

Sree Siddaganga Education Society ®Estd: 1966-67

Sree Siddaganga College of Arts Science & Commerce

Affiliated to Tumkur University

Admission for Both Boys & Girls

A College with a difference-**NAAC B++**

B.H.Road, Tumkur-572102.Ph:0816-2278569, 8277338148

Website: - www.sscasc.in

e-mail:-principal.sscasc@gmail.com



STUDY MATERIAL



Subject:-Database Management System

DEPARTMENT OF COMPUTER SCIENCE

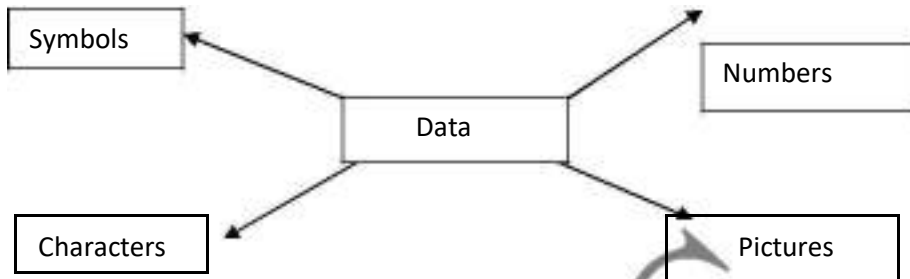
2 Semester BCA

Introduction to DBMS

Chapter 1

Data:

It is a collection or representation of facts, figures and statistics etc which does not have particular meaning. Data is in the form of numbers, characters, symbols, pictures also.



Information:

It is a processed data or collection of data which is having perfect meaning.

Data [raw facts]-> data Transformation->Information.

Database:

Database is an organized collection of related data for one or more purposes. The

symbol to identify database in DBMS is



Field:

It is a single piece of information across vertical manner.

Record:

Collection of fields is known as one record.

File:

Collection of records is known as one file.

DBMS:

It is a system software or package. It allows access to the data in the database.

It also deletes the data from the database and done any type of the modification

Traditional File Processing System

In this approach each application will maintain separate master file. This approach has duplication of data is the main theme.

Limitation of this file based approach is that the program will be dependent on the files. It will lead inconsistency.

Merits/Advantages:

1. No need of highly technical person to handle the database.
2. Processing speed is high compared to the DBMS approach.
3. No need of external storage.

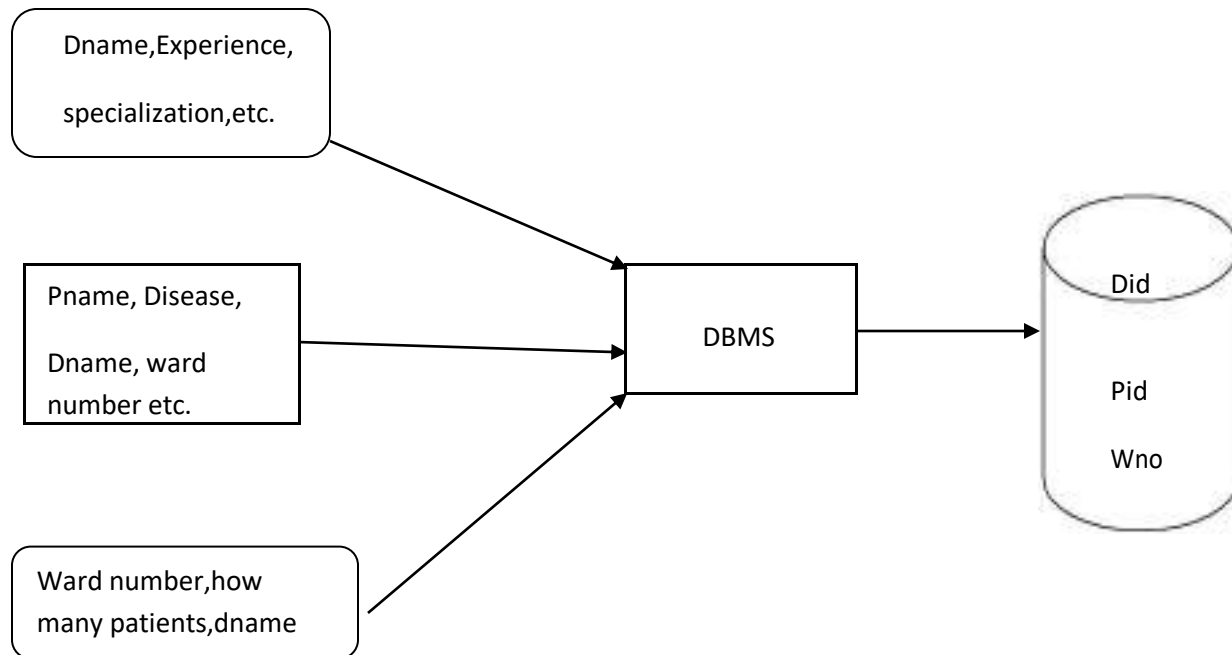
Demerits/Disadvantages:

1. **Duplication of data: Redundancy:**-It means the information will be repeated one or more times.
In the above figure doctor's details will be present not only in the doctor's file but also in patient and ward files. This mechanism will be applied on both patient and wards files. Hence same data will be repeated again and again.
2. **Data dependence:**-Due to the redundancy it will lead to the data dependence. It means one file information is totally dependent upon another file information.
In the above figure without knowing doctor's details we can't do anything on the patient and ward files. The same mechanism will be applicable on both patient and ward files. Hence data will be dependent on each other.
3. **Data inconsistency:**-Due to the redundancy it will lead to the data inconsistency. Means data is not an ordered.
4. **Limited data sharing:**-Each application has its own private files. In this users have little opportunity to share the Data from outside applications.
5. **Lack of security:** In this approach Unable to protect data from unauthorized persons. Because this approach does not have any mechanism to check the identification of the users who are authorized to that application.
6. **Lengthy development time:** It will take more time to develop several numbers of applications. It will take more time.
7. **Concurrent access anomalies:**- In this approach number of users can not access the system at the same time. Means it does not support multiple users at a time.

These drawbacks are overcome by using database approach.

Data base Approach

It is a single storage of data it is maintained for so many users. That storage is known as Repository.



As shown in the above figure the DBMS is central system which provides a common interface between the data and end users.

Advantages:

1. **Minimum data redundancy:**-In this approach we will maintain the database for storing different (unique) details of different users. Hence data present in one file need not be duplicated in other. It will reduce the redundancy of data.
2. **Data Consistency:**-Reduced data redundancy leads to better data consistency.
3. **Data Sharing:**-Related data can be shared through dbms. Means doctor share his data with other doctor through database.
4. **Data Independence:**-Here doctor details are not depends upon the patient, patient does not depends upon the ward and so on. Because each component must be know others data.
5. **Data Integrity:**-In this approach data integrity (security) is more compared to TFS.
6. Maintenance is easy due to centralized database system.

Disadvantages:

1. Cost of hardware and software.
2. Need for external database.
3. Database damage.
4. Highly dependent on DBMS operation.
5. Need technical staff to handle database

Difference between TFS and Database approach:

In TFS approach each user defines and implements their own files for a specific application to run where as database approach maintains a single repository to store multiple users' data.

Applications of DBMS:

- Storage and reproduction of graphic images, video and audio.
- Building of the virtual company office.
- Building of internet shops and distributed information systems.
- Creation of web sites, allotted to unlimited opportunities.

Characteristics of DBMS:

- **Self describing nature of a database system:** This is the fundamental characteristic of database. A DBMS catalog stores the description of a particular database.

- **Data abstraction:**

The characteristic that allows program-data independence and program-operation independence is called as data abstraction.

- **Support of multiple views of the data:**

Each user may see a different view of the database, which describes only the data interest to that user. View may be a subset of the database or it contains virtual data.

- **Sharing of data and multi-user transaction processing:**

It allows a set of concurrent users to retrieve from and to update the database.

- **Insulation between programs and data:**

In TFS the structures of data files is embedded in the accessing programs, so many changes to the structure of a file may require changing all programs that access this file.

DBMS Users

Users may be divided into 2 types. They are

- Actors
- Workers

Those who actually use and control the content called **Actors**.

Those who enable the database to be developed and designed the DBMS software and implemented called **Workers**.

The overall aim of the database is to implement the concept of data abstraction and data independence. For this we need some data base users. They are

1. **Application programmers:**

This type of users has a complete knowledge about the existing database. These users can operate on the database.

2. **Sophisticated users:**

These users interact with the system using query based language. These people having knowledge about database design.

Ex: scientists, engineers, system Analyst etc.

3. **Stand alone End users:**

This type of users who are specialized to access personal databases and also small applications.

4. **Naïve end users or Parametric Users:**

These types of users cannot have the alteration ideas but they focus to get the output of existing application to satisfy customer needs. These people frequently access the data base.

5. **Casual end users:**

These types of users can have little bit knowledge about the operations. These people occasionally access the data base. These users are also bothered about only output but not the designing part

Database Administrator [DBA]

The person who is responsible for controlling and coordinating the system is called DBA.

Functions or roles of DBA:

- Evaluate oracle features and related products.
- Installation configuration and upgrading of oracle software.
- Establish and maintain backup and recovery.
- DBA is responsible for modifications such as inserting data, delete and update data.
- DBA also creates a set of definition which are responsible for actual storage structure.
- The DBA responsible for creation of user groups it means various view level schemas.
- Providing data security.

Roles and responsibilities of DBA

1. Selection of Hardware and Software:

- Keep up with current technological trends.
- Predict future changes.
- Establish self products.

2. Managing data Security:

- Establishment of user requirements.
- Firewalls for network security.
- Protection of data against accidental or intentional or miss use.

3. Managing Data Integrity:

- Integrity controls, protect data from unauthorized users.
- Maintains data consistency.
- Maintaining data relationship.

4. Data Back up:

- **DBA** must assume what type of data should be backed up more frequently.
- **There** are several back up facilities done by DBA they are Automatic dump, periodic back up etc.

5. Database Recovery:

- Re installation of database after crash. Recovery includes back up, check point these are done by recovery manager.

6. DBA is responsible for improving query processing performance and maintain documentation standards.

7. He is responsible for version control and technical support.

8. Take care of database design and implementation.

9. Checking memory and CPU usage.

When not to use a DBMS:

- Main cost of using a DBMS, it means high initial investment in hardware, software etc.
- When a DBMS may be unnecessary, if database and applications may not be met because of DBMS overhead.
- If the database users need special operations not supported by the DBMS.

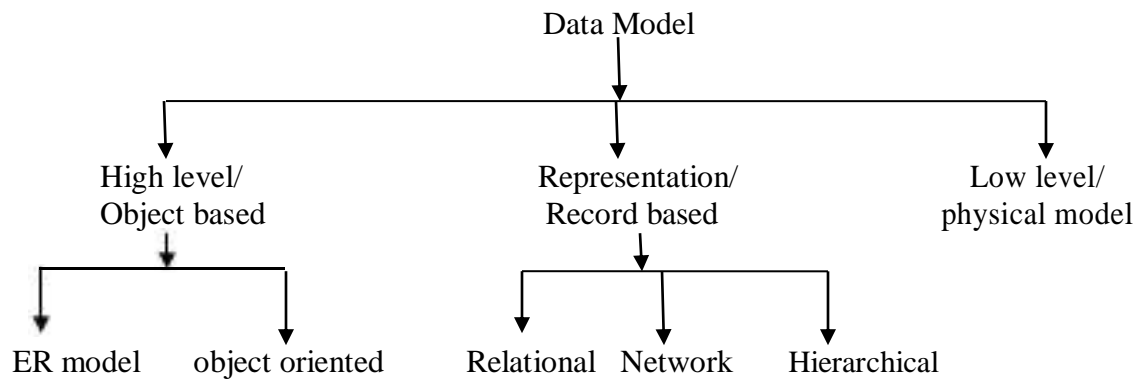
Database System Concepts and Architecture

CHAPTER 2

Definition:

Data model is a set of data structure and conceptual tools used to describe the structure of a database.

Classification of Data models



High level or object based data model:

This model is used to describe the data. In this model we do not bother about the storage of data. “The main aim of this model is only the relationship between the various objects”.

There are two types

1. E-R Model
2. Object Model

E-R Model: This is entity relationship model.

“It is a graphical or symbolical or pictorial notation for data base”.

The main components of this model are

- Entity.
- Relationship.
- Attribute.

Entity:

It is the real world object/concept like place, person or thing.

Entity can be classified into 2 types. They are

- Strong entity
- Weak entity

Strong entity is an entity which will not depends on the other entity. It will be represented by Weak entity is an entity which will depends on the other entity. It will be represented by

Representation/ Record Based model

There are 3 types of record based models. They are

1. **Hierarchical database model**
2. **Network database model**
3. **Relational database model**

Relational database model

- ❖ This model was developed by E. F CODD in 1960's.
- ❖ In this model all the data is represented (or) organized in the form of two-dimensional format that is table.
- ❖ Table is a combination of unnamed rows and Named columns (or) is a collection of records.
- ❖ Rows are also called as tuples or records.
- ❖ Columns are also called as fields or attributes.

Cols (or) fields (or) attributes

SID	SNAME	COURSE	FEE	SEM
1	Aaa	Bca	1000	2 nd
2	Bbb	Bsc	2000	2 nd
3	Bbb	bca	1000	2 nd

Body of relation

Terminology:

Domain:

A set of unique values in a particular column is referred as domain.

Ex: in the above relation SID column is the domain rest of the columns have duplicate values.

Cardinality of Relation:

“Number of rows presented in that particular relation is referred as cardinality of relation”.

Ex: cardinality of above relation is 3.

Degree of Relation:

“Number of columns presented in that particular relation is referred as degree of relation”.

Ex: degree of above relation is 5.

ADVANTAGES:

- Compared to network and hierarchical models creation of relation will be easy.
- In this model changes in the structure do not affect the data access.
- It is a very powerful and flexible model for every user.

DISADVANTAGES:

- In this we hide the implementation complexity and data storage details from the users.
- Using this model we cannot implement reference concept.
- It is poorly designed database system means every user can implement the model.

Hierarchical database model



In this model data will be represent in the form of inverted tree like structure.

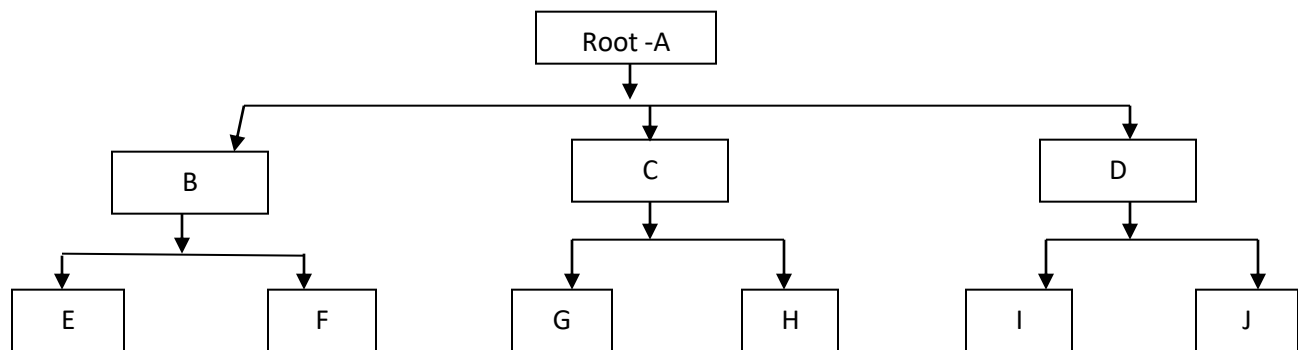


This structure allows representing information using parent-child relationship.



Each parent can have many children. But each child has only one parent. Hence, this relationship is called as one to many relationships

This model is used to describe the data and maintain the logical and view level at the top of the hierarchy, there is one entity, which is called the root.



ADVANTAGES:

- In this model DBMS provides lot of security.
- There is always link between the parent and child.
- Proper ordering of a tree result is easier and faster.

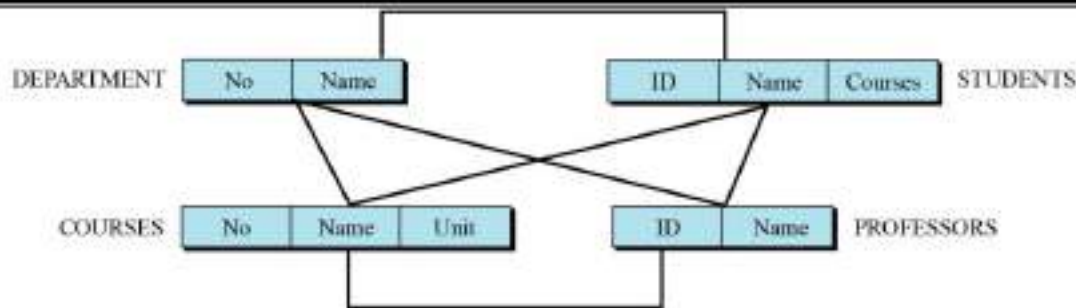
DISADVANTAGES:

- Although it is a simple to design but complex to implement.
- If you make any changes in the database structure then you must make changes in the entire application in the DBMS.
- This model suffers from the insert, delete and update anomalies.

Network database model

- ❖ It is a flexible of representing objects and their relationships.
- ❖ In this data can be representing in the form of relationship through the pointers.
- ❖ This model allows each record have multiple parent and child records, hence this is referred as many to many relationship.
- ❖ It is as same as the hierarchical model but need not necessary to use a downward tree like structure to represent the data.

In the network model, the entities are organized in a graph, in which some entities can be accessed through several paths as shown in figure below.



ADVANTAGES:

- Data must be tree like structure because of we are using the pointers.
- Data manipulation can be done easily.
- It is possible to represent many to many relationship

DISADVANTAGES:

- It is very complex to implement the pointer concept.
- It is very difficult to make the structural changes in the database.

Data base Schema and Instance

Schema:- Schema is an overall structure or description of a database, which does not have any content in it.

Sno	Sname	Course	Sem

Instance:

The actual content of database at a particular point at a time is referred as an Instance.

Sno	Sname	Course	Sem
1	Aaa	bca	3rd
2	Bbb	bsc	2nd

} → Instance

DBMS Architecture

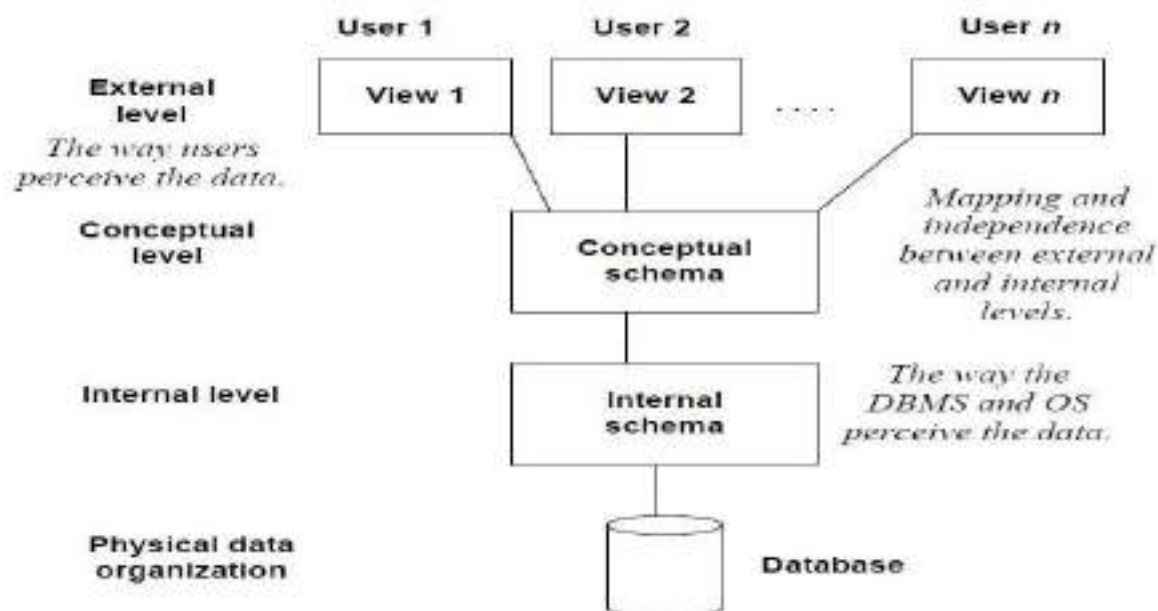
There are 2 different ways to look at the architecture of DBMS they are

1. Logical Architecture
2. Physical Architecture.

Logical DBMS Architecture / 3-Schema Architecture / 3 level Architecture

- This architecture describes how the data transfers from database to users through the DBMS.
- The aim of this architecture is separation of user application from the database.
- This Architecture divided into 3 levels hence it is referred as 3 schemas. Those levels are
 - Internal or physical or low level
 - Conceptual or logical or middle level
 - View or External or high level

All these levels are shown in the following figure.



Internal level:

- It is the lowest level of data abstraction that deals with the physical representation of the database on the computer and thus, is also known as physical level.
- It describes how the data is physically stored and organized on the storage medium.
- In this level users can create only empty structure of the database.

Conceptual level:

- This level of abstraction deals with the logical structure of the entire database and thus, is also known as logical level.
- It describes what data is stored in the database, the relationships among the data and complete view of the user's requirements without any concern for the physical implementation.
- That is, it hides the complexity of physical storage structures. The conceptual view is the overall view of the database and it includes all types of operations like insertion, deletion and modifications.

External level:

- It is the highest level of abstraction that deals with the user's view of the database and thus, is also known as view level.
- In general, most of the users and application programs do not require the entire data stored in the database.
- It permits users to access data in a way that is customized according to their needs, so that the same data can be seen by different users in different ways, at the same time.

The process of transforming the requests and results between various levels of DBMS architecture is known as **mapping**.

There are 2 ways of mapping used in 3 schema architecture. They are

- **Internal mapping:**
The mapping between internal and conceptual level and it gives the correspondence between structure and stored data.
- **External mapping:**
The mapping between external and conceptual level and it defines the correspondence between particular records and fields.

The main advantage of three-schema architecture is that it provides **data independence**.

Data independence

“It is the ability to change the schema at one level of the database system without having to change the schema at the other levels. “

Data independence is of two types, namely,

- logical data independence
- Physical data independence.

Logical data independence:

It is the ability to change the conceptual schema without affecting the external schemas or application programs.

Physical data independence:

It is the ability to change the internal schema without affecting the conceptual or external schema. An internal schema may be changed due to several reasons such as for creating additional access structure, changing the storage structure, etc.

DBMS LANGUAGES

DBMS provides mainly 3 types of languages. They are

- **Data definition language (DDL)**

These commands are used to define the logical schema. Users can define, alter and also drop the table by using these commands.

Create, Alter, Drop, truncate

- **Data manipulation language (DML)**

These commands are used to insert data into table, delete data from the table and also update the content in the table.

Insert, Delete, Update, Select

- **Data Control language (DCL)**

Grant, Revoke,

- **Transaction Control language (TCL)**

Commit rollback

Basic Client/Server Architecture

The client/server architecture was developed to deal with computer environment in which a large number of PCs, workstation, file server...

- A client is the requesting machine and the server is the supplying machine. Both are connected through network is referred as client Server Architecture.

There are 2 types of DBMS Client-Server Architecture. They are

- Two-tier client server architecture
- Three-tier client server architecture

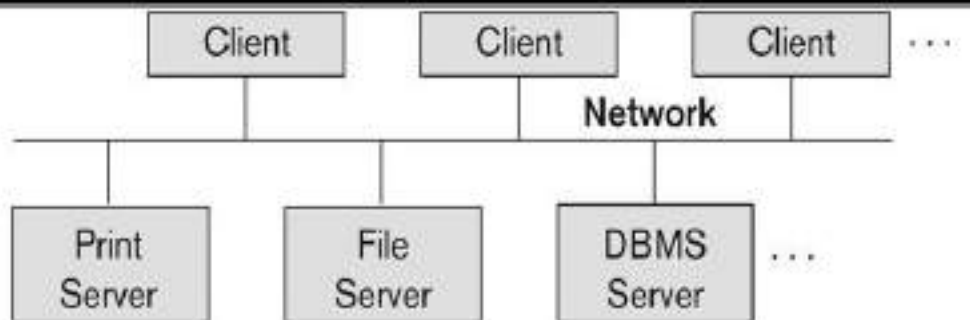
This is called **two-tire** architectures because the software components are distributed over two systems: client and server

- In this establish a connection between client and server that is ODBC (open database connectivity) once this connection is formed then the client will communicate with the server.
- Query requests are sent from the client to the server and server processors then it sends results to the client.

Advantages:

1. Clients are not powerful.
2. Reduces the data traffic on the network.
3. Improve data integrity.

Figure 2.5
Logical two-tier
client/server
architecture.

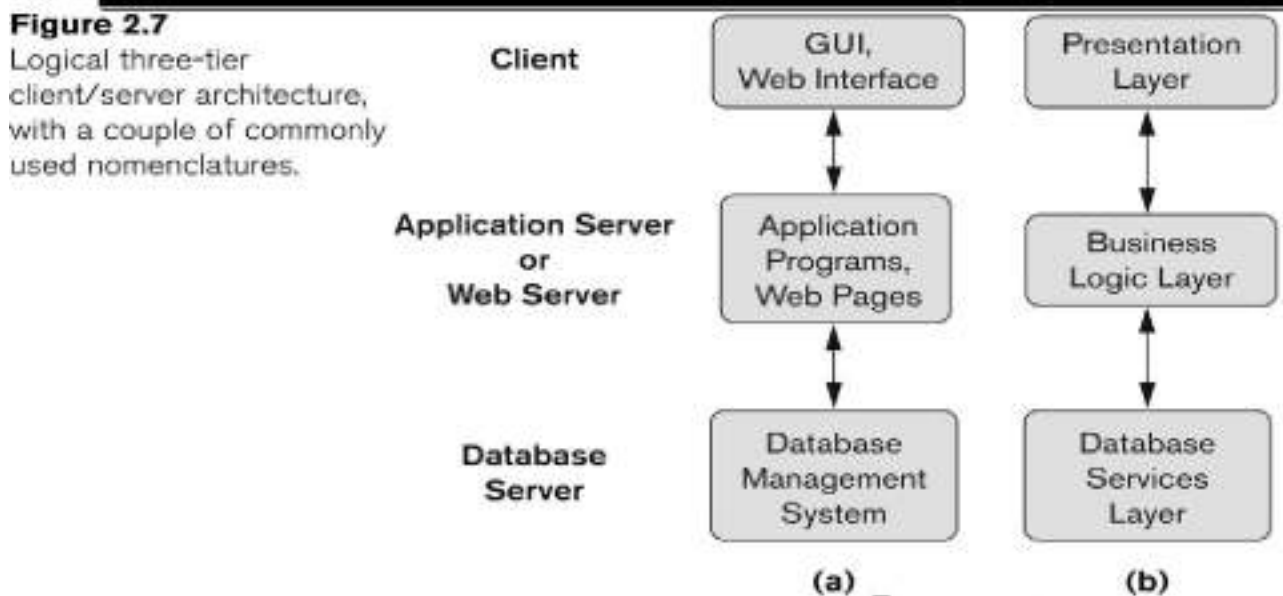


Three-tier architecture

The emergence of the **Web** changed the roles of client and server, leading to the **three-tier** architecture

The intermediate layer or **middle layer** is sometimes called the **application server** or **Web server** Three-tier Architecture Can Enhance Security:

1. Database server only accessible via middle tier
2. Clients cannot directly access database server



The presentation layer displays information to the user

- The business logic layer handles intermediate rules and constrains before data is passed up to the user or down to the DBMS
- If the bottom layer is split into two layers (a web server and a database server), then it is a **4-tier** architecture (possible to the **n-tier**)

Advantages:

1. Technological flexibility.
2. Long term cost reduction.
3. Reduced risk.

Improve customer services

DBMS Interfaces

Interfaces are the programs which convert system language to user understandable language; hence it is as same as the translator.

The user friendly interfaces provided by DBMS are

1. Menu-Based Interfaces:

These interfaces present the user with lists of options, called menus that lead the user through the formulation of a request. By using these menus users form a request through query processing.

2. Forms-Based Interfaces :

A forms-based interface displays a form to each user. Users can fill out all of the form entries to insert new data, or they fill out only certain entries, in which case the DBMS will retrieve matching data for the remaining entries.

3. Graphical User Interfaces :

A graphical interface (CUI) typically displays a schema to the user in diagrammatic form. The user can then specify a query by manipulating the diagram.

4. Interfaces for Parametric c Users :

Parametric users, such as bank tellers, often have a small set of operations that they must perform repeatedly. Usually, a small set of abbreviated commands is included, with the goal of minimizing the number of keystrokes required for each request.

5. Interfaces for the DBA: Most database systems contain privileged commands that can be used only by the DBA's staff. These include commands for creating accounts, setting system parameters, granting account authorization, changing a schema, and reorganizing the storage structures of a database.

Meta data: It is data about data; it includes data of the data types and size etc.

Meta data stores in a file that is repository or data dictionary.

Ex:
Student

Sno	Sname	course
1	A	Bca
2	B	Bsc

Meta data of above data is

Sno	sid	Number
Sname	name	Char
course	scourse	char

Classification of DBMS

DBMS can be classified into different categories on the basis of several criteria such as

1. Data models:

In this which type of data model can be used to design or define the data base.

2. Number of users support:

In this how many numbers of users and it supports which type of operating system like single or multiuser.

3. Number of sites:

Here users are used to store data in a single site or distributed site.

4. Cost:

Whether it is low end system or high end system.

5. Purpose:

Whether it may be used for general purpose or special purpose software system

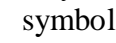
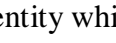
Data modeling using E-R model

CHAPTER 3

Entity: An entity is an object or thing in the real world with an independent existence.

Ex: place, person, thing etc.

Entity can be classified into two types they are

- **Strong Entity:** It is an entity which doesn't depend upon on another entity. It will be represented by  symbol.
- **Weak Entity:** It is an entity which will depends upon the key entity. It will be represented by  symbol.

Entity set:

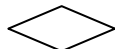
It is a collection of related entities of same type.


Entity type:

Is a collection or set of entities that have same attributes


Relationship:

It is an association between two or more entities is referred as relationship.

It can be represented by  symbol.

Attribute: It is a property or characteristic that described an entity or relationship and both. It is represented by  symbol.

Types of attributes: There are different types of attributes




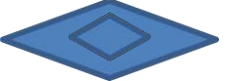


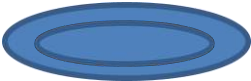
1. **Simple Attribute:** It is an Attribute which cannot be divided into sub parts. It consists of single atomic value.
Ex: Regno, age etc.
2. **Composite Attribute:**-It is an attribute which can be further divided into subparts. This value not atomic. Ex: Name, Address etc.
3. **Single valued Attribute:**-This attribute can have only a single value for particular entity.
Ex: A person can have only one date of birth, Regno etc
4. **Multi Valued Attribute:**-This Attribute can have multiple values. It is represented by 
Ex: skill, phone number etc.
5. **Derived Attribute:**-It is an attribute derived from another attribute.
It is represented by ex:
age, experience etc.
6. **Null Attribute:**-An entity may not have an applicable value for an attribute.
7. **Stored Attribute:**-It stores value of the related attribute.
8. **Key Attribute:**- It is an attribute it cannot be allow duplicate values; it can allow only unique values



Notation for E-R diagram

Diagram:

The graphical representation of entities and relationship is known as **ER Diagram**.

Symbols	Meaning
	it represents entity
	weak entity
	Relationship
	Identifying weak relationship
	Attribute
	key attribute
	Multivalued attribute

Mapping cardinalities (or) types of relationships between entity sets

Express the number of entities to which another entity can be associated via a relationship set. On the other hand, it is the cardinality of relationship that defines the number of instances of one entity as it relates to the number of instances of the other entity.

Based on the different combinations between two entities we can have either

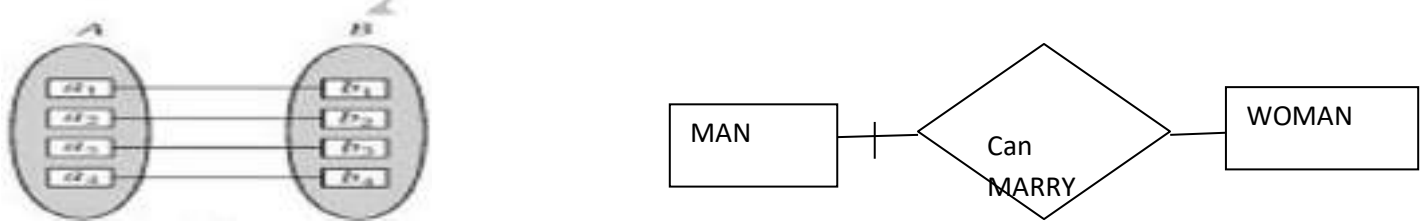
- One-to- One
- One- to- Many
- Many- to- One
- Many -to- Many

One to One Relationship:

One instance of one entity can be associates with one instance of another entity.

Ex: one man can marry with one woman.

This can be represented in the form of both set and E-R notation.

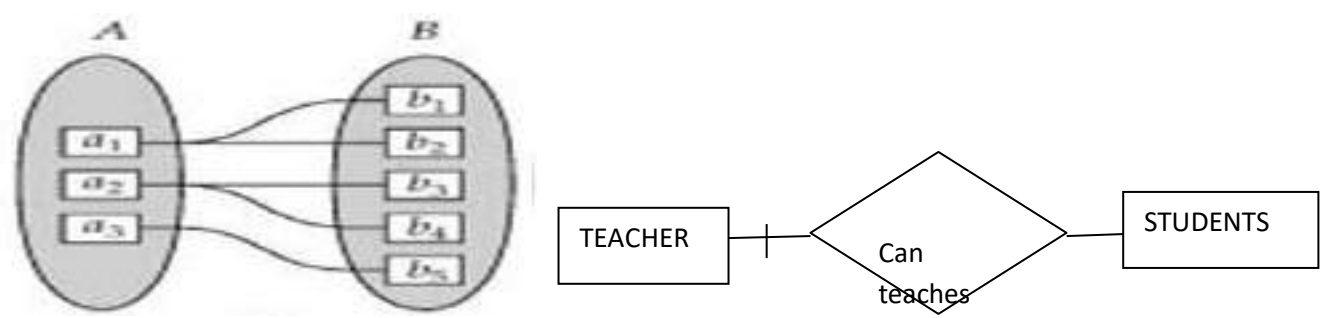


One to Many Relationships:

One instance of one entity can be associates with many instances of another entity.

Ex: one teacher can teach many students at a time.

This can be represented in the form of both set and E-R notation.

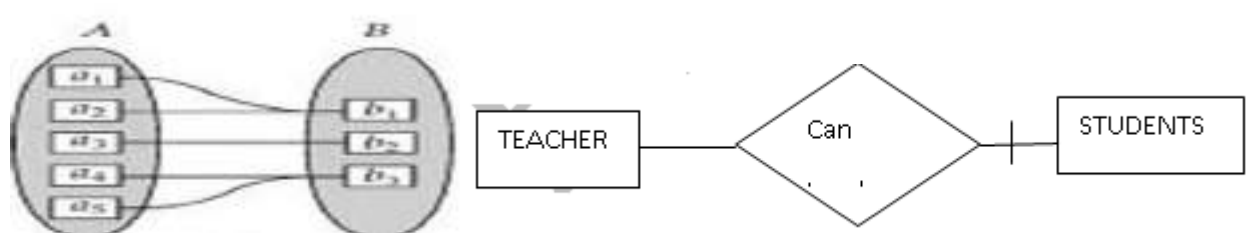


Many to One Relationship:

Many instances of one entity can be associates with one instance of another entity.

Ex: many students can learn one course at a time.

This can be represented in the form of both set and E-R notation.

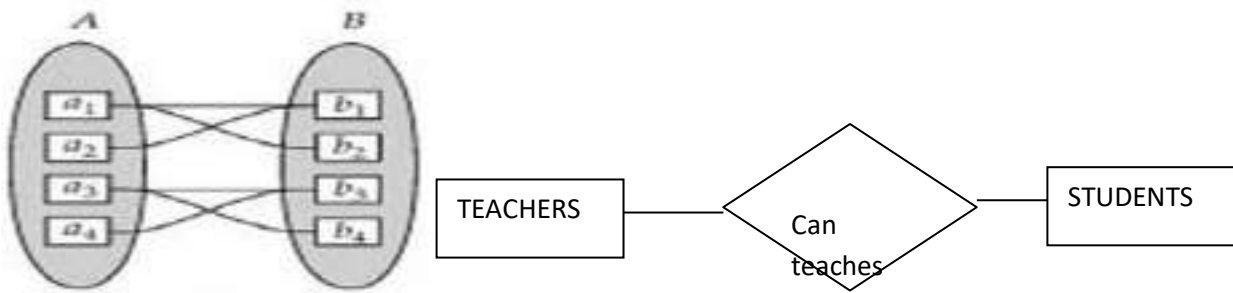


Many to Many Relationships:

Many instances of one entity can be associates many instances of another entity.

Ex: many teachers can teach many students at a time.

This can be represented in the form of both set and E-R notation.



Degree of relationship

It is process of counting number of participating entities in an E-R diagram.

There are 3 types of degree. They are

- Binary relationship degree.
- Ternary relationship degree.
- N-Array relationship degree.

Binary relationship degree:

If there are two entities involved in relationship then it is referred to as binary relationship. This type of degree is most common type of degree used in E-R diagram.

Ternary relationship degree:

If there are three entities involved then it is called as ternary relationship.

N-Array relationship degree

If there are more than 3 entities participated in E- R diagram is called as N-array relationship degree.

Data Abstraction

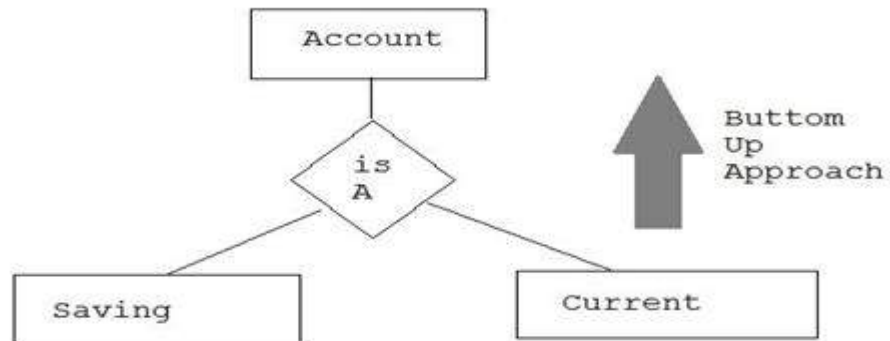
The process of hiding the irrelevant data or things from the users. It reduces the complexity and increases efficiency.

There are 3 main aspects of abstraction. They are

- Generalization
- Specialization
- Aggregation

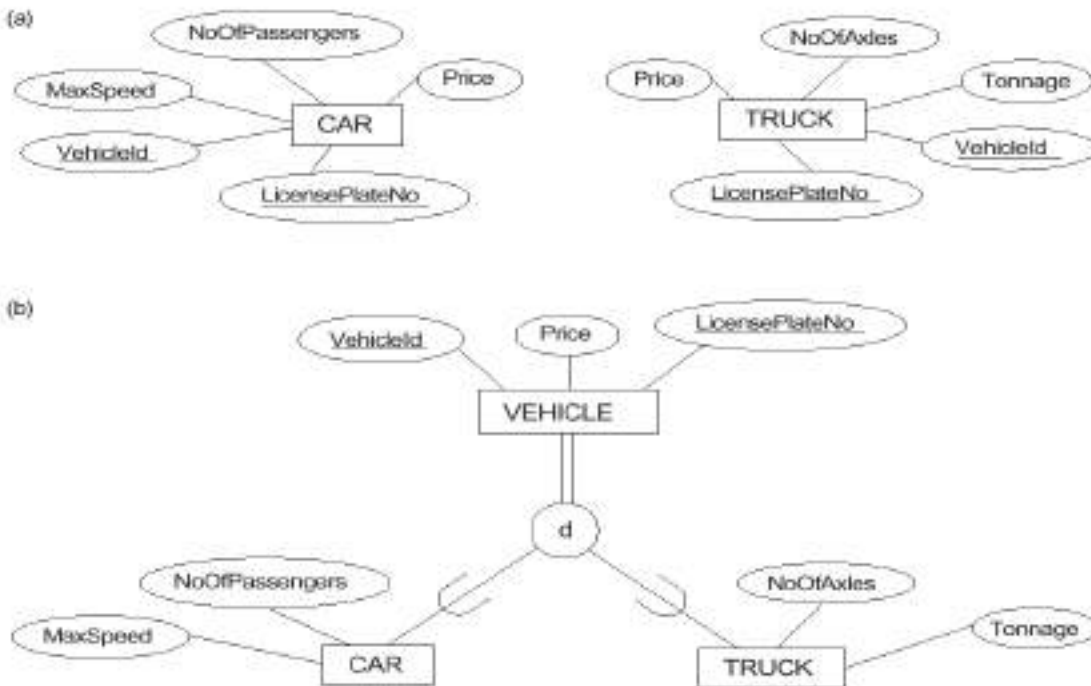
Generalization

Generalization is a bottom-up approach in which two lower level entities combine to form a higher level entity. In generalization, the higher level entity can also combine with other lower level entity to make further higher level entity.



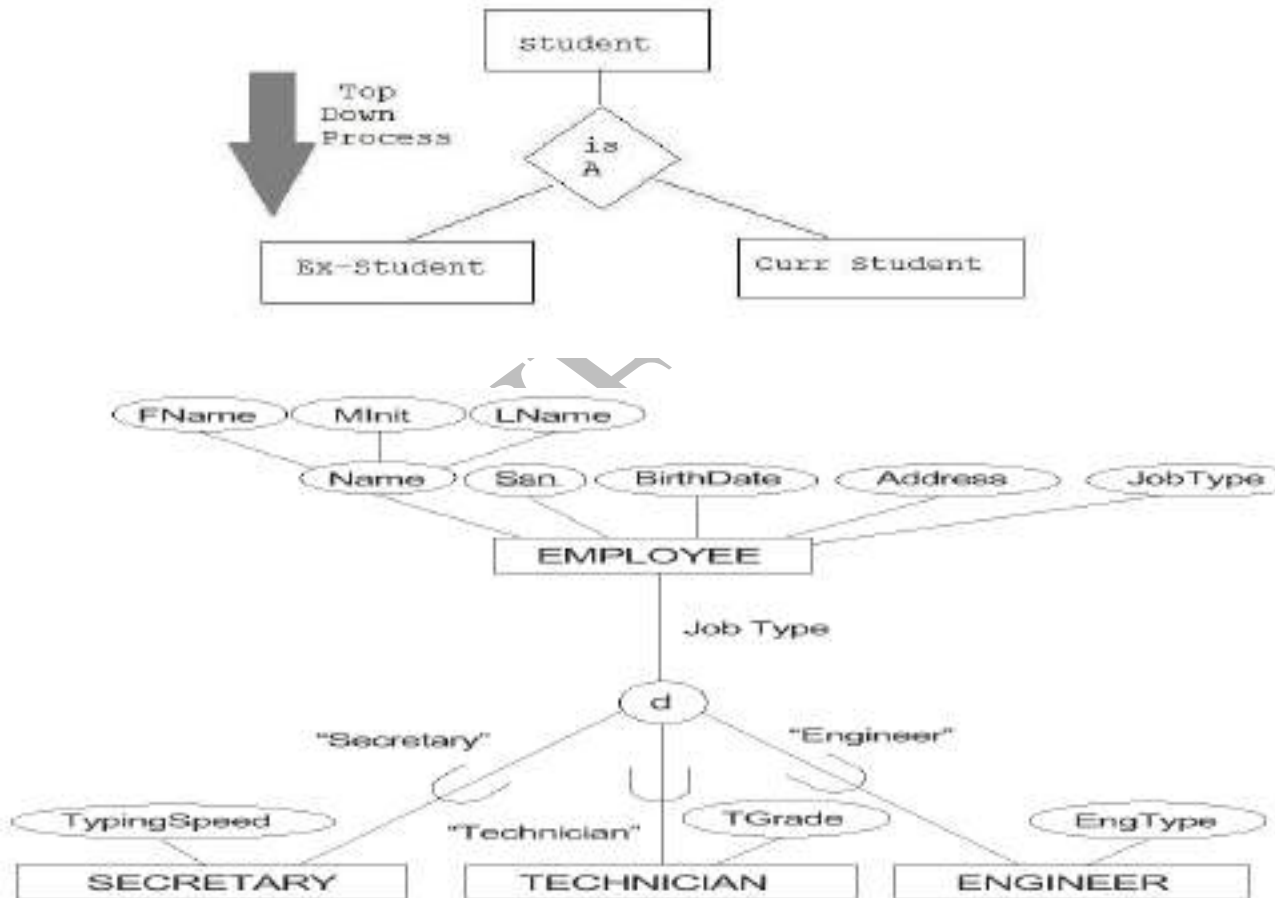
Example of Generalization. Two entity types CAR and TRUCK.

(a) Generalizing CAR and TRUCK into VEHICLE.



Specialization

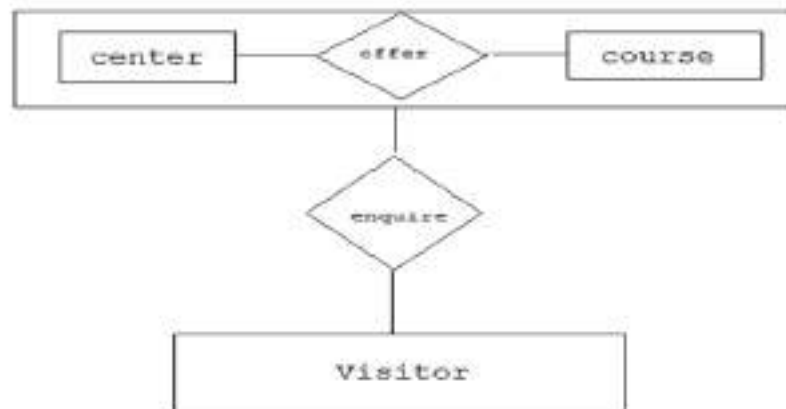
Specialization is opposite to Generalization. It is a top-down approach in which one higher level entity can be broken down into two lower level entity. In specialization, some higher level entities may not have lower-level entity sets at all.



Ex: An attribute-defined specialization on the job Type attribute of EMPLOYEE.

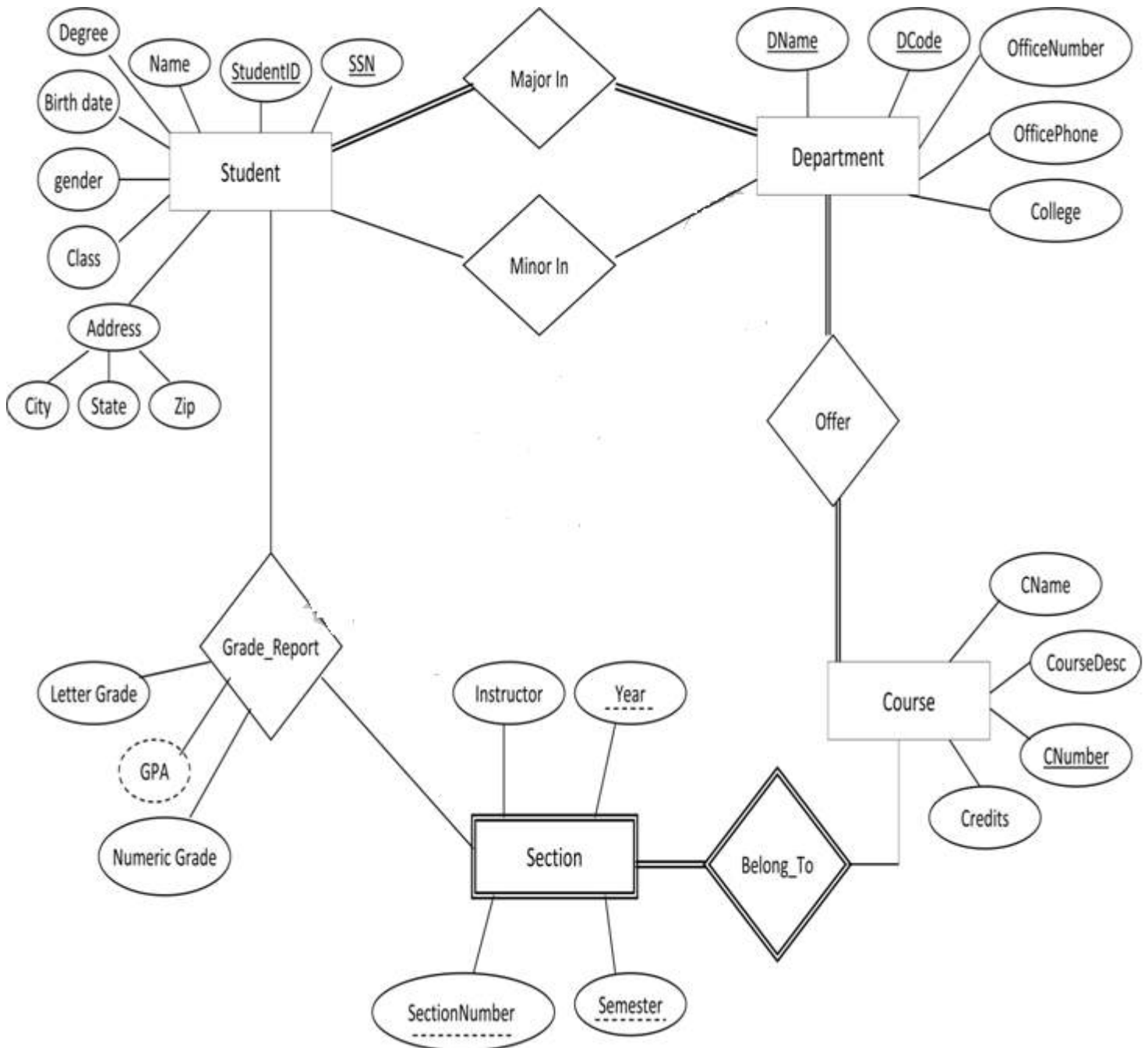
Aggregation

Aggregation is a process when relation between two entity is treated as a single entity. Here the relation between Center and Course, is acting as an Entity in relation with Visitor.

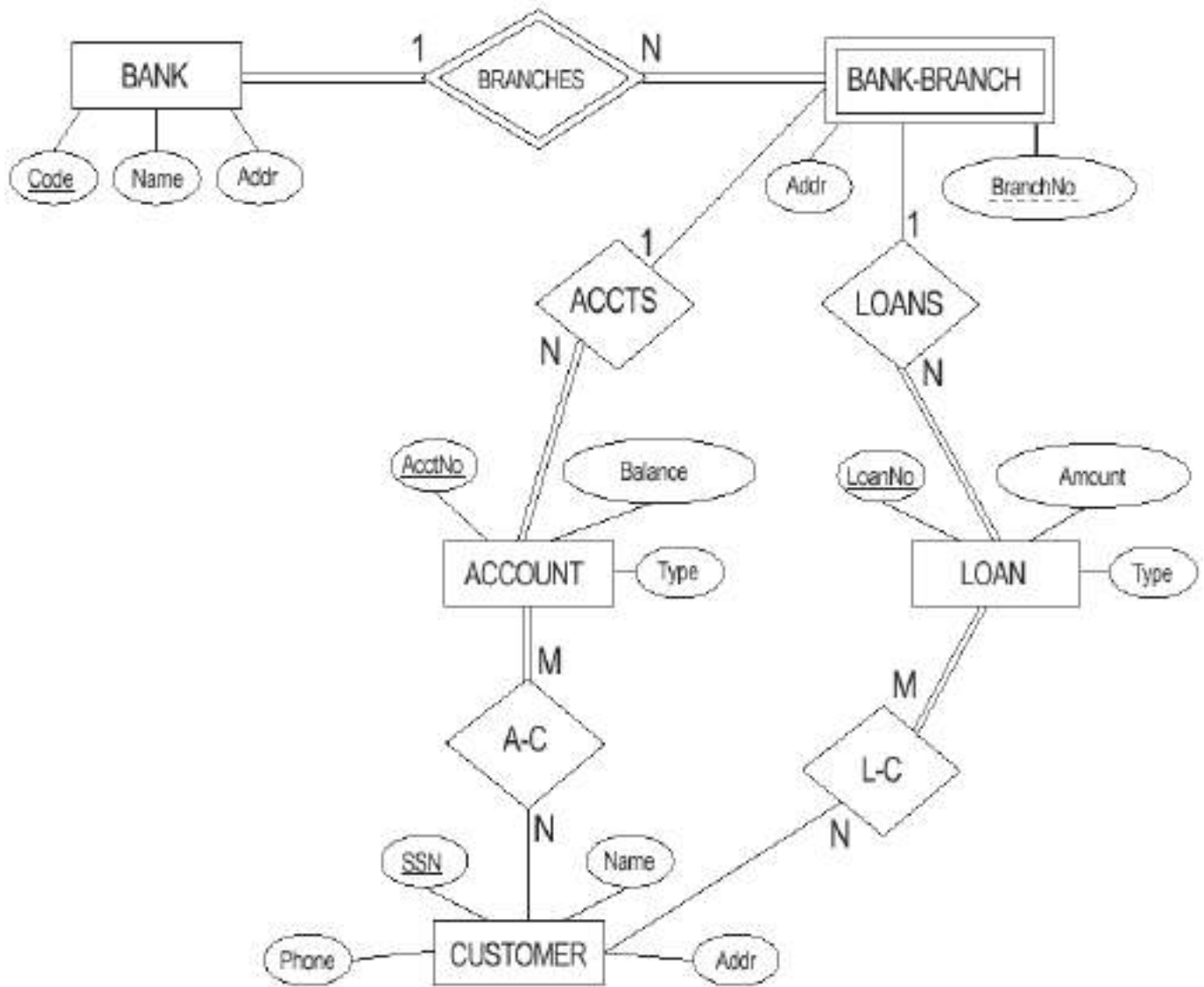


In the above figure “offer” is a relationship between center and course entities and acts as an entity for “enquires” relationship. Hence offer is called as associative entity because it acts as both relationship and entity.

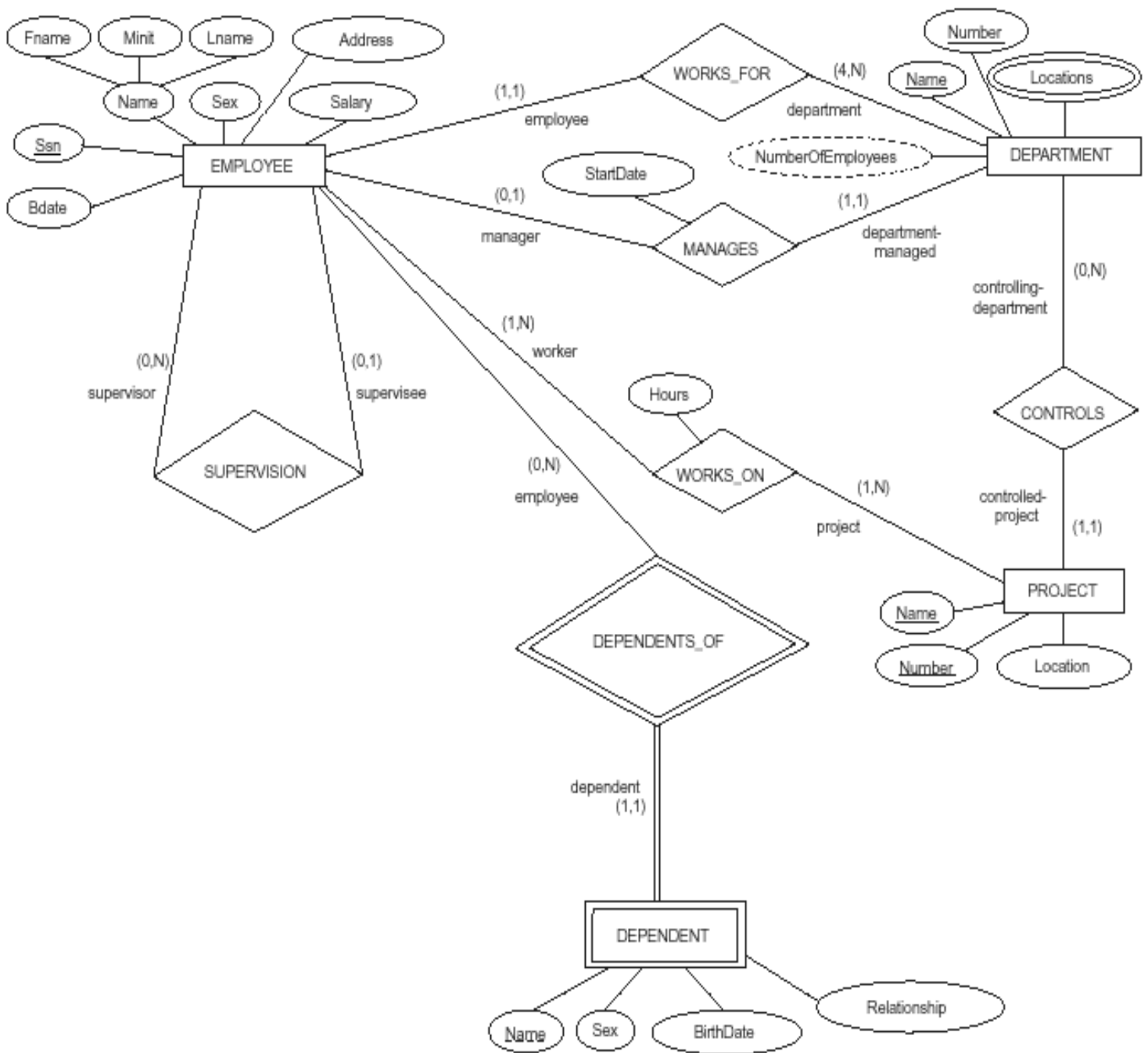
E-R diagram for University database



E-R diagram for bank database



E R Diagram for company database



Functional Dependencies and Normalization

Chapter -5

Functional Dependency

Definition:

“All non key attributes are fully functionally dependent on key attribute”.

A *functional dependency* (FD) on a relation schema **R** is a constraint is represented by

$X \rightarrow Y$, where X and Y are subsets of attributes of **R**.

- X is a determinant and Y is dependent.
- X determines Y
- Y is functionally dependent on X
- $X \rightarrow Y$

A key constraint is a special kind of functional dependency: all attributes of relation occur on the right-hand side of the FD:

SSN	Name	Town	Zip
1234	Joe	Huntingdon	16652
2345	Mary	Huntingdon	16652
3456	Tom	Huntingdon	16652
5948	Harry	Alexandria	16603

- $SSN \rightarrow SSN, Name, Address$

Redundancy

Dependencies between attributes within a relation cause redundancy

Ex. All addresses in the same town have the same zip code

SSN	Name	Town	Zip
1234	Joe	Huntingdon	16652
2345	Mary	Huntingdon	16652
3456	Tom	Huntingdon	16652
5948	Harry	Alexandria	16603

There's clearly redundant information stored here

Consistency and integrity are harder to maintain even in this simple example. e.g., ensuring the fact that the zip code always refers the same city and the city is spelled consistently

Data Anomalies

Anomalies are the problems occur on the relations due to the redundancy.

There are three types of anomalies that occur when the database is not normalized.

These are –

- Insertion,
- Update and
- Deletion anomaly.

Let's take an example to understand this.

Example

Suppose a manufacturing company stores the employee details in a table named employee that has four attributes: emp_id for storing employee's id, emp_name for storing employee's name, emp_address for storing employee's address and emp_dept for storing the department details in which the employee works. At some point of time the table looks like this:

emp_id	emp_name	emp_address	emp_dept
101	Rick	Delhi	D001
101	Rick	Delhi	D002
123	Maggie	Agra	D890
166	Glenn	Chennai	D900
166	Glenn	Chennai	D004

The above table is not normalized. We will see the problems that we face when a table is not normalized.

Update anomaly:

The modification anomaly occurs when the record is updated in the relation. This happens with modification of a attribute requires modification in all records in which value occurs or with partial update etc.

In the above table we have two rows for employee **Rick** as he belongs to two departments of the company. If we want to update the address of Rick then we have to update the same in two rows or the data will become inconsistent.

Insert anomaly:

The insertion anomaly occurs when a new record is inserted. We must insert the correct details so that they are consistent.

Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if emp_dept field doesn't allow nulls.

Delete anomaly:

The deletion anomaly occurs when a record is deleted from the relation. This happens with unexpected error while data being deleted, unseen drop off the data etc.

Suppose, if at a point of time the company closes the department D890 then deleting the rows that are having emp_dept as D890 would also delete the information of employee **Maggie** since she is assigned only to this department.

To overcome these anomalies we need to normalize the data.

Types of Constraints in DBMS or Relational Constraints in DBMS with Examples-

What are Database Constraints?

Database constraints are restrictions on the contents of the database or on database operations. It is a condition specified on a database schema that restricts the data to be inserted in an instance of the database.

Need of Constraints:

Constraints in the database provide a way to guarantee that

- The values of individual columns are valid.
- In a table, rows have valid primary key or unique key values.
- In a dependent table, rows have valid foreign key values that reference rows in a parent table.

Different Types of constraints in DBMS with Example:

- Domain Constraints
- Tuple Uniqueness Constraints
- Key Constraints
- Single Value Constraints
- Integrity Rule 1 (Entity Integrity Rule or Constraint)
- Integrity Rule 2 (Referential Integrity Rule or Constraint)

Domain Constraints –

Domain Constraints specifies that what set of values an attribute can take. Value of each attribute X must be an atomic value from the domain of X.

The data type associated with domains includes integer, character, string, date, time, currency etc.

An attribute value must be available in the corresponding domain.

Consider the example below –

SID	Name	Class (semester)	Age
8001	Ankit	1 st	19
8002	Srishti	1 st	18
8003	Somvir	4 th	22
8004	Sourabh	6 th	A

Not Allowed. Because Age is an Integer Attribute.

Tuple Uniqueness Constraints –

A relation is defined as a set of tuples. All tuples or all rows in a relation must be unique or distinct. Suppose if in a relation, tuple uniqueness constraint is applied, then all the rows of that table must be unique i.e. it does not contain the duplicate values. For example,

SID	Name	Class (semester)	Age
8001	Ankit	1 st	19
8002	Srishti	2 nd	18
8003	Somvir	4 th	22
8004	Sourabh	6 th	19

Not Allowed. Because all rows must be unique.

Key Constraints –

Keys are attributes or sets of attributes that uniquely identify an entity within its entity set. An Entity set E can have multiple keys out of which one key will be designated as the primary key. Primary Key must have unique and not null values in the relational table. In an subclass hierarchy, only the root entity set has a key or primary key and that primary key must serve as the key for all entities in the hierarchy.

Example of Key Constraints in a simple relational table

SID	Name	Class (semester)	Age
8001	Ankit	1 st	19
8002	Srishti	1 st	18
8003	Somvir	4 th	22
8004	Sourabh	6 th	45
8002	Tony	5 th	23

Not allowed as Primary Key Values must be unique

Integrity Rule 1 (Entity Integrity Rule or Constraint) –

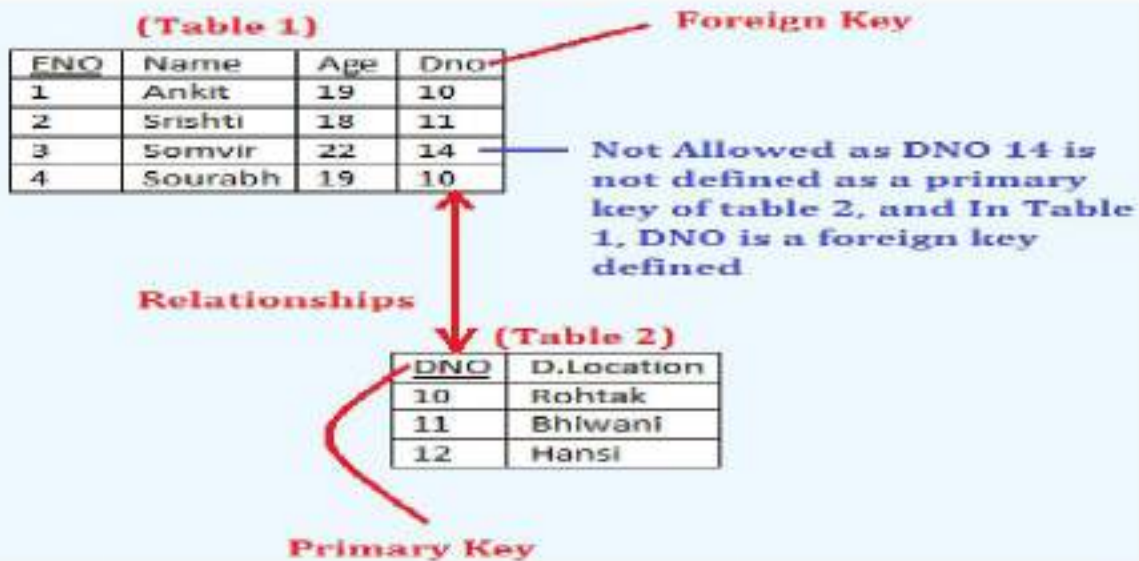
The Integrity Rule 1 is also called Entity Integrity Rule or Constraint. This rule states that no attribute of primary key will contain a null value. If a relation have a null value in the primary key attribute, then uniqueness property of the primary key cannot be maintained. Consider the example below

<u>SID</u>	Name	Class (semester)	Age
8001	Ankit	1 st	19
8002	Srishti	2 nd	18
8003	Somvir	4 th	22
	Sourabh	6 th	19

Not allowed as primary key cannot contain a NULL value

Integrity Rule 2 (Referential Integrity Rule or Constraint) –

The integrity Rule 2 is also called the Referential Integrity Constraints. This rule states that if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2. For example,



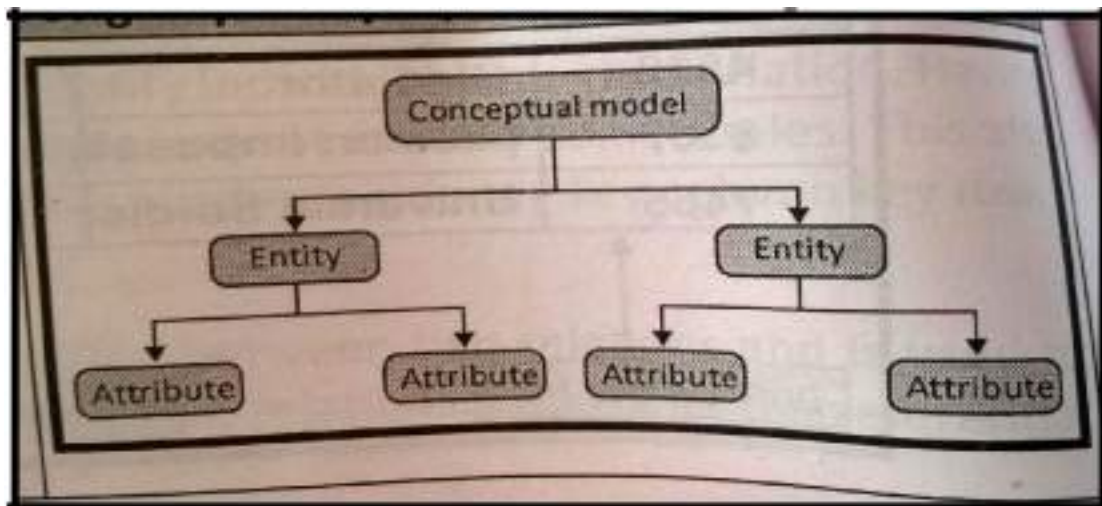
Database Design

It is the creation of entity relationship diagram to develop applications to that perform well.

Design Strategies: Depending up on factors like scope, size of the system the database is designed.

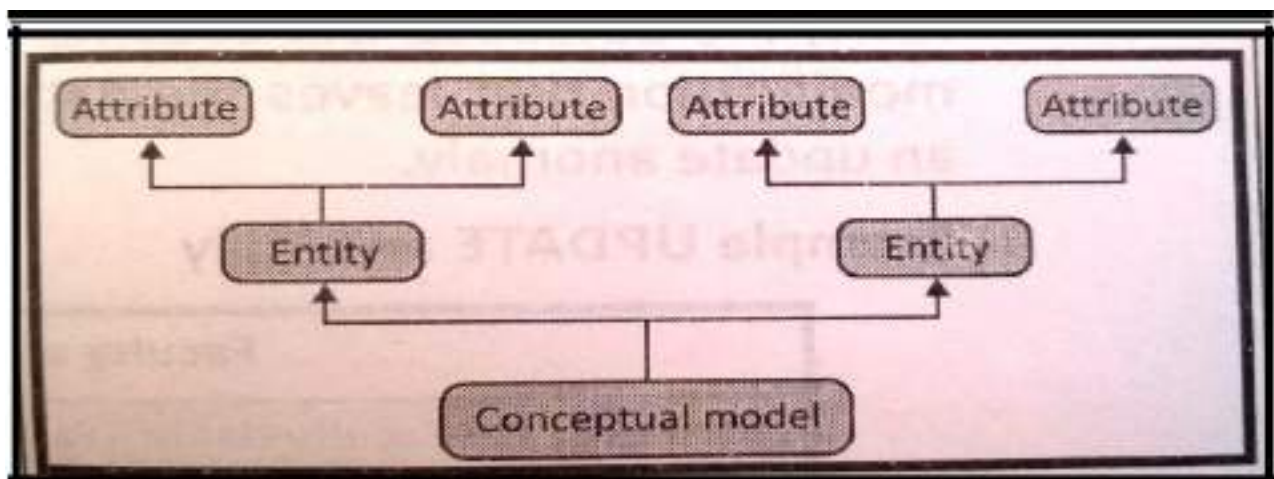
Top down design:

The top down design method starts from the root and moves to the sub level entities.



Bottom Up:

Bottom up approach begins with sub level entities and moves to root



Informal design guidelines for Relational schemas

Goodness of Relational DB Design

1. Semantics of attributes

It specifies how to interpret the attribute values stored in tuple.

How to relate one tuple to another

Ex: Each tuple in a relation should represent one entity instance

Only foreign keys should be used to refer to other entities

Redundant information in tuples and anomalies

The main goal of this is to minimize the storage space used by the base relations by avoiding anomalies

3. Removing Null Values in Tuples

If attributes does not apply to the tuples in the relation we end up with many nulls in those tuples.

This may lead to wastage of memory.

It usually happens when

- The attributes does not apply to the tuple
- The attribute value for the tuple is unknown
- The value is known but absent

4. Generation of spurious tuples:

This happens when decomposing relations where unwanted extra tuples are formed which leads to bad schema.

Decomposition:

States that we can divide the large tables into smaller tables.

There are three desirable properties:

1. Lossless.
2. Dependency preservation.
3. Minimal redundancy.

Goals of Decomposition:

- Eliminate redundancy by decomposing a relation into several relations in a higher normal form.
- It is important to check that decomposition does not lead to bad design.

Example of decomposition

There are 2 subsets

$R_1(x,z)$ and $R_2(Y,Z)$

If we union R_1 and $R_2(R_1 \cup R_2)$, we get

$R=(x,y)U(y,z)$

$R=(x,y,z)$

$R(x,y,z)$

LOSSY DECOMPOSITION:

Additional tuples are obtained along with original tuples, although there are more tuples, this leads to less information. Due to the loss of information, this decomposition is called lossy decomposition.

LOSSLESS DECOMPOSITION:

A decomposition $\{R_1, R_2, \dots, R_n\}$ of a relation R is called a lossless decomposition for R if the natural join of R_1, R_2, \dots, R_n produces exactly the relation R .

What is Normalization

Definition:

It is a step by step procedure to reduce redundancy in a relation is called as normalization.

Normalization is a database design technique which organizes tables in a manner that reduces redundancy and dependency of data.

It divides larger tables to smaller tables and link them using relationships.

Normalization Benefits:

- >Facilitates data integration.
- >Reduces data redundancy.
- >Provides a robust architecture for retrieving and maintaining data.
- >Compliments data modeling.
- >Reduces the chances of data anomalies occurring.

Goals of Normalization (purposes):

There are two goals of the normalization process:

- >Eliminating redundant data (for example, storing the same data in more than one table) and
- >Ensuring data dependencies make sense (only storing related data in a table).
- >Reduce the potential for anomalies.

Advantages of Normalization:

- >Avoid redundancy (same data stored many times in same/different tables).
- >All the update anomalies and does not have any loss of data or inefficient data update process.
- >In this, well organized databases where all the tables are inter-related maintain integrity and consistency of data.
- >All data are stored efficiently since there is no redundancy.

Disadvantages of Normalization:

- >Maintaining more tables is a bit tough.
- >Nested queries over multiple tables' gats tricky.

First normal form (1NF)

A relation is said to be in first normal form it must be satisfy some rules.

- Each and every column must be having atomic values.
- No repeating groups are entertained.
- Each column must be having unique name.
- No two rows are identical.

As per the rule of first normal form, an attribute (column) of a table cannot hold multiple values. It should hold only atomic values.

Example: Suppose a company wants to store the names and contact details of its employees.

It creates a table that looks like this:

emp_id	emp_name	emp_address	emp_mobile
101	Herschel	New Delhi	8912312390
102	Jon	Kanpur	8812121212 9900012222
103	Ron	Chennai	7778881212
104	Lester	Bangalore	9990000123 8123450987

Two employees (Jon & Lester) are having two mobile numbers so the company stored them in the same field as you can see in the table above.

This table is **not in 1NF** as the rule says “each attribute of a table must have atomic (single) values”, the emp_mobile values for employees Jon & Lester violates that rule.

emp_id	emp_name	emp_address	emp_mobile
101	Herschel	New Delhi	8912312390
102	Jon	Kanpur	8812121212
102	Jon	Kanpur	9900012222
103	Ron	Chennai	7778881212
104	Lester	Bangalore	9990000123
104	Lester	Bangalore	8123450987

Now the above relation in the 1st normal form.

Second normal form (2NF)

A table is said to be in 2NF if both the following conditions hold:

- Table is in 1NF (First normal form)
- All non-prime attributes are dependent on the key attribute. (it achieves full functional dependency)

Example: Suppose a school wants to store the data of teachers and the subjects they teach. They create a table that looks like this: Since a teacher can teach more than one subjects, the table can have multiple rows for a same teacher.

teacher_id	subject	teacher age
111	Maths	38
111	Physics	38
222	Biology	38
333	Physics	40
333	Chemistry	40

The table is in 1 NF because each attribute has atomic values. However, it is not in 2NF because non prime attribute teacher age is dependent on teacher_id alone which is a proper subset of candidate key. This violates the rule for 2NF as the rule says “**no** non-prime attribute is dependent on the proper subset of any candidate key of the table”.

To make the table in 2NF we can break it in two tables like this:

Teacher details table:

teacher_id	teacher age
111	38
222	38
333	40

teacher_subject table:

teacher_id	subject
111	Maths
111	Physics
222	Biology
333	Physics
333	Chemistry

Now the tables satisfy with Second normal form (2NF).

Third Normal Form (3NF)

A relation is said to be in third normal form it must be follows some rules.

- It should be in 2NF.
- It eliminates Transitive dependency.

Definition for transitive dependency:

Some times non key attributes are not only depends on key attributes also depends on non key attributes is referred as transitive dependency.

For example, consider a table with following fields.

Student_Detail Table :

<u>Studentid</u>	Student_name	DOB	Street	city	State	Zip
------------------	--------------	-----	--------	------	-------	-----

In this table Student_id is Primary key, but street, city and state depends upon Zip.

The dependency between zip and other fields is called transitive dependency.

Hence to apply 3NF, we need to move the street, city and state to new table, with Zip as primary key.

New Student_Detail Table :

<u>Student_id</u>	Student_name	DOB	Zip
-------------------	--------------	-----	-----

Address Table :

<u>Zip</u>	Street	city	state
------------	--------	------	-------

The advantage of removing transitive dependency is,

- Amount of data duplication is reduced.
- Data integrity achieved.

BCNF: (Boyce Codd Normal form)

Def:-A relation is said to be in BCNF it must be satisfy some rules

- It satisfies 3NF.
- All determinants must be candidate key or super key

Ex: A table not in BCNF.

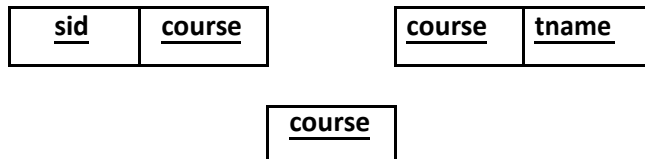
<u>sname</u>	<u>course</u>	tname
a	bca	aaa
b	bsc	bbb
c	be	ccc
d	bcom	ddd

Here key: {sname, course} Tname -----> course

Here problem is tname attribute is not a key attribute but determines course attribute.

Solution:

Decomposing this tname and course into other tables.



Differences between 3rd NF and BCNF

BASIS FOR COMPARISON	3NF	BCNF
Concept	No non-prime attribute must be Transitively dependent on the Candidate key.	For any trivial dependency in a relation R say $X \rightarrow Y$, X should be a super key of
Dependency	3NF can be obtained without sacrificing all dependencies.	Dependencies may not be preserved in BCNF.
Decomposition	Lossless decomposition can be achieved in 3NF.	Lossless decomposition is hard to achieve in BCNF.

4th Normal Form

A relation is said to be in 4th normal form it satisfy some rules.

- It satisfies BCNF.
- A table contains no Multivalve dependencies.

Ex:

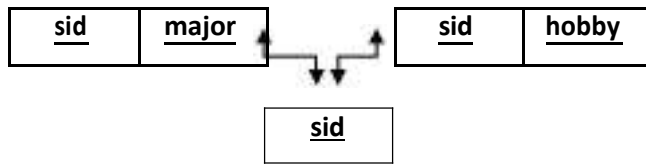
A table not in 4NF

<u>sid</u>	<u>major</u>	<u>hobby</u>
1	IT	reading
2	CS	singing
3	CS	learning

Here

Key: {sid, major, hobby} Multivalve dependency: sid \twoheadrightarrow major, hobby

This relation will be bringing into 4th normal form we can decompose into this way.



5th Normal form (Projection based join normal form)

A relation is said to be in 5NF, if and only if,

- It's in 4NF
- It has no join dependency.
- If we can decompose table further to eliminate redundancy and anomaly, and when we re-join the decomposed tables by means of candidate keys, we should not be losing the original data or any new record set should not arise. In simple words, joining two or more decomposed table should not lose records nor create new records.

Types of SQL Keys

A key is a single or combination of multiple fields in a table.

It is used to fetch or retrieve records/data-rows from data table according to the condition/requirement.

Keys are also used to create relationship among different database tables or views

We have following types of keys in SQL which are used to fetch records from tables and to make relationship among tables or views.

1. Super Key

Super key is a set of one or more than one keys that can be used to identify a record uniquely in a table.

Example : Primary key, Unique key, Alternate key are subset of Super Keys.

2. Candidate Key

A Candidate Key is a set of one or more fields/columns that can identify a record uniquely in a table. There can be multiple Candidate Keys in one table. Each Candidate Key can work as Primary Key.

Example: In below diagram ID, RollNo and EnrollNo are Candidate Keys since all these three fields can be work as Primary Key.

3. Primary Key

Primary key is a set of one or more fields/columns of a table that uniquely identify a record in database table. It cannot accept null, duplicate values. Only one Candidate Key can be Primary Key.

4. Alternate key

A Alternate key is a key that can be work as a primary key. Basically it is a candidate key that currently is not primary key.

Example: In below diagram RollNo and EnrollNo becomes Alternate Keys when we define ID as Primary Key.

5. Composite/Compound Key

Composite Key is a combination of more than one fields/columns of a table. It can be a Candidate key, Primary key.

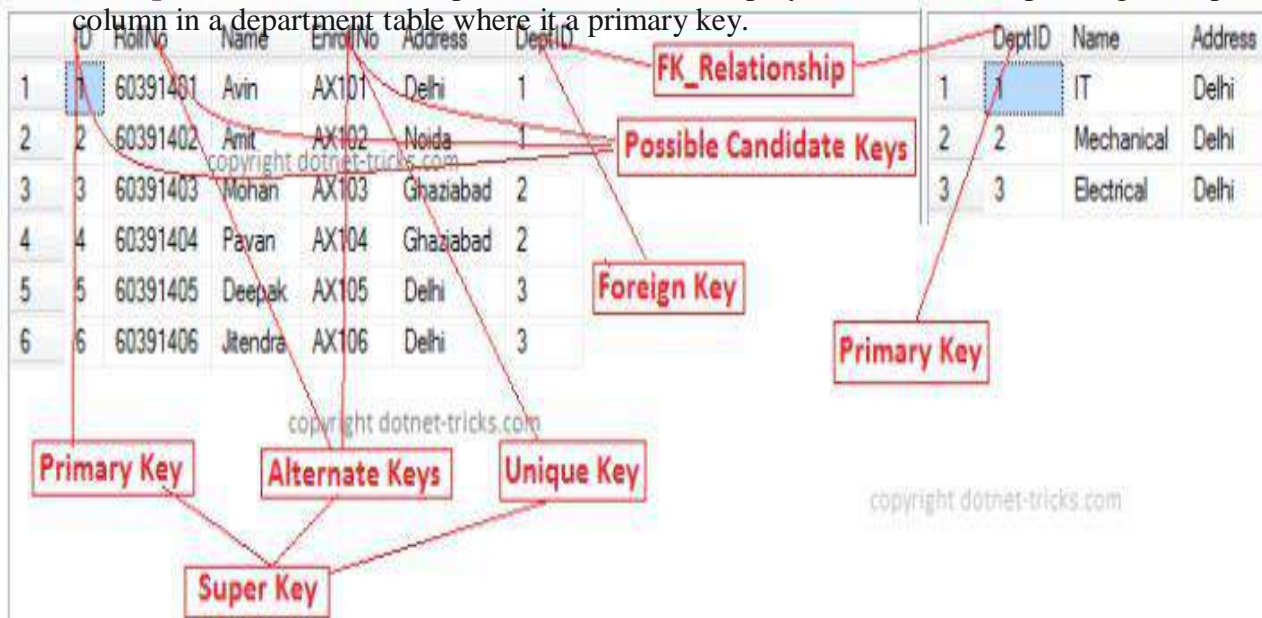
6. Unique Key

Uniquekey is a set of one or more fields/columns of a table that uniquely identify a record in database table. It is like Primary key but it can accept only one null value and it cannot have duplicate values.

7. Foreign Key

Foreign Key is a field in database table that is Primary key in another table. It can accept multiple null, duplicate values.

Example : We can have a DeptID column in the Employee table which is pointing to DeptID column in a department table where it a primary key.



CHAPTER 6

Introduction: SQL

SQL stands **Structural Query Language** the name is derived from the word of **SEQUEL** (sequential English query language). This was designed and implemented at IBM research center. It is a standard relational database language which has several parts.

RDBMS:

It is a DBMS in which data is stored in the form of tables and relationship among the data was also stored in the form of tables.

Query:

Retrieving or extracting required information from the database by using DML commands is known as query.

Differences between DBMS and RDBMS

DBMS	RDBMS
In DBMS no relationship concept.	It is used to establish the relationship.
It supports single user only.	It supports multiple users.
It treats data as files internally.	It treats data as tables internally.
It supports 3 rules E.F.CODD.	It supports minimum 6 rules out of 12.
It requires low software and hardware Requirements	It requires high software and hardware requirements.
Normalization process will not be used.	Normalization process will be present.
It does not support any constraints.	It supports all types constraints.

Basic terminologies used in RDBMS

Tuple or Row:

A tuple is a collection of attribute values. (or) A single row in the table is called as tuple. It is also known as record.

Cardinality of relation:

The total number of rows in the table is called as cardinality of that relation.

Degree of relation:

The total number of columns in the table is called as degree of that relation.

Attribute or Column:

A column is a single piece of information on vertical side of table. It is also known as field.

Cell:

Intersection of rows and columns is referred as cell.

Relation:

Relation is nothing but a table. It is in 2 dimensional formats.

Combination of unnamed rows and named columns is known as table or relation.

SQL Data Types

When we creating tables, we must specify a data type for each column.

The data types in SQL can be classified into 5 major categories. They are

- ❖ Character (char, varchar2, text)
- ❖ Numeric (number)
- ❖ Binary (raw, long raw, BLOB etc.)
- ❖ Date and time (date, timestamp etc.)
- ❖ Row id (rowid etc.)

Most frequently used data types in SQL are

- Text (or) char (or) varchar2
- number
- Date/Time types.

CHAR (size):

Holds a fixed length string (can contain letters and special characters).

The fixed size is specified in parenthesis. Can store up to 255 characters. If we give more letters then it will display error message or it will take only fixed size letters.

VARCHAR2 (size):

Holds a variable length string (can contain letters, numbers, and special characters).

The maximum size is specified in parenthesis. Can store up to 255 characters.

Note: If you put a greater value than 255 it will be converted to a TEXT type.

TEXT:

Holds a string with a maximum length of 65,535 characters. It does not have size.

BLOB:

For BLOBs (Binary Large Objects). The maximum size of this data type is 4 giga bytes of data.

CLOB

For CLOBs (character Large Objects). The maximum size of this data type is 4 giga bytes of data.

Numeric (p, s):

This data type is used to store numeric data.

The p parameter indicates the maximum total number of digits that can be stored left side of decimal point. The s parameter indicates the maximum number of digits stored to the right of the decimal point.

Integer:

The INTEGER data type accepts a 64-bit signed integer value with an implied scale of zero. It does not have size.

Date:

The DATE data type accepts date values, consisting of year, month, and day. No parameters are required when declaring a DATE data type. Date values should be specified in the form: DD-
MMM-YYYY.

Types of SQL Statements

All most all relational database management systems use SQL for data manipulation and retrieval.

SQL is a non-procedural language, here we need to concentrate on what you want, not how to get.

You need not to concentrates on procedural concepts.

SQL Statements are divided into 4 categories. They are

- ❖ DDL (Data Definition Language)
- ❖ DML (Data Manipulation Language)
- ❖ DCL (Data Control Language)
- ❖ TCL (Transaction Control Language)

Data Definition Language (DDL)

This part of SQL provides commands for defining relational schemas like tables, views, index etc. It also provides commands for deleting and modifying the relational scheme.

DDL Commands are

- ❖ **Create**
- ❖ **Alter**
- ❖ **Drop**
- ❖ **Truncate**
- **Create Command:**

It is used to create the definition of a new relation.

Syntax: Create Table Table Name

```
(  
    Column name1 DataType1 (Size),  
    Column name2 DataType2 (Size),  
    :  
    Column name n Data Type n (Size)  
);
```

- **Drop Command:**

This command is used to delete unwanted tables from the database.

Syntax: Drop Table Table Name;

- **Alter Command:**

The alter command can be used on a table to modify the table.

**Syntax: Alter Table Table Name Add/modify/drop/rename(NewColumnName
DataType(Size));**

It has several options.

Add option:

Using this we can add column or primary key to the existing table.

Syntax: alter table table name add (new colname datatype(size));

Syntax: alter table table name add primary key (colname);

Drop option:

Using this we can drop a particular column from the table.

Syntax: alter table table name drop column colname;

Modify option:

Using this option we can modify data types of particular column and also we can do increase or decrease sizes of data type.

Syntax: alter table table name modify(colname data type(size));

Rename option:

Using this we can rename the table name as well as column name in a table.

If we want to rename the table then we need to follow the syntax.

Syntax:

Alter table existing table name rename to new table name;

If we want to rename the column then we need to follow the syntax.

Syntax:

Alter table table name rename column existing colname to new colname;

- **Truncate Command:**

This command is used to deletes the data in the table. But not the structure.

Syntax: truncate table table name;

Data Manipulation Language (DML)

This part of SQL provides commands for manipulating adding and deleting data in various tables of the databases.

Some of the DML commands are

- ❖ **Insert**
- ❖ **Delete**
- ❖ **Update**
- ❖ **Select**

- **Insert Command:**

It is used to add one or more tuples into an existing table.

Using this we can insert one tuple at a time.

Syntax1:

Insert into table Name (colName1, ColName 2,) values (value1, value2);
(or)
Insert into table name values (value1, 'value2', Value n);

Using this we can insert multiple tuples at a time.

Syntax 2:

Insert into table name values (&colname1,'&colname2', &colname n);

Those colnames are having varchar2 or char or text or date data types must be enclosed within the single quotes.

- **Delete Command:**

It is used to remove the tuples or records which are existing in a relation relation, deletes entire table temporarily and also delete one particular record from table.

Syntax1:

delete table name ;

Using this we can delete entire tuples but not the structure.

Syntax 2:

delete table name where condition;

Using this delete particular tuple if it is matched that condition.

- **Select Command:**

This command is used extract information from a particular table of a database. This command performs a query on the table and extracts data from it. Query is a request for data.

Syntax:

Select ColName1, ColName2, from Table name 1, TableName2...
Where <Condition>;

- **Update Command:**

This command is used update or modifies the contents of a table or relation.

Syntax:

Update table name set colname=value1, colname=value2... from table name
Where condition;

Transaction Control Language (TCL)

This part of SQL provides commands for controlling the transactions. Some TCL commands are

- **Commit:**

This command is used to save the changes made by the transactions on

The databases from the last commit point.

Syntax: Commit;

- **Rollback:**

This command is used to undo the changes made by the transactions on the database before it is committed.

Syntax: Rollback;

Data Control Language (DCL)

This part of SQL provides commands for the DBA to either give or take away rights from various users in controlling the data. Some DCL Commands are

- **Grant :**

Used to give privileges (authority) to specified users. This command is used only by the DBA.

- **Revoke :**

This command is used to take away some or all privileges from the specified user

SQL Constraints

SQL constraints are used to specify additional rules applied on the data in a table.

Constraints can be specified when the table is created (inside the CREATE TABLE statement) or after the table is created (inside the ALTER TABLE statement).

General syntax to give constraint to the table:

```
CREATE TABLE table_name
(
column_name1 data_type(size) constraint_name,
column_name2 data_type(size) constraint_name,
column_name3 data_type(size) constraint_name,
....
);
```


SQL NOT NULL Constraint

- By default each and every column can hold NULL value in a table.
- If we do not want to allow NULL value to a column then we need to add this constraint to the column.
- A table can have number of not null constraints.
- The NOT NULL constraint enforces a column to NOT accept NULL values.
- The following SQL enforces the "Pid" column and the "LastName" column to not accept NULL values:
- It will accepts duplicate but not accept NULL values.

Syntax:

Create table tablename

```
(  
  Colname1 datatype(size) not null,  
  Colname2 datatype(size),  
  :  
  Colname n datatype n  
);
```

Example:

```
CREATE TABLE Persons  
(  
  PId number(10) NOT NULL,  
  LastName varchar2(25),  
  FirstName varchar2(25),  
  Address varchar2(25),  
  City varchar2(25)  
);
```

After creating a table then we need to check whether not null constraint will applied or not on this table then we need to insert values into this table.

SQL UNIQUE Constraint

- The UNIQUE constraint uniquely identifies each record in a database table.
- The UNIQUE and PRIMARY KEY constraints both provide a guarantee for uniqueness for a column or set of columns.

- Note that you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.
- This will be applicable on both numeric and non numeric data.

Syntax:

```

Create table tablename
(
  Colname1 datatype(size) unique,
  Colname2 datatype(size) unique,
  :
  Colname n datatype n
);
CREATE TABLE Persons
(
  PId number(10) unique,
  LastName varchar2(25) unique,
  FirstName varchar2(25),
  Address varchar2(25),
  City varchar2(25) unique
);

```

SQL UNIQUE Constraint on ALTER TABLE

ALTER TABLE table name ADD UNIQUE (colname);

After creating a table then we need to check whether unique constraint will applied or not on this table then we need to insert values into this table.

SQL PRIMARY KEY Constraint

- The PRIMARY KEY constraint uniquely identifies each record in a database table.
- Primary keys must contain unique values.
- A primary key column cannot contain NULL values.
- Each table should have a primary key, and each table can have only ONE primary key.

SQL PRIMARY KEY Constraint on CREATE TABLE

Syntax:

```

Create table tablename
(
  Colname1 datatype(size) primary key,

```

```
Colname2 datatype(size),
    :
Colname n datatype n
);
```

The following SQL creates a PRIMARY KEY on the "PID" column when the "Persons" table is created:

```
CREATE TABLE Persons
(
  PID number(10) primary key,
  LastName varchar2(25),
  FirstName varchar2(25),
  Address varchar2(25),
  City varchar2(25)
);
```

SQL PRIMARY KEY Constraint on ALTER TABLE

```
ALTER TABLE tablename ADD PRIMARY KEY (colname);
```

After creating a table then we need to check whether primary key constraint will be applied or not on this table then we need to insert values into this table.

SQL FOREIGN KEY Constraint

- A FOREIGN KEY in one which points to a PRIMARY KEY on another table.
- It allows values from primary key table and also maintains relationship between the parent and the child tables.

Syntax:

```
Create table tablename
(
  Colname1 datatype(size) ,
  Colname2 datatype(size),
  :
  Colname n datatype (size) references parent tablename;
);
```

Example:

```
CREATE TABLE Persons
( PID number(10) references persons,
  Result varchar2(25) );
```

SQL FOREIGN KEY Constraint on ALTER TABLE

```
ALTER TABLE TABLENAME ADD FOREIGN KEY (COLNAME) REFERENCES  
PARENT TABLENAME (COLNAME);
```

After creating a table then we need to check whether foreign key constraint will be applied or not on this table then we need to insert values into this table.

SQL CHECK Constraint

- The CHECK constraint is ensures that all values in a column satisfy certain condition.
- Based on that condition only we can insert values into the table.
- If you define a CHECK constraint on a single column it allows only certain values for this column.
- It will be applicable only on numeric data.

Syntax:

```
Create table tablename  
(  
  Colname1 datatype(size) ,  
  Colname2 datatype(size),  
  :  
  Colname n datatype (size) check (condition);  
);
```

Example:

```
CREATE TABLE Persons  
(  
  PId number(10) check (pid>2),  
  LastName varchar2(25),  
  FirstName varchar2(25),
```

PL/SQL

CHAPTER-7

What is PL/SQL?

PL/SQL is oracle's procedural language extension to SQL. PL/SQL allows to mix SQL statements with procedural statements.

PL/SQL is superset of SQL.

Differences between SQL and PL/SQL:

SQL:

- It is a data oriented language for selecting and manipulating data.
- It is a non-procedural language.
- It does not support programming part.
- SQL engine executes only one statement at a time.
- We can embed SQL statements in a PL/SQL program.
- It is used to frame the queries by using the commands.
- If we made a mistake SQL engine will display its own error messages.

PL/SQL:

- PL/SQL is Oracle's procedural language to create applications.
- PL/SQL is executed block of code.
- PL/SQL is the superset of SQL.
- We cannot embed PL/SQL statements within a SQL.
- PL/SQL allows you to mix SQL statements with procedural statements like IF statement, Looping structures etc.
- It uses SQL for data retrieval and manipulation and uses its own statements for data processing.
- It will display user friendly errors.

Advantages (or) features of PL/SQL

Block structure:

PL/SQL consists of a block of code which can be nested within each other. Each program written in PL/SQL is written as a block. It is easy to understand and access.

Procedural Language capability

PL/SQL consists of procedural language constructs such as conditional statements and looping statements.

Better performance:

PL/SQL engine processes multiple SQL statements simultaneously as a single block. it improves performance.

Exception (or) error handling:

PL/SQL allows errors, called as exceptions, to be detected and handled. Whenever there is a predefined error PL/SQL raises an exception automatically. These exceptions can be handled to recover from errors.

Modularity:

PL/SQL allows process to be divided into different modules. Subprograms called as procedures and functions can be defined and invoked using the name. These subprograms can also take parameters.

Portability:

Applications written in PL/SQL are portable to any platform on which Oracle runs. Once you write a program in PL/SQL, it can be used in any environment without any change at all.

PL/SQL BLOCKS

PL/SQL blocks can be generally categorized as follows:

- Named blocks
- Anonymous (or) Unnamed blocks

Named blocks: are the blocks these are having names like procedures, functions, packages, cursors and triggers.

Unnamed blocks: are the blocks which are not having names. These are

Declare
Begin
Exception
End

PL/SQL block Structure

PL/SQL programs are written as blocks. Block allows you to group logically related statements and declarations.

PL/SQL block is consisting of the following three parts:

- ❖ Declarative part (optional)
- ❖ Executable part (mandatory)
- ❖ Exception-handling part (optional)

The following is the syntax of PL/SQL block.

```
DECLARE
    Declaration of variable
BEGIN
    Executable commands
EXCEPTION
    Exception handlers
END;
```

Declarative Part:

This section is optional and is used to declare variables, constants, records and cursors etc.

The declaration section of a PL/SQL block starts with the keyword “declare”.

The following is the example of declarative part.

First variable is of type NUMBER (5). As we have seen before PL/SQL supports the data types of SQL.

The second variable is initialized to 0.

```
Declare
V_rollno number (5);
V_count number (2):= 0;
```

Executable part:

The execution section of a PL/SQL block starts with the keyword begin and ends with end keyword.

This is the only mandatory part of the entire block, where the program logic is written to perform any task.

```
Begin
    Statements
End;
```

Exception-handling part:

The exception section of a PL/SQL block starts with the keyword “exception”.

This section is optional. Any errors in the program can be handled in this section.

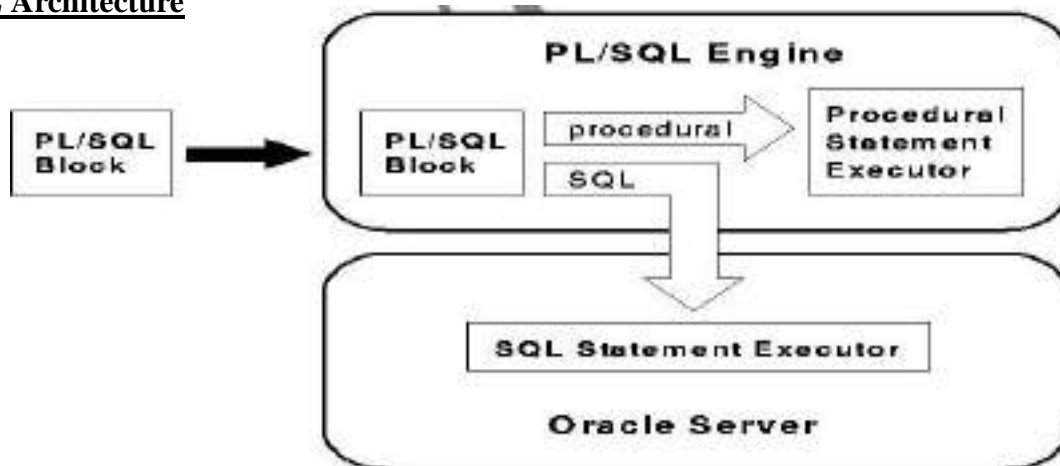
Exception handlers handle exceptions.

```
Exception
    Statements;
```

End section:

It is the last section of PL/SQL block. It indicates end of program.

End;
PL/SQL Architecture



- Every PL/SQL block is first executed by PL/SQL engine.
- This is the engine that compiles and executes PL/SQL blocks.
- PL/SQL engine executes all procedural statements of a PL/SQL of the block, but sends SQL command to SQL statements executor in the Oracle RDBMS.
- That means PL/SQL separates SQL commands from PL/SQL commands and executes PL/SQL commands using Procedural statement executor, which is a part of PL/SQL engine.
- PL/SQL engine locates both client side as well as server side also.

All SQL statements that are embedded within the PL/SQL block sent to the oracle server

Comments in PL/SQL

You can give comments in PL/SQL block in two ways.

- **First way is** by preceding the comment with two hyphens (- -).
Example: -- this is single line comment
- **Second way is** by starting the comment with /* and ending it with */.

Example: /* this comment can be of multiple lines */

PL/SQL Data types

PL/SQL provides a variety of predefined data types, which can be divided into three categories:

- Scalar represents a single value.
- Composite Is a collection of components
- LOB holds a large object locator.

The following are the data types in various categories

Scalar: NUMBER, CHAR, VARCHAR2, DATE, BOOLEAN

Composite: RECORD, TABLE and VARRAY.

LOB: BFILE, BLOB, CLOB, and NCLOB.

Scope and visibility of variables

Scope of the variable refers to the region of the program in which the variable can be used.

A variable is said to be visible when it can be referred without any qualifier.

Examine the following examples to understand scope and visibility of variables.

Example 1:

```
Declare
  num1 number(5);
Begin

  Declare
    num2 number(5);
  Begin
    ...
  End;

End;
```

The diagram illustrates the scope of variables NUM1 and NUM2. NUM1 is declared in the outer block and its scope extends to the end of the outer block. NUM2 is declared in the inner block and its scope extends to the end of the inner block.

Variable **NUM1** can be accessed from the point of declaration to the end of the outer block.

Variable **NUM2** can be accessed from the point of declaration to the end of inner block

Assignment Operator (:=)

Assignment operator allows a value to be stored in a variable.

Syntax: Variable: = expression;

The following are examples of assignment operator:

Count: = 1;

Name: = 'Swamy';

Displaying output from PL/SQL

In order to display output from a PL/SQL block, we have to use DBMS_OUTPUT package.

A package is a collection of procedures and functions.

PUT and PUT_LINE procedures both these procedures are used to display a

NUMBER, VARCHAR2 or DATE type value.

PUT allows you to put multiple pieces that make up a line. PUT_LINE puts the given data followed by end-of-line character.

In order to see the output sent using these procedures, the following must satisfy:

The program unit from where they are called must be completed

SERVEROUTPUT variable of SQL*PLUS must be set to ON by using the command

Set serveroutput on;

PL/SQL Execution

- Create a program in text editor or SQL window.
- After writing a program then type / at the SQL prompt then enter to execute.

PL/SQL Control Statements

There are 3 types of control structures supported by PL/SQL. They are

- Sequential control statements
- Conditional or selection control statements
- Looping statements

Conditional Statements in PL/SQL

IF - THEN statement:

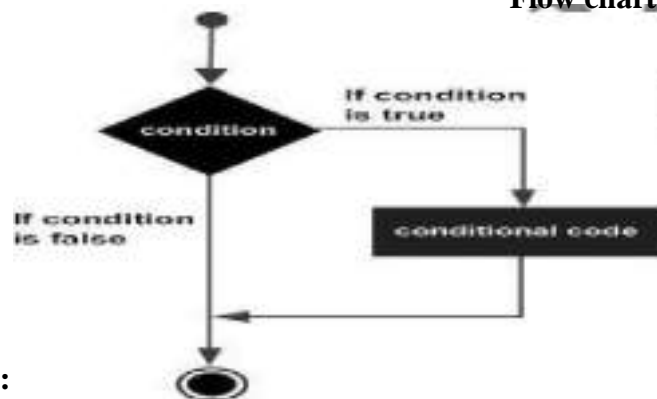
The **IF statement** associates a condition with a sequence of statements enclosed by the keywords **THEN (starting of IF)** and **END IF; (ending of IF)**.

If the condition is **TRUE**, the statements get executed, and if the condition is **FALSE** or **NULL**, then the **IF** statement display nothing.

Syntax for IF-THEN statement is:

```
IF condition
THEN
    Statements;
END IF;
```

Flow chart:



Example of an IF-THEN statement is:

```
Declare
    Age number:=19;
Begin
    IF (a <= 20)
    THEN
        Dbms_output.put_line ('voting is eligible');
    END IF;
```

End;

IF-THEN-ELSE statement

In this user executes true block as well as false block statements.

Syntax for the IF-THEN-ELSE statement is:

IF condition

THEN

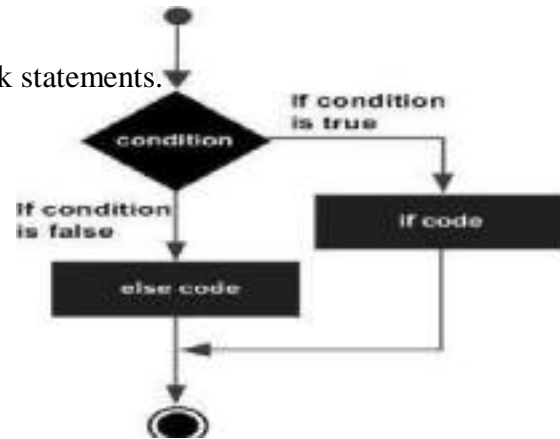
S1;

ELSE

S2;

END IF;

Flow Chart:



Where, S1 and S2 are different sequence of statements. In the IF-THEN-ELSE statements, when the test condition is TRUE, the statement S1 is executed and S2 is skipped; when the test condition is FALSE, then statement S2 is executed.

For example:

Declare

A number:=10;

B number:=20;

Begin

IF (a < b)

THEN

Dbms_output.put_line ('a is big'||a);

Else

Dbms_output.put_line ('b is big'||b);

END IF;

End;

IF-THEN-ELSE IF statement

When we mention 2 or more conditions on one program we use this syntax.

The syntax of an IF-THEN-ELSE IF Statement in PL/SQL programming language is:

IF (condition 1)

THEN

S1; -- Executes when the condition 1 is true

ELSE IF (condition 2)

THEN

S2; -- Executes when the condition 2 is true

ELSE IF (condition 3)

THEN

S3; -- Executes when the condition 3 is true

ELSE

S4; -- executes when the none of the above condition is true END IF;

Example:

```
DECLARE
a number(3) := 100;
BEGIN
IF ( a := 10 ) THEN
    dbms_output.put_line ('Value of a is 10');
ELSE IF (a = 20) THEN
    dbms_output.put_line ('Value of a is 20' );
ELSE IF ( a := 30 ) THEN
    dbms_output.put_line('Value of a is 30' );
ELSE
    dbms_output.put_line('None of the values is matching');
END IF;
    dbms_output.put_line('Exact value of a is: '|| a );
END;
```

Nested IF-THEN-ELSE

It is always legal in PL/SQL programming to nest **IF-ELSE** statements, which means you can use one **IF** or **ELSE IF** statement inside another **IF** or **ELSE IF** statement(s).

Syntax:

IF (condition 1)

THEN

IF (condition 2)

THEN

Sequence-of-statements;

END IF;

ELSE

Else-statements;

END IF;

Example: DECLARE

a number(3) := 100;

b number(3) := 200;

BEGIN

IF(a := 100)

THEN

IF(b := 200)

Case statement

Like the **IF** statement, the **CASE statement** selects one sequence of statements to execute.

This statement will select any one of several alternatives.

```
CASE selector
WHEN 'value1' THEN Statement1;
```

```
WHEN 'value2' THEN Statement2;
WHEN 'value3' THEN Statement3;
...
ELSE Statement n; -- default case
END CASE;
```

Example:

```
DECLARE
Grade char(1) := 'A';
BEGIN
CASE grade
    when 'A' then dbms_output.put_line('Excellent');
    when 'B' then dbms_output.put_line('Very good');
    when 'C' then dbms_output.put_line('Well done');
    when 'D' then dbms_output.put_line('You passed');
    when 'F' then dbms_output.put_line('Better try
again'); else dbms_output.put_line('No such grade');
END CASE;
```

Looping Control Statements

PL/SQL Basic LOOP

Basic loop structure encloses sequence of statements in between the **LOOP** and **END LOOP** statements. With each iteration, the sequence of statements is executed and then control resumes at the top of the loop.

The syntax of a basic loop in PL/SQL programming language is:

LOOP

 Sequence of statements;

END LOOP;

Here, sequence of statement(s) may be a single statement or a block of statements. An EXIT statement or an EXIT WHEN statement is required to break the loop.

Example:

```
DECLARE
    N number:=1;
Begin
Loop
    Dbms_output.put_line(n);
    N:=n+1;
```

Exit when n>5;
End loop;
End;

PL/SQL WHILE LOOP

A **WHILE LOOP** statement in PL/SQL programming language repeatedly executes a target statement as long as a given condition is true.

Syntax:

Example: DECLARE

```
WHILE condition
LOOP
    sequence_of_statements
END LOOP;

a number(2) := 1;

BEGIN

    WHILE a <=5
```

PL/SQL FOR LOOP

If number of times body of loop will be executed in such case we need to use this loop.

Syntax:

Example

```
FOR variable in condition
LOOP
    sequence_of_statements;
END LOOP;

DECLARE

a number(2);

BEGIN

    FOR a in 10 .. 20
```

Reverse FOR LOOP Statement

By default, iteration proceeds from the initial value to the final value, generally upward from the lower bound to the higher bound. You can reverse this order by using the **REVERSE** keyword.

Example:

```
DECLARE
    a number(2) ;
BEGIN
    FOR a IN REVERSE 10 .. 20
    LOOP
        dbms_output.put_line('value of a: ' || a);
    END LOOP;
END;
```

When the above code is executed at SQL prompt, it produces the following result:

```

Value of a: 20
Value of a: 19
Value of a: 18
Value of a: 17
Value of a: 16
Value of a: 15
Value of a: 14
Value of a: 13
Value of a: 12
Value of a: 11

```

ps in PL/SQL

PL/SQL allows using one loop inside another loop.

The syntax for a nested basic LOOP statement in PL/SQL is as follows:

```

LOOP
    Sequence of statements1
    LOOP
        Sequence of statements2
    END LOOP;
END LOOP;

```

The syntax for a nested FOR LOOP statement in PL/SQL is as follows:

```

FOR variable in condition1
LOOP
    sequence_of_statements
    FOR variable in condition2
    LOOP
        sequence_of_statements;
    END LOOP;
END LOOP;

```

The syntax for a nested WHILE LOOP statement in Pascal is as follows:

```

WHILE condition1
LOOP
    sequence_of_statements1
    WHILE condition2
    LOOP
        sequence_of_statements2
    END LOOP;
END LOOP;

```

Labeling a PL/SQL Loop

PL/SQL loops can be labeled. The label should be enclosed by double angle brackets (<< and >>) and appear at the beginning of the LOOP statement. The label name can also appear at the end of the LOOP statement.

The following program illustrates the concept:

```

<<outer>>
DECLAR
    n number:=10;
BEGIN

```

```

DECLARE
    n number:=20; begin
    dbms_output.put_line('value of n in outer block is:'||outer.n);
    dbms_output.put_line('value of n in inner block is:'|| n);      End;      End;

```

Sample Programs

1. Program to illustrate exit

statement. Example:

```

DECLARE
    a number(2) := 10;
BEGIN
    WHILE a < 20
    LOOP
        dbms_output.put_line ('value of a: ' || a);
        a := a + 1;
    IF a > 15 THEN
        EXIT;
    END IF;
    END LOOP;
END;

```

2. Program to illustrate continue statement.

```

DECLARE
    a number(2) := 10;
BEGIN
    WHILE a < 20
    LOOP
        dbms_output.put_line ('value of a: ' || a);
        a := a + 1;
    IF a = 15 THEN
        a := a + 1;
        CONTINUE;
    END IF;
    END LOOP;
END;

```

3. Program to find factorial of a given number using while loop.

```

DECLARE
    i number := 1;
    n number:=6;
    fact number:=1;
BEGIN
    WHILE i<=6
    LOOP
        fact: =fact*i;

```



```

        i:=i+1;
    END LOOP;
    dbms_output.put_line ('factorial of '||n||' is ||fact);
END;

```

4. Program to find factorial of a given number using for loop.

```

DECLARE
    i number := 1;
    n number:=6;
    fact number:=1;
BEGIN
    For I in 1 .. 6 LOOP
        fact:=fact*i;
        i:=i+1;
    END LOOP;
    dbms_output.put_line ('factorial of '||n||' is ||fact);
END;

```

5. Create a table circle with the field radius and area write a PL/SQL program to find area for given radius.

```

Create table circle ( r number(5),
                    Area number(10,2));
DECLARE
    Pi constant number:=3.14;
    r number(6);
    area number(10,2);
BEGIN
    For r in 5 .. 10
    LOOP
        Area:=pi * power(r,2);
        Insert into circle values(r, area);
    END LOOP;
END;
Select *from circle;

```

6. Program to find sum of first 10 natural numbers.

```

DECLARE
    i number := 1;
    sum1 number:=1;
BEGIN
    LOOP
        I:=i+1;
        If i>10 then

```

```

                Exit;
            End if;
            Sum1:=sum1+I;
        END LOOP;
        dbms_output.put_line ('sum of 10 natural numbers is' || sum1);
END;

```

7. Program to add 2 numbers

```

declare
    a number;
    b number;
    c number;
begin
    a:=&a;
    b:=&b;
    c:=a+b;
    dbms_output.put_line ('sum of numbers is'||c);
end;

```

8. Program to find sum of first 100 numbers.

```

declare
    a number;
    s1 number default 0;
begin
    a:=1;
    loop
        s1:=s1+a;
        exit when (a=100);
        a:=a+1;
    end loop;
    dbms_output.put_line ('sum between 1 to 100 is'||s1);
end;

```

9. Program to generate Fibonacci series.

```

declare
    n number:=&n;
    n1 number:=0;
    n2 number:=1;
    n3 number;
begin

```

```

dbms_output.put_line(n1);
dbms_output.put_line(n2);
for i in 3..n
loop
    n3:=n1+n2;
    dbms_output.put_line(n3);

```

n1:=n2;
n2:=n3;

Named

There are several named blocks in PL/SQL. They are

- Procedure
- Function
- Package
- Cursors
- Triggers

Procedure

Definition:

“Procedure is a **subprogram** is a program unit/module that performs a particular action”. Procedure consists of 2 parts. They are

- Specification
- Body of procedure

Creating a Procedure

A procedure is created with the CREATE OR REPLACE PROCEDURE statement.

```
CREATE OR REPLACE PROCEDURE procedure name <parameters> is /as
```

```
BEGIN
```

```
    Statements;
```

```
END procedure name;
```

Where,

procedure-name specifies the name of the procedure.

Parameter list contains 3 parameters. They are IN ,OUT and INOUT. IN parameter is used to send valued to stored procedure. OUT parameter is used to get values from stored procedure.

procedure-body contains the executable part.

Example 1: Program to illustrate creation and execution of a procedure.

```
CREATE OR REPLACE PROCEDURE greetings AS
```

```
BEGIN
```

```
    dbms_output.put_line('Hello World!');
```

```
END;
```

```
/
When above code is executed using SQL prompt.
SQL> EXECUTE greetings;
```

Output:

```
        Hello World
```

PL/SQL procedure successfully completed.

Deleting a Standalone Procedure

A standalone procedure is deleted with the DROP PROCEDURE statement.

Syntax for deleting a procedure is:

```
        DROP PROCEDURE procedure-name;
```

So you can drop greetings procedure by using the following statement:

```
BEGIN
        DROP PROCEDURE greetings;
END;
```

Example 2: Create a procedure to add 2 numbers.

```
create or replace procedure add1(fn in number,sn in number,res out number) as
begin
    res:=fn+sn;
end;
/
procedure created.
```

Now we will write the definition of the procedure.

```
declare
    c number;
begin
    add1(10,20,c);
    dbms_output.put_line ('sum is' ||c); end;
```

Example 3: IN & OUT Mode

This program finds the minimum of two values, here procedure takes two numbers using

IN mode and returns their minimum using OUT parameters.

```
DECLARE
a number;
b number;
c number;
PROCEDURE findMin(x IN number, y IN number, z OUT number) IS BEGIN
    IF x < y THEN
        z:= x;
    ELSE
        z:= y;
```

```

        END IF;
    END;
    BEGIN
        a:= 23;
        b:= 45;

        findMin(a, b, c);

        dbms_output.put_line(' Minimum of (23, 45) : ' || c);
    END;
/

```

When the above code is executed at SQL prompt, it produces the following result:

```
Minimum of (23, 45) : 23
```

PL/SQL procedure successfully completed.

Functions

Definition:

“Function is a named PL/SQL block it is set of statements carrying a particular task.” **Major difference between Procedure and function is**

A PL/SQL Procedure may or may not return a value where as Function must always return a value.

Creating a Function

```

CREATE OR REPLACE FUNCTION function name <parameter name> RETURN data type
IS |AS
BEGIN
    function body
END [function name];

```

Where,

Function-name specifies the name of the function.

REPLACE keyword used to overwrite the function with the same name was already present.

Parameter list contains 3 parameters. They are IN, OUT and INOUT. IN parameter is used to send valued to stored function. OUT parameter is used to get values from stored function.

the function must contain a **return** statement.

RETURN clause specifies that data type you are going to return from the function.

Function-body contains the executable part.

Example 1: Program to illustrate function without arguments.

```

CREATE OR REPLACE function fun1 return varchar2 AS
BEGIN

```

```
        dbms_output.put_line('Hello World!');
END;
/
```

When above code is executed using SQLQuery.

```
SQL> select *from dual;
```

Example 2: Program to illustrate function with arguments.

```
CREATE OR REPLACE function fun2 (s varchar2) return varchar2 AS
BEGIN
```

```
    Return s;
```

```
END;
```

```
/
```

When above code is executed using SQL Query.

```
SQL> select fun2('SSCASC') from dual;
```

Example 3: Program to find sum of 2 numbers using functions.

Create or replace function fun1 (a number, b number) return number

as begin

```
    return(a+b);
```

```
end;
```

```
/
```

function created.

Now we will define body of the function.

```
declare
```

```
    result number;
```

```
begin
```

```
    result:=fun1(10,20);
```

```
    dbms_output.put_line('addition of 2 numbers is'||result);
```

```
end;
```

```
/
```

Example 4: The following is one more example which demonstrates Declaring, Defining, and Invoking a Simple PL/SQL Function that computes and returns the maximum of two values.

```
DECLARE
```

```
a        number;
```

```
b        number;
```

```
c        number;
```

```
z        number;
```

```
BEGIN
```

```
    IF x > y THEN
```

```
        z:= x;
```

```
    ELSE
```

```
        Z:= y;
```

```
    END IF;
```

```
RETURN z;
END;
BEGIN
```

```
    a:= 23;
```

```
    b:= 45;
```

```
    c := findMax(a, b);
```

```
    dbms_output.put_line(' Maximum of (23,45): ' || c);
```

```
END;
```

When the above code is executed at SQL prompt, it produces the following result:

Package

Definition:

Maximum of (23, 45): 78

Package is a collection of procedures and functions.

These are having 2 parts. They are specification and body of the package.

Syntax:

```
CREATE OR REPLACE PACKAGE package name IS |AS
```

```
Procedure procedure name;
```

```
Function function name;
```

```
BEGIN
```

```
    Statements;
```

```
END package name;
```

EXAMPLE 1: Program to illustrate creates and execute package.

```
SQL> create or replace package pack1 as
```

```
    procedure proc1;
```

```
    function fun1 return varchar2;
```

```
end pack1;
```

```
/
```

Package created.

The next step we will write the code for body of the package.

```
create or replace package body pack1 as
```

```
    procedure proc1 as
```

```
    begin
```

```
        dbms_output.put_line('this is procedure concept');
```

```
    end proc1;
```

```
    function fun1 return varchar2 is
```

```
    begin
```

```
        return('hello this is function concept');
```

```
    end fun1;
```

```
end pack1;
```

```
/
```

Package body created.

Now execute procedure and function through package.

```
SQL> execute pack1.proc1; this is procedure concept
```

PL/SQL procedure successfully completed.

```
SQL> select pack1.fun1 from dual;
```

FUN1

Cursors

Cursor represents a structure in memory and is different from cursor variable. When we declare a cursor it gets pointer variable which does not point anything.

When the cursor is open memory is allocated and the cursor will be created.

There are two types of cursors:

- Implicit cursors.
- Explicit cursors

The main use of cursor will be allowing the programmer to retrieve data from a table and perform some actions on the data.

Implicit Cursors:

- Implicit cursors are automatically created by Oracle whenever an SQL statement is executed.
- Programmers cannot control the implicit cursors and the information in it.

In PL/SQL, implicit cursor always has 4 attributes like %FOUND, %ISOPEN, %NOTFOUND, and %ROWCOUNT.

The following table provides the description of the most used attribute

<u>Attribute</u>	<u>Descriptions</u>
%Found	Returns TRUE if an Insert The logical opposite of %FOUND. It returns TRUE if
%NOTFOUND	an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.
%ISOPEN	Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.
%ROWCOUNT	Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement UPDATE, or DELETE statement

Example:

We will be using the CUSTOMERS table we had created and used in the previous chapters. Select * from customers;

```
+----+-----+ +----+-----+ +
| ID | NAME | AGE | ADDRESS | SALARY |
+----+-----+ +----+-----+ +
| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |
| 2 | Khilan | 25 | Delhi | 1500.00 |
| 3 | kaushik | 23 | Kota | 2000.00 |
| 4 | Chaitali | 25 | Mumbai | 6500.00 |
| 5 | Hardik | 27 | Bhopal | 8500.00 |
| 6 | Komal | 22 | MP | 4500.00 |
```

The following program would update the table and increase salary of each customer by 500 and use the SQL%ROWCOUNT attribute to determine the number of rows affected:

```
DECLARE
Total_ rows number(2);
BEGIN
UPDATE customers
SET salary = salary + 500;
IF sql %notfound THEN
dbms_output.put_line('no customers selected');
ELSIF sql%found THEN
Total_ rows := sql%rowcount;
dbms_output.put_line( total _rows || ' customers selected ');
END IF;
END;
```

```
+----+-----+ +----+-----+ +
| 1 | Ramesh | 32 | Ahmedabad | 2500.00 |
```

When the above code is executed at SQL prompt, it produces the following result:
6 customers selected

2	Khilan	25	Delhi	2000.00
3	kaushik	23	Kota	2500.00
4	Chaitali	25	Mumbai	7000.00
5	Hardik	27	Bhopal	9000.00
6	Komal	22	MP	5000.00

PL/SQL procedure successfully completed.

If you check the records in customers table,

you will find that the rows have been updated:

Select * from customers;



Explicit Cursors

Explicit cursors are used to write a query for more than one row.

An explicit cursor should be defined in the declaration section of the PL/SQL Block.

It is created on a SELECT Statement which returns more than one row.

The syntax for creating an explicit cursor is

```
CURSOR cursor_name IS select_statement;
```

Working with an explicit cursor involves four steps:

Declaring the cursor for initializing in the memory

Opening the cursor for allocating memory

Fetching th

Declaring the Cursor

Declaring the cursor defines the cursor with a name and the associated SELECT statement.

Opening the Cursor

Opening the cursor allocates memory for the cursor and makes it ready for fetching the rows returned by the SQL statement into it.

Fetching the Cursor

Fetching the cursor involves accessing one row at a time.

Closing the Cursor

Closing the cursor means releasing the allocated memory.

Example:

Following is a complete example to illustrate the concepts of explicit cursors:

```
DECLARE  
c_id customers.id%type;
```

```

c_name customers.name%type;
c_addr customers.address%type;
CURSOR c_customers is
SELECT id, name, address FROM customers;
BEGIN
OPEN c_customers;
LOOP
FETCH c_customers into c_id, c_name, c_addr;
dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr); EXIT
WHEN c_customers%notfound; END LOOP;

CLOSE c_customers;
END; /

```

Triggers

Definition:

“Triggers are stored programs, which are automatically executed or fired when some events occur.”

Triggers are, in fact, written to be executed in response to any of the following events:

Triggers could be defined on the table, view, schema, or database with which the event is associated.

Benefits of Triggers

Triggers can be written for the following purposes:

Generating some derived column values automatically

| Enforcing referential integrity

| Event logging and storing information on table access

| Auditing

| Synchronous replication of tables

| S

Creating Triggers

The syntax for creating a trigger is:

```

CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER }
{INSERT [OR] | UPDATE [OR] | DELETE} ON table_name [FOR
EACH ROW] WHEN (condition) BEGIN
    Trigger body
END;

```

Where,

- `CREATE [OR REPLACE] TRIGGER trigger_name`: Creates or replaces an existing trigger with the `trigger_name`.
- `{BEFORE | AFTER}`: This specifies when the trigger would be executed.
- `{INSERT [OR] | UPDATE [OR] | DELETE}`: This specifies the DML operation.
- `[ON table_name]`: This specifies the name of the table associated with the trigger.
- `FOR EACH ROW`: This specifies a row level trigger, i.e., the trigger would be executed for each row being affected.
- `WHEN (condition)`: This provides a condition for rows for which the trigger would fire. Trigger executes if the condition is true.

Chapter 8

Relational Data Model and Relational Algebra

This chapter has 2 parts. They are relational data model and relational algebra.

Part 1 concepts are:

- Terminologies about relation like domain, degree, cardinality, tuple and column etc.
- Constraints on relations like domain, key, entity, referential integrity etc.
- Operations on relations like insertion, deletion and updation etc.

All these topics are referred in Relational model, SQL and Normalization.

Relational Algebra

Definition:

The relational algebra is a procedural query language. It consists of a set of operations that take one or two relations as input and produces a new relation as output.

Relational algebra has 2 types of operators.

They are

- Unary operations
- Binary operations
- Additional relational operations

Unary relational Operations

These kinds of operations can be work only on single table.

Operations are

- Select
- Project

The select operation: -

It is used to identify a set of tuples which is a part of a relation and to extract only these tuples out. The select operation selects tuples that satisfy a given predicate or condition.

- It is a unary operation defined on a single relation.
- It is denoted as σ .

Syntax:

σ (condition) tablename ;

Consider the following table "Book" :-

Code:

Acc-no	Yr-pub	title
734216	1982	Algorithm design
237235	1995	Database systems
631523	1992	Compiler design
543211	1991	Programming
376112	1992	Machine design

Example1:- Select from the relation “Book” all the books whose Acc-no is greater than equal to 56782.

Syntax:

$\sigma_{accno \geq 56782}(book)$

The project operation: -

It is used to returns its argument relation with certain attributes left out.

- It is a unary operation defined on a single relation
- It is denoted as Π .

Syntax:

$\Pi_{col1,col2...col n}(tablename)$

Example:- List all the Title and Acc-no of the “Book” relation.

$\Pi_{accno, title}(book)$

Example2: The table **E** (for **EMPLOYEE**)

nr	name	salary
1	John	100
5	Sarah	300
7	Tom	100

Selection

The same table **E** (for **EMPLOYEE**) as above.

SQL	Result	Relational algebra									
<pre>select * from E where salary < 200</pre>	<table border="1"> <thead> <tr> <th>nr</th> <th>name</th> <th>salary</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>John</td> <td>100</td> </tr> <tr> <td>7</td> <td>Tom</td> <td>100</td> </tr> </tbody> </table>	nr	name	salary	1	John	100	7	Tom	100	$\sigma_{salary < 200}(E)$
nr	name	salary									
1	John	100									
7	Tom	100									
<pre>select *from E where salary < 200 and nr >= 7</pre>	<table border="1"> <thead> <tr> <th>nr</th> <th>name</th> <th>salary</th> </tr> </thead> <tbody> <tr> <td>7</td> <td>Tom</td> <td>100</td> </tr> </tbody> </table>	nr	name	salary	7	Tom	100	$\sigma_{salary < 200 \text{ and } nr \geq 7}(E)$			
nr	name	salary									
7	Tom	100									

Projection:

SQL	Result	Relational algebra
<pre>select salary from E</pre>	salary 100 300	$\Pi_{\text{salary}}(E)$

<pre>select nr, salary from E</pre>	<table><thead><tr><th>nr</th><th>salary</th></tr></thead><tbody><tr><td>1</td><td>100</td></tr><tr><td>5</td><td>300</td></tr><tr><td>7</td><td>100</td></tr></tbody></table>	nr	salary	1	100	5	300	7	100	$\Pi_{\text{nr, salary}}(E)$
nr	salary									
1	100									
5	300									
7	100									

Binary operators

These kinds of operations require minimum 2 tables or sets.

Set operators combine the results of 2 queries into a single result. Queries contains set operators are called compound queries.

The set operators in SQL are

- ❖ Union
- ❖ Intersect
- ❖ Minus
- ❖ Cartesian product.

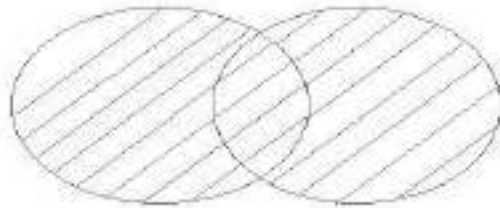
1. Union:

Union returns all distinct (unique) tuples selected by both the queries.

It is used when we need some attributes that appear in either or both of the two relations.

It is represented by

- It is denoted as **U**.



The union operator selects only unique values by default.

Syntax 1:

Select column names from table1 **union** select column names from table2;

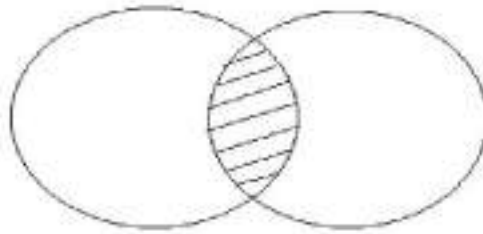
2. Intersect:

This operation returns only those rows which are common in both the queries.

It finds tuples in both the relations.

- It is denoted as \cap .

It is represented by



Syntax:

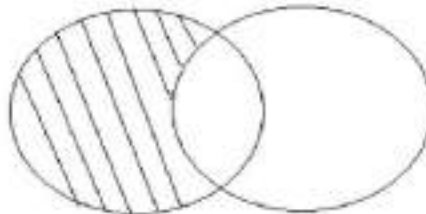
Select column names from table1 **intersect** select column names from table2;

3. **Minus:**

This operation returns only those rows that are distinct in first table but not present in second table.

It finds tuples in one relation but not in other.

- It is denoted as $-$



Syntax:

Select column names from table1 **minus** select column names from table2;

4. **The Cartesian product operation: -**

It allows combining information from two relations.

- It is denoted as $r \times s$ where r and s are relations.

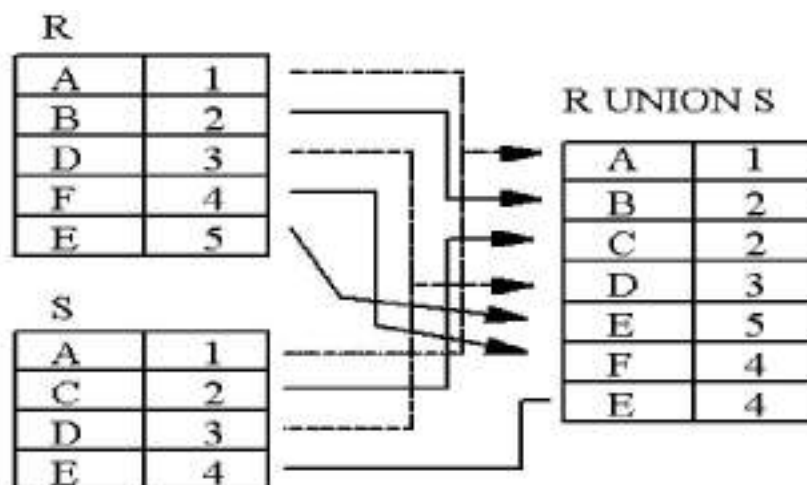
Examples of Set operators:

SET Operations – requirements

For set operations to function correctly the relations R and S must be union compatible. Two relations are union compatible if

- they have the same number of attributes
- the domain of each attribute in column order is the same in both R and S.

UNION Example



INTERSECTION Example

R

A	1
B	2
D	3
F	4
E	5

R INTERSECTION S

A	1
D	3

S

A	1
C	2
D	3
E	4

DIFFERENCE Example

R

A	1
B	2
D	3
F	4
E	5

R DIFFERENCE S

B	2
F	4
E	5

S

A	1
C	2
D	3
E	4

S DIFFERENCE R

C	2
E	4

CARTESIAN PRODUCT

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

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

CARTESIAN PRODUCT example

R

A	1
B	2
D	3
F	4
E	5

S

A	1
C	2
D	3
E	4

R CROSS S

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

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

Notation

The operations are used in relational algebra have their own symbols.

Operation	Symbol	Operation	Symbol
Projection		Cartesian product	
Selection		Join	
Renaming		Left outer join	
Union		Right outer join	
Intersection		Full outer join	
Assignment		Semi join	

Outer join

- An outer-join operation does the same thing as natural join, except that it preserves tuples without a match in the other relation.
- The tuple that does not have a match uses null values to populate the attributes from the second relation.
- There are three types of outer join, based on which non-matching tuples “stick around” after the operation: –

Explain relational database design using ER to Relational Mapping

Seven-step algorithm to convert the basic ER model constructs into relations

Step 1: Mapping of Regular Entity Types

For each regular entity type, create a relation R that includes all the simple attributes of E Called entity relations. Each tuple represents an entity instance

Step 2: Mapping of Weak Entity Types

For each weak entity type, create a relation R and include all simple attributes of the entity type as attributes of R Include primary key attribute of owner as foreign key attributes of R

Step 3: Mapping of Binary 1:1 Relationship Types

For each binary 1:1 relationship type • Identify relations that correspond to entity types participating in R.

Step 4: Mapping of Binary 1:N Relationship Types

For each regular binary 1:N relationship type • Identify relation that represents participating entity type at N-side of relationship type • Include primary key of other entity type as foreign key in S • Include simple attributes of 1:N relationship type as attributes of S

Step 5: Mapping of Binary M:N Relationship Types

For each binary M:N relationship type • Create a new relation S • Include primary key of participating entity types as foreign key attributes in S • Include any simple attributes of M:N relationship type

Step 6: Mapping of Multivalued Attributes

For each multivalued attribute • Create a new relation • Primary key of R is the combination of A and K • If the multivalued attribute is composite, include its simple components

Step 7: Mapping of N-array Relationship Types

For each n-array relationship type Create a new relation S to represent R include primary keys of participating entity types as foreign keys • Include any simple attributes as attributes