.
- ❖ **Time variant.** In contrast to operational data, which focus on current transactions, warehouse data represent the flow of data through time. The data warehouse can even contain projected data generated through statistical and other models. It is also time variant in the sense that once data are periodically uploaded to the data warehouse, all time-dependent aggregations are recomputed. For example, when data for previous weekly sales are uploaded to the data warehouse, the weekly, monthly, yearly, and other time dependent aggregates for products, customers, stores, and other variables are also updated. The data warehouse contains a time ID that is used to generate summaries and aggregations by week, month, quarter, year, and so on. Once the data enter the data warehouse, the time ID assigned to the data cannot be changed.
- ❖ **Nonvolatile.** Once data enter the data warehouse, they are never removed. Because the data in the warehouse represent the company's history, the operational data, representing the near term history, are always added to it. Because data are never deleted and new data are continually added, the data

warehouse is always growing. That's why the DSS DBMS must be able to support multi-gigabyte and even multi-terabyte databases and multiprocessor hardware.

TWELVE RULES THAT DEFINE A DATA WAREHOUSE

In 1994, William H. Inmon and Chuck Kelley created 12 rules defining a data warehouse, which summarize many of the points made in this chapter about data warehouses.

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

These 12 rules capture the complete data warehouse life cycle.

ONLINE ANALYTICAL PROCESSING

The need for **more intensive decision support prompted the introduction of a new generation of tools.** Those new tools Called **online analytical processing (OLAP)**, create an advanced data analysis environment that supports decision making, business modeling, and operations research. **OLAP systems share four main characteristics;** they

- ❖ Use multidimensional data analysis techniques.
- ❖ Provide advanced database support.
- ❖ Provide easy-to-use end-user interface.
- ❖ Support client / server architecture.

MULTIDIMENSIONAL DATA ANALYSIS TECHNIQUES

The most distinct characteristic of modern OLAP tools is their capacity for multidimensional analysis. In multidimensional analysis, data are processed and viewed as part of a multidimensional structure. This type of data analysis is particularly attractive to business decision makers because they tend to view business data as data that are related to other business data.

Multidimensional data analysis techniques are augmented by the following functions:

- ❖ **Advanced data presentation functions:** 3-D graphics, pivot tables, cross tabs, data rotation, and three dimensional cubes.
- ❖ **Advanced data aggregation, consolidation, and classification functions** that allow the data analyst to create multiple data aggregation levels. Aggregating data across the time dimension (by week, month, quarter, and year) allows the data analyst to drill down and roll up across time dimensions.
- ❖ **Advanced computational functions:** business-oriented variables (market share, period comparisons, sales margins, product margins, and percentage changes), financial and accounting ratios (profitability, overhead, cost allocations, and returns), and statistical and forecasting functions. These functions are provided automatically, and the end user does not need to redefine their components each time they are accessed.

Operational vs. Multidimensional View of Sales

Database name: Ch13_Text					
Table name:DW_INVOICE					
		INV_NUM	INV_DATE	CUS_NAME	INV_TOTAL
	+	2034	15-May-06	Dartonic	1400.00
	+	2035	15-May-06	Summer Lake	1200.00
	+	2036	1615/2006	Dartonic	1350.00
	+	2037	1615/2006	Summer lake	3100.00
	+	2038	1615/2006	Trydon	400.00

Table name:DW_LINE							
		INV_NUM	INV_NUM	CUS_NAME	INV_PRICE	LINE_QTY	LINE_AMOUNT
	+	2034	1-Jan-00	Optical Mouse	45.00	20	900.00
	+	2034	2-Jan-00	Wireless RF remote and laser pointer	50.00	10	500.00
	+	2035	1-Jan-00	Everlast Hard Drive, 60 GB	200.00	6	1200.00
	+	2036	1-Jan-00	Optical Mouse	45.00	30	1350.00
	+	2037	1-Jan-00	Optical Mouse	45.00	10	450.00
	+	2037	2-Jan-00	Roadster 56KB Ext.Modem	120.00	5	600.00
	+	2037	3-Jan-00	Everlast Hard Drive, 60 GB	205.00	10	2050.00
	+	2038	1-Jan-00	No.Tech Speaker Set	50.00	8	400.00

Multidimensional View of Sales

Customer Dimention	Time Dimension		Totals
	15-May-06	16-May-06	
Dartonic	\$1,400.00	\$1,350.00	\$2,750.00
Summer Lake	\$1,800.00	\$3,100.00	\$4,900.00
Trydon		\$400.00	\$400.00
Totals	\$3,200.00	\$4,850.00	\$8,050.00

Sales are located in the intersection Of a customer row and time column

Aggregations are provided for both dimensions

ADVANCED DATABASE SUPPORT

The deliver efficient decision support. OLAP tools must have advanced data access features. Such features include:

- ❖ Access to many different kinds of DBMSs, flat files, and internal and external data sources.
- ❖ Access to aggregated data warehouse data as well as to the detail data found in operational databases.
- ❖ Advanced data navigation features such as drill-down and roll-up
- ❖ Rapid and consistent query response times.

EASY-TO-USE END-USER INTERFACE

Advanced OLAP features become more useful when access to them is kept simple. OLAP tool vendors learned this lesson early and have equipped their sophisticated data extraction and analysis tools with easy-to-use graphical interfaces. This familiarity makes OLAP easily accepted and readily used.

CLIENT / SERVER ARCHITECTURE

Client / Server architecture provides a framework within which new systems can be designed, developed, and implemented.