Database Management System

STUDENT ADVISORY

Dear Students,

Please be informed that the notes provided by the institute offer a concise presentation of the syllabus. While these notes are helpful for an overview and quick revision, We would strongly suggest that you refer to the prescribed textbooks / Reference book for a comprehensive understanding and thorough preparation of all exams and writing in the examination.

Best regards,

LJ Polytechnic.

પ્રિય વિદ્યાર્થીઓ,

તમને જાણ કરવામા આવે છે કે સંસ્થા દ્વારા પ્રદાન કરવામાં આવેલી નોંધો અભ્યાસક્રમની સંક્ષિપ્ત પ્રસ્તુતિ આપે છે. આ નોંધો વિહંગાવલોકન અને ઝડપી પુનરાવર્તન માટે મદદરૂપ હોઈ શકે છે તેમ છતા, અમે ભારપૂર્વક સૂચન કરીએ છીએ કે વિદ્યાર્થી તમામ પરીક્ષાઓ અને પરીક્ષામાં લેખનની વ્યાપક સમજણ અને સંપૂર્ણ તૈયારી માટે માત્ર સૂચવેલા પાઠ્યપુસ્તકો/સંદર્ભ પુસ્તકનો સંદર્ભ લો.

એલજે પોલિટેકનિક.

Unit-1
Introduction of Database Management System

➢ Introduction

- DBMS stand for Database Management System.
- Database management system is a software, which is used to manage the Database.
- DBMS provides an interface to perform various operations like Database creation, storing data in it, updating data, creating a table in the Database and a lot more.
- It provides protection and security to the Database. In the case of multiple users, it also maintain data consistency.

➢ Terms of Database

- **Data**
  Word 'Data' is originated from the word 'datum' that means 'single piece of information.' It is plural of the word datum.
  For example, age salary, result, etc.
- **Information**
  Information is organized data or proceed data.
- **Database**
  The Database is a collection of inter-related data.
- **Metadata**
  Metadata is data about data.
  Data such as table name, column name, data type, authorized user, user access privileges for any table is called metadata for that table.
- **Data Dictionary**
  Data dictionary is an information repository, which contains metadata.
  It is usually a part of the system catalog.
  This can involves information such as table name, column names, data types, size and constraints.
- **Data Warehouse**
  Data Warehouse is an information repository which stored data.
  It is design to facilitate reporting and analysis.
- **Data Items(Field)**
  A field is a character or group of characters (alphabetic or numeric) that have specific meaning.
  It represents in the Database by a value.
  For example customer id, name, society, and city are all fields for customer data.
- **Record**
  A record is a collection of logically related field
  Here, each field in a record contains a fixed size and fixed data type.
  Example collection of field -id, name, address, and city form a record for customer.
- **Files**
  A file is collection of related record.
  These records generally arranged in specific sequence.

➢ Operations Performed on Database

- **Create**
  It create the Database such as table.

- **Insert**
  It inserting the data into existing table.
- **Modify**
  It modify the data in existing table.
- **Remove**
  It remove the data in existing table.
- **Retrieve**
  It retrieving data from existing table.
- **Destroy**
  It destroy the Database such as table.

➢ Need of Databases

- **Manages large amounts of data**
  A Database stores and manages a large amount of data on a daily basis.
  This would not be possible using any other tool such as a spreadsheet as they would simply not work.
- **Accurate**
  A Database is pretty accurate as it has all sorts of build in constraints, checks etc.
  This means that the information available in a Database is guaranteed to be correct in most cases.
- **Easy to update data**
  In a Database, it is easy to update data using various Data Manipulation languages (DML) available.
- **Security of data**
  Databases have various methods to ensure security of data. There are user logins required before accessing a Database and various access specifiers. These allow only authorised users to access the Database.
- **Data integrity**
  This is ensured in Databases by using various constraints for data. Data integrity in Databases makes sure that the data is accurate and consistent in a Database.
- **Easy to research data**
  It is very easy to access and research data in a Database.
  This is done using Data Query Languages (DQL) which allow searching of any data in the Database and performing computations on it.

➢ Applications of Database

- **Railway Reservation System**
  The Database is required to keep record of ticket booking, train departure, and arrival status. Also if trains get late then people get to know it through Database updates.
- **Library Management System**
  There are thousands of books in the library so it is very difficult to keep record of all the books in a copy or register. So DBMS is used to maintain all the information related to book issuing dates, name of the book, author, and availability of the book.
- **Banking**
  We make thousands of transactions through banks daily and we can do this without going to the bank. So how banking has become so easy that by sitting at home we can send or get money through banks. That is all possible just because of DBMS that manages all the bank transactions.
- **Universities and colleges**
  Examinations are done online today and universities and colleges maintain all these records through DBMS. Student’s registrations details, results, courses, and grades all the information are stored in the Database.
- **Telecommunications**
  Any telecommunication company cannot even think about its business without DBMS. DBMS is necessary for these companies to store the call details and monthly post-paid bills.

- **Finance**
  Those days have gone far when information related to money was stored in registers and files. Today the time has totally changed because there are many things to do with finance like storing sales, holding information and financial statement management, etc.
- **Military**
  The military keeps records of millions of soldiers and it has millions of files that should be kept secured and safe. As DBMS provides a big security assurance to the military information so it is widely used in militaries. One can easily search for all the information about anyone within seconds with the help of DBMS.
- **Online Shopping**
  Online shopping has become a big trend these days. No one wants to go to the shops and waste his time. Everyone wants to shop from home. So all these products are added and sold only with the help of DBMS. Purchase information, invoice bills, and payment, all of these are done with the help of DBMS.
- **Human Resource Management**
  Big firms have many workers working under them. The human resource management department keeps records of each employee's salary, tax, and work through DBMS.

➢ Data Administrator(DA) and Database Administrator(DBA)

➢ Data Administrator(DA)

- Data Administrator (DA) is a person who is responsible for processing data into a convenient data model.
- The person is in charge of figuring out which data is relevant to be stored in the Database.
- Data Administrator (DA) is less of a technical role and more of a business role with some technical knowledge.
- This role is also known as Data Analyst. Therefore, it is mostly a high-level function, which is responsible for the overall management of data resources in an organization.

➢ Database Administrator(DBA)

- Database Administrator (DBA) is a person who creates updates and maintains the Database.
- It is more of a wide role as a Data Administrator(DA) might be someone who is hired to create, maintain, and backup the Database, optimize the Database for high performance, or someone who helps in integrating Databases into applications.
- The major skills required to be an excellent Database Administrator (DBA) are troubleshooting, logical thought process, and a strong will to learn as it involves a vast area.
- This role is also known as Database Coordinator or Database Programmer.
- **Responsibilities**
- **Defining conceptual schema and Database creation.**
  The DBA defines the logical schema of the database. A schema refers to the overall logical structure of the database.
- **Storage structure and access method definition**
  The DBA decides how the data is to be represented in the stored database.
- **Assisting Application Programmers.**
  The DBA provides assistance to application programmers to develop application programs.
- **Physical organization modification.**
  The DBA determines which users needs access to which part of the database.
- **Monitoring Performance**
  The DBA monitors performance of the system,The DBA ensures that better performance is maintained by making changes in physical or logical schema if required.

Data Administrator(DA)	Database Administrator(DBA)
Data Administrator (DA) is also known as data analyst.	Database Administrator (DBA) is also known as Database coordinator or Database programmer.
Data Administrator (DA) monitors data flow across the organization.	Database Administrator (DBA) ensures Database security.
Data Administrator (DA) requires excellent data analysing, expression of ideas, and strategic thinking.	Database Administrator (DBA) mostly require logical thought process, troubleshooting and will to learn.
Data Administrator (DA) is less of a technical role and more of a business role.	Database Administrator (DBA) is a wide role as it has multiple responsibilities
Main tasks include data planning, definition, architecture and management etc.	Main tasks include Database design, construction, security, backup and recovery, performance tuning etc.
Generally it owns the data.	Whereas it owns the Database.

➢ Difference Between Data and Information

Data	Information
Data can be any character, number, images, words, text, means little or nothing to a man. When data is processed.	Organized and presented in a context to make it useful is called information
Data alone can never be significant	Information is always important by itself.
Data is based on record, observations, etc.	Information is based on analysis of data.
For example, id, name, dob. address, phone, etc.	For Example— gtu website,

➢ Data Dictionary:-

- A data dictionary contains metadata i.e. data about the Database.
- It is a collection of names, definitions, and attributes about data elements that are being used or captured in a Database, information system, or part of a research project.
- The data dictionary is very important as it contains information such as what is in the Database, who is allowed to access it, where is the Database physically stored etc.
- The users of the Database normally don't interact with the data dictionary, it is only handled by the Database Administrator(DBA)
- Here some components of Data Dictionary
- Entities
- Attributes
- Relationships
- Keys

➢ Types of Data Dictionary

- **Active Data Dictionary:-**
- The DBMS software manages the active data dictionary automatically.
- The modification is an automatic task and most RDBMS has active data dictionary.
- It is also known as integrated data dictionary.
- **Passive Data Dictionary**
- Managed by the users and is modified manually when the Database structure change.
- Also known as non-integrated data dictionary.

➢ Data Warehouse:-

- "Data Warehouse is a subject-oriented, integrated, and time-variant store of information in support of management's decisions."
- A Data Warehouse (DW) is a relational Database that is designed for query and analysis rather than transaction processing.
- It includes historical data derived from transaction data from single and multiple sources.
- A **Data Warehouse** is separate from DBMS, it stores a huge amount of data, which is typically collected from multiple heterogeneous sources like files, DBMS, etc.
- The goal is to produce statistical results that may help in decision makings.
- For example, a college might want to see quick different results, like how the placement of CS students has improved over the last 10 years, in terms of salaries, counts, etc.
- Data stored in Data Warehouse possess following characteristics:
- **Subject-Oriented**
  A data warehouse target on the modelling and analysis of data for decision-makers.
  Therefore, data warehouses typically provide a concise and straightforward view around a particular subject, such as customer, product, or sales, instead of the global organization's ongoing operations.
  This is done by excluding data that are not useful concerning the subject and including all data needed by the users to understand the subject
- **Integrated**
  A data warehouse integrates various heterogeneous data sources like RDBMS, flat files, and online transaction records.
  It requires performing data cleaning and integration during data warehousing to ensure consistency in naming conventions, attributes types, etc., among different data sources.
- **Time-Variant**
  Historical information is kept in a data warehouse.
  For example, one can retrieve files from 3 months, 6 months, 12 months, or even previous data from a data warehouse.
  These variations with a transactions system, where often only the most current file is kept.
- **Non-Volatile**
  The data warehouse is a physically separate data storage, which is transformed from the source operational RDBMS.
  The operational updates of data do not occur in the data warehouse, i.e., update, insert, and delete operations are not performed.
  Non-Volatile defines that once entered into the warehouse, and data should not change.

File-Oriented System and Database Management System

File-Oriented System

- A file-oriented approach to storage creates files in sets as needed when a business sells products or services.
- Each file operates independently from other files in storage. This means files do not share information with other files stored in the system.

Advantages of File-Oriented System

- **Back up**

  It is possible to take faster and automatic back up of Database stored in files of computer-based system.
- **Compactness**

  It is possible to store data compactly.

  For example, to store all words of English Dictionary , only few kilobytes of memory is required in computer-based systems.
- **Data Retrieval**

  Computer-based system provide enhanced data retrieval techniques to retrieve data stored in files in easy and efficient way.
- **Editing**

  It is easy to edit any information stored in computers in form of files.
- **Remote Access**

  In computer-based system, it is possible to access data remotely. Therefore, to access data, it is not necessary for a user to remain present at location where these data are kept.
- **Sharing**

  Data stored in files of computer-based systems can be shared among multiple users at a same time.

Disadvantages of File-Oriented System

- **Data Redundancy (Duplication of data)**

  It is possible that the same information may be duplicated in different files, this lead data redundancy. Data redundancy results in memory wastage.
- **Data Inconsistency**

  Due to data redundancy, it is possible that will not be in consistent state.
- **Difficulty in Accessing Data**

  Accessing data is not convenient and efficient in file processing system.
- **Limited Data Sharing**

  Data are scattered in various files also different files may have different formats.And these files may be stored in different folders may be of different computers of different departments.
- **Integrity Problems**

  Data integrity means that the data contained in the Database is both correct and consistent.
- **Atomicity Problems**

  Any operation on Database must be atomic. This means it must happen entirely or not at all.

Database Management System:-

- Database management system is a software, which is used to manage the Database. For example: MySQL, Oracle, etc are a very popular commercial Database, which is used in different applications.
- DBMS provides an interface to perform various operations like Database creation, storing data in it, updating data, creating a table in the Database and a lot more.
- It provides protection and security to the Database. In the case of multiple users, it also maintains data consistency

Characteristics of DBMS

- It uses a digital repository established on a server to store and manage the information.
- It can provide a clear and logical view of the process that manipulates data.
- DBMS contains automatic backup and recovery procedures.
- It contains ACID properties which maintain data in a healthy state in case of failure.
- It can reduce the complex relationship between data.
- It is used to support manipulation and processing of data.

Advantages of DBMS

- **Controls Database redundancy**
  It can control data redundancy because it stores all the data in one single Database file and that recorded data is placed in the Database.
- **Data sharing**
  In DBMS, the authorized users of an organization can share the data among multiple users.
- **Easily Maintenance**
  It can be easily maintainable due to the centralized nature of the Database system.
- **Reduce time**
  It reduces development time and maintenance need.
- **Backup**
  It provides backup and recovery subsystems which create automatic backup of data from hardware and software failures and restores the data if required.
- **multiple user interface**
  It provides different types of user interfaces like graphical user interfaces, application program interfaces

Disadvantages of DBMS

- **Cost of Hardware and Software**
  It requires a high speed of data processor and large memory size to run DBMS software.
- **Size**
  It occupies a large space of disks and large memory to run them efficiently.
- **Complexity**
  Database system creates additional complexity and requirements.
- **Higher impact of failure**
  Failure is highly impacted the Database because in most of the organization, all the data stored in a single Database and if the Database is damaged due to electric failure or Database corruption then the data may be lost forever.

Unit-2

Database Architecture

➢ Introduction

Database architecture can be seen as a single tier or multi-tier. However, logically, Database architecture is of two types like 1-tier architecture, 2-tier architecture and 3-tier architecture.

➢ Schema, Subschema and Instances:-

- **Schema**
  The overall logical design of Database is known as Schema.
- **Subschema**
  The subset of schema and inherits the same property of schema is called as Subschema.
- **Instance**
  The collection of data items values or content of Database at any point of time is called Instance.
  Database systems have several schemas in the three-tire Database architecture as follows:
- **Internal Schema**
  Describe physical storage structures and access paths.
- **Conceptual Schema**
  Describe structure and constraints for the Database.
- **External Schema**
  Describe particular view of the Database.

➢ 1-Tier Architecture

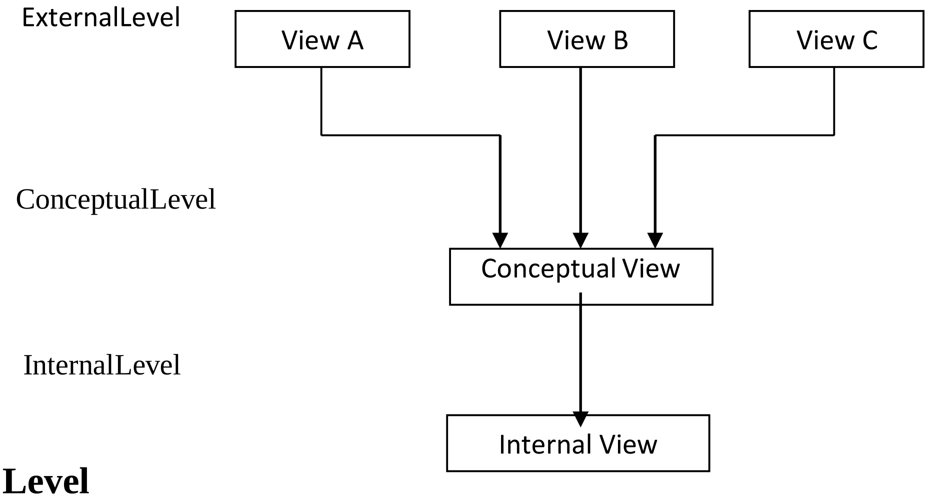
- In this architecture, the Database is directly available to the user. It means the user can directly sit on the DBMS and uses it.
- Any changes done here will directly be done on the Database itself. It does not provide a handy tool for end users.

➢ 2-Tier Architecture

- The 2-Tier architecture is same as basic client-server. In the two-tier architecture, applications on the client end can directly communicate with the Database at the server side. For this interaction, API's like: ODBC, JDBC are used.
- The user interfaces and application programs are run on the client-side.
- The server side is responsible to provide the functionalities like: query processing and transaction management.
- To communicate with the DBMS, client-side application establishes a connection with the server side.

➢ Three-tier ANSI-SPARC Database Architecture.

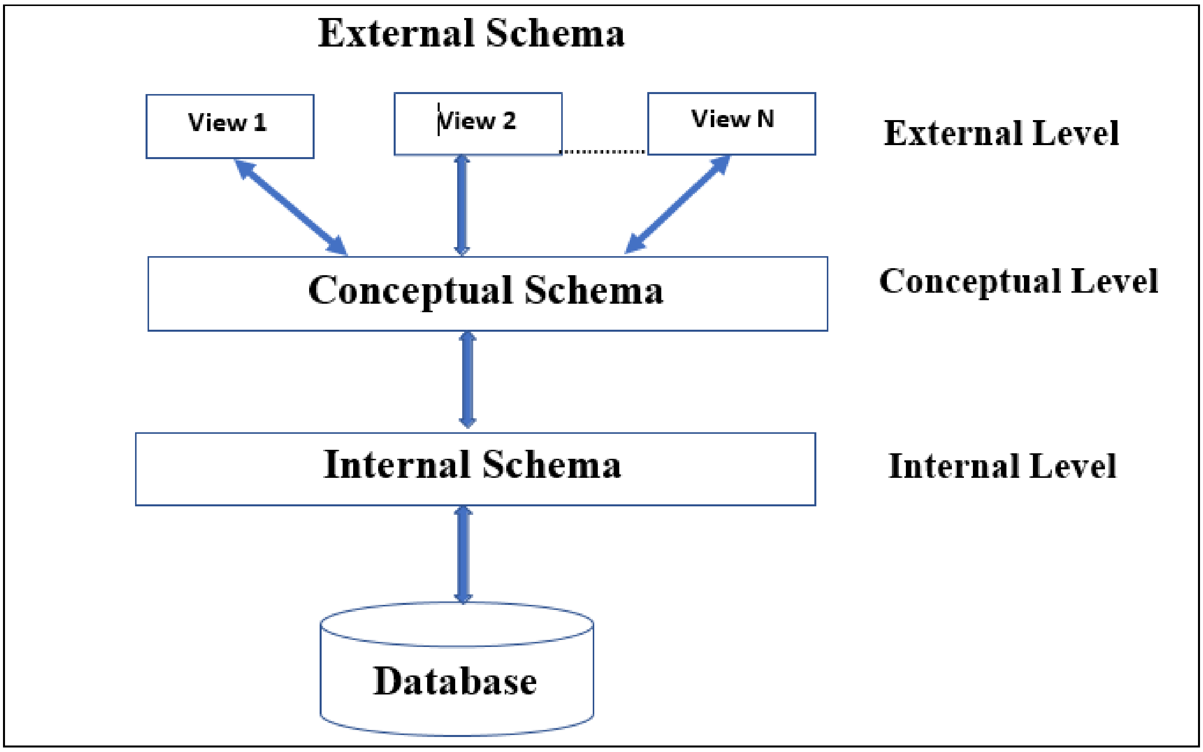
- In 1975, American National Standards Institute-Standard Planning and Requirement Committee introduced three-tier or three-level architecture for the Database system.
- Database Architecture Levels *OR* Three level Data abstraction
- **The ANSI SPARC architecture divided into three levels:**
- External level
- Conceptual level
- Internal level



- **Internal Level**
- This is the lowest level of the data abstraction.
- It describes how the data are actually stored on storage devices.
- It is also known as a physical level.
- The internal view is described by internal schema.
- Internal schema consists of definition of stored record, method of representing the datafield and access method used.
- **Conceptual Level**
- This is the next higher level of the data abstraction.
- It describes what data are stored in the Database and what relationships exist amongthose data.
- It is also known as a logical level.
- Conceptual view is defined by conceptual schema. It describes all records andrelationship.
- **External Level**
- This is the highest level of data abstraction.It is also known as view level.
- It describes only part of the entire Database that a particular end user requires. External view is describes by external schema.
- External schema consists of definition of logical records, relationship in the external view
- and method of deriving the objects from the conceptual view.This object includes entities, attributes and relationship.

➢ Data Independence:-

- Data independence refers characteristic of being able to modify the schema at one level of the Database system without altering the schema at the next higher level.
- Data Independence is ability to modify the one schema definition without affecting another schema definition in the next higher level.
- In other words, the application programs do not depend on any particular physical representation or access technique.
- The Data Independence in achieved by DBMS through the use of the three-tier architecture of the data abstraction.



- There are two types of Data Independence as shown in figure.

➢ Physical Data Independence

- Physical Data Independence is ability to modify the physical schema without affecting next higher level schema definition.
- Physical data independence can be defined as the capacity to change the internal schema without having to change the conceptual schema.
- If we do any changes in the storage size of the Database system server, then the Conceptual structure of the Database will not be affected.
- Physical data independence is used to separate conceptual levels from the internal levels.
- Physical data independence occurs at the logical interface level.

➢ Logical Data Independence

- Logical Data Independence is ability to modify the logical schema definition without affecting next higher level schema definition.
- Logical data independence refers characteristic of being able to change the conceptual schema without having to change the external schema.
- Logical data independence is used to separate the external level from the conceptual view.
- If we do any changes in the conceptual view of the data, then the user view of the data would not be affected.
- Logical data independence occurs at the user interface level.

➢ Mappings

- The three levels of DBMS architecture don't exist independently of each other.
- There must be correspondence between the three levels i.e. how they actually correspond with each other.
- DBMS is responsible for correspondence between the three types of schema. This correspondence is called Mapping.
- There are basically two types of mapping in the Database architecture:

- **Conceptual/ Internal Mapping**

- **External / Conceptual Mapping**

➢ Conceptual/ Internal Mapping

- The Conceptual/ Internal Mapping lies between the conceptual level and the internal level.
- Its role is to define the correspondence between the records and fields of the conceptual level and files and data structures of the internal level.

➢ External/ Conceptual Mapping

- The external/Conceptual Mapping lies between the external level and the Conceptual level.

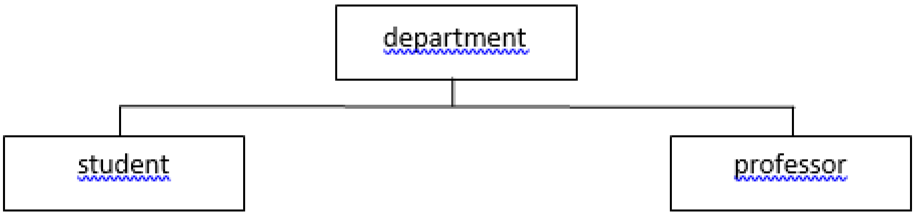
- Its role is to define the correspondence between a particular external and the conceptual view.

➢ Data Model

- Data Model is the modeling of the data description, data semantics, and consistency constraints of the data.
- It provides the conceptual tools for describing the design of a Database at each level of data abstraction.
- Therefore, there are following four data models used for understanding the structure of the Database:
- A Database model is a type of data model that defines the logical structure of aDatabase.It determines how data can be stored, accessed and updated in a Database management system
- The most popular example of a Database model is the relational model, which uses atable-based format.
- A Database model is a type of data model that defines the logical structure of aDatabase. It determines how data can be stored, accessed and updated in a Database management system
- The most popular example of a Database model is the relational model, which uses a table-based format.
- **Type of Database Models are**
- Hierarchical Model
- Network Model
- Entity-relationship Model
- Relational Model
- Object-oriented Database model

➢ Hierarchical Model

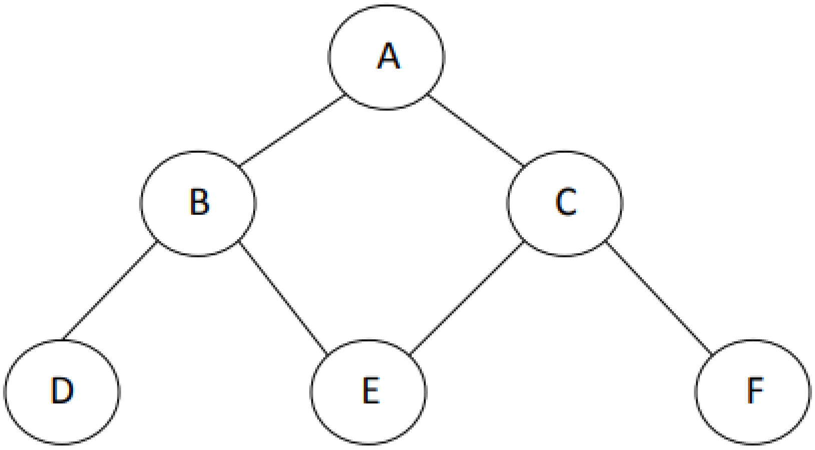
- The hierarchical model organizes data into a tree-like structure, where each record has a single parent or root.



- The hierarchy starts from the Root data, and expands like a tree, adding child nodes to the parent nodes.
- In hierarchical model, data is organized into tree-like structure with one-to-many relationship between two different types of data, for example, one department can have many professors and many students.
- **Advantage of Hierarchical Model**
- Fast and efficient data retrieval
- Easy to add/delete information.
- Predictable data structure.
- **Disadvantage of Hierarchical Model.**
- Limited flexibility
- Difficult to maintain and update
- Lack of standardization

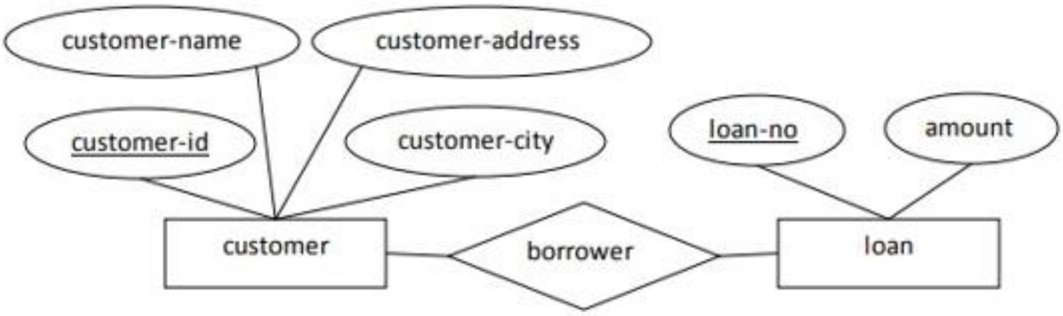
➢ Network Model

- This is an extension of the hierarchical model, allowing many-to-many relationships in a tree-like structure that allows multiple parents.



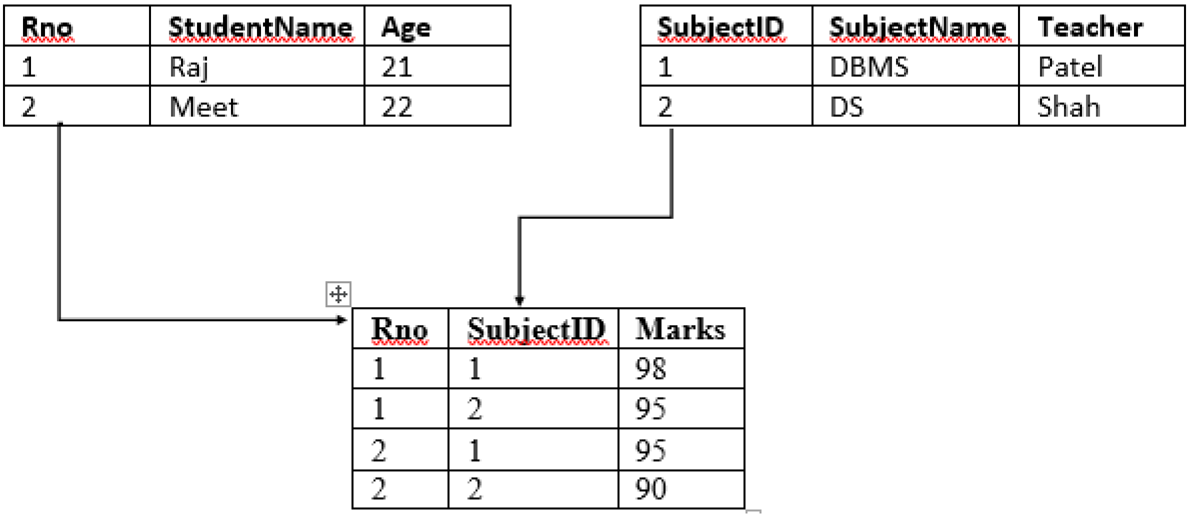
- **Advantage of Network Model**
- Conceptual simplicity
- Capability to handle more relationship types
- Ease of data access
- Data integrity
- **Disadvantage of Network Model**
- System complexity
- Operational Anomalies
- Absence of structural independence.

- **Entity-relationship Model**
- In this Database model, relationships are created by dividing object of interest into entity and its characteristics into attributes.



- **Advantage of E-R model**
- Exceptional conceptual simplicity
- Visual representation
- Effective communication tool
- **Disadvantage of E-R model**.
- Limited constraint representation
- Limited relationship representation
- No data manipulation language

- **Relational Model**
- In this model, data is organized in two-dimensional tables and the relationship is maintained by storing a common attribute.



Rno	StudentName	Age
1	Raj	21
2	Meet	22

SubjectID	SubjectName	Teacher
1	DBMS	Patel
2	DS	Shah

Rno	SubjectID	Marks
1	1	98
1	2	95
2	1	95
2	2	90

- **Advantage of Relational Model**
- Simplicity of model
- Ease of use
- Accuracy
- Data Integrity
- Normalization
- Security
- **Disadvantage of Relational Model**
- Maintenance Problem
- Cost
- Physical Storage
- Lack of Scalability

- **Object-oriented Database model**
- This data model is another method of representing real world objects.
- It considers each object in the world as objects and isolates it from each other.
- It groups its related functionalities together and allows inheriting its functionality to other related sub-groups.
- **Advantage and disadvantage of Object-Oriented Model**
- Efficient representation of complex data structures
- Improved code reuse and modularity
- Support for data integration
- Flexibility and adaptability
- Improved performance
- **Disadvantage OF Object-Oriented Model**
- Complexity and learning curve
- Limited vendor support
- Limited support for SQL.

Unit-3
Structured Query Language

❖ **Give full name of SQL and Explain different data types of oracle.**

Answer:

SQL: Structured Query Language

Data Types of Oracle:

1. Numerical Data Type
2. Character/String Data Type
3. Date Data Type
4. Binary Data Type

1. Numerical Data Type:

- Used to store zero, negative and positive numerical values. These values can be fixed-point or floating-point.
- Numbers can range between 1.0x10^{-130} and 1.0x10^{126}

2. Character/String Data Type:

1. CHAR (size):

- Stores character string of fixed length.
- Size represents the no. of characters (length) to be stored. Default size is 1.
- Maximum length is 255 characters.

2. VARCHAR (size)/ VARCHAR2 (size):

- Stores character strings of variable length.
- More flexible than CHAR.
- No default size will be considered. So, size must be specified explicitly.
- Maximum length is 2000 characters.

3. LONG:

- Stores large amount of character strings of variable length.
- Maximum length is up to 2 GB.
- Only one column per table can be defined as LONG.
- A LONG column cannot be indexed.
- A LONG column cannot be used as argument or return value in procedures or functions.

3. Date Data Type:

- Used to store date and time.
- The standard format is DD-MON-YY to store date, such as 01-APR-07.
- The current date and time can be retrieved using function SYSDATE.
- Addition and subtraction operation are possible using number constants and other dates. For example, SYSDATE + 7 will return same day of next week

4. Binary Data Type:

1. RAW:

- Stores binary type data.
- Maximum length is up to 255 bytes.

2. LONG RAW:

- Stores large amount of binary type data.
- Often referred as binary large object (BLOB).
- Maximum length is up to 2 GB.
- A LONG RAW column cannot be indexed.

❖ **Explain DDL commands with suitable example.**

Answer:

- It is a set of SQL commands used to create, modify and delete database objects such as tables, views, indices, etc.
- It is normally used by DBA and database designers.
- It provides commands like – CREATE, ALTER, DROP, TRUNCATE

CREATE:

Syntax:

```
Create table tablename(
ColumnName1 datatype(size),
ColumnName2 datatype(size),
:
:
ColumnNameN datatype(size)
);
```

Description:

- A tableName should be unique.
- A tableName and columnName must start with alphabet.
- TableName and columnName are not case sensitive.

Example:

Create an Account table having three columns for account number, balance, branch name.

Input:

```
CREATE TABLE ACCOUNT(
ANO VARCHAR2(3),
BALANCE NUMBER(9),
BNAME VARCHAR2(10)
);
```

Output:

```
Table Created
```

ALTER:

- The ALTER TABLE command can be used to alter the schema of a table.

Adding New Columns:

- This command adds a new column or columns in an exiting table.

Example:

Add new column acc_type in an Account table.

Input:

```
ALTER TABLE ACCOUNT
ADD (ACC_TYPE CHAR(10));
```

Output:

```
Table altered.
```

Dropping Columns:

- This command deletes an existing column from the table along with data held by the column.

Example:

Drop a column acc_type from the Account table.

```
ALTER TABLE ACCOUNT
DROP COLUMN ACC_TYPE;
```

Output:

```
Table altered.
```

Modifying Existing Columns:

- An existing column can be modified in one of the two ways:
  1. Change the **datatype** of a column
  2. Change the **size** of a column

Example:

Alter column *ano* to hold maximum 20 characters with datatype VARCHAR2 in table Account.

Input:

```
ALTER TABLE ACCOUNT
MODIFY(ANO VARCHAR(20));
```

Output:

```
Table altered.
```

DROP:

- The DROP TABLE command drops the specified table.
- This means, all records along with structure (or schema) of the table will be destroyed.

Example:

Remove the table Account along with the data held.

Input:

```
DROP TABLE ACCOUNT;
```

Output:

```
Table dropped.
```

TRUNCATE:

- The TRUNCATE TABLE command empties given table completely.
- Truncate operation drops and re-creates the tables.

Example:

Truncate the table Account.

Input:

```
TRUNCATE TABLE ACCOUNT;
```

Output:

```
Table truncated.
```

❖ **Explain DML commands with suitable example.**

Answer:

- It is a set of SQL commands used to insert, modify and delete data in a database.
- It is normally used by general users who are accessing database via pre-developed applications.
- It provides commands like - INSERT, UPDATE, DELETE, LOCK

INSERT:

- To insert user data into tables, “INSERT INTO…” SQL statement is used.
- This statement creates a new row (record) in a table, and stores the values into respective columns.

Syntax:

```
INSERT INTO tableName (column1, column2, …., columnN)
VALUES (expression1, expression2, …., expressionN);
```

- Characters and date constants are enclosed within *single* quotes.

- There is one to one mapping between column specified and expressions passed.

Example:

```
INSERT INTO ACCOUNT(ANO, BALANCE, BNAME)
VALUES ('A01',5000,'VVN');
```

Output:

```
1 row created.
```

UPDATE:

- The update command can be used to change or modify data values in a table.
- It can be used to update either all rows or a set of rows from a table.

Syntax:

```
UPDATE tableName
SET column1 = expression1, column2 = expression2
WHERE condition;
```

- UPDATE command updates rows from the table, and displays message regarding how many rows have been updated.
- The SET clause specifies which column data to modify.
- An expression can be a constant value, a variable, or some expression and it specifies new value for related columns.

Example:

Change branch name from ‘ksad’ to’ karamsad’ for each account which belongs to ‘ksad’ branch.

Input:

```
UPDATE ACCOUNT
SET BNAME='KARAMSAD'
WHERE BNAME='KSAD';
```

Output:

```
2 rows updated.
```

DELETE:

- The DELETE command can be used to remove either all rows of a table, or a set of rows from a table.

Example:

Delete all accounts which belong to ‘ksad’ branch.

Input:

```
DELETE FROM ACCOUNT
WHERE BNAME='KSAD';
```

Output:

```
2 rows deleted.
```

- ❖ **Explain DISTINCT, GROUP BY, ORDER BY, HAVING with purpose and example.**

Answer:

DISTINCT:

- The DISTINCT keyword removes duplicate rows from the result. It can be used only with SELECT statement.

Syntax:

```
SELECT DISTINCT column1, column2, …, column FROM tableName;
```

Example:

Display only branch names for an Account table with distinct.

Input:

```
SELECT DISTINCT BNAME FROM ACCOUNT;
```

Output:

```
BNAME
----------
VVN
ANAND
KSAD
```

Order By:

- This statement retrieves data in a sorted manner.
- Rows are sorted based on values of columns specified with ORDER BY clause.
- By default, order is considered an Ascending order. To sort data in descending order, it is necessary to specify DESC as an order.

Example:

Display all accounts according to their branch name.

Input:

```
SELECT * FROM ACCOUNT
ORDER BY BNAME;
```

Output:

```
ANO   BALANCE    BNAME
---   ----------  ----------
A03     7000      ANAND
A02     6000      KSAD
A04     8000      KSAD
A05     6000      VVN
A01     5000      VVN
```

Display all accounts sorted in descending order according to their balance.

Input:

```
SELECT * FROM ACCOUNT
ORDER BY BALANCE DESC;
```

Output:

```
ANO   BALANCE    BNAME
---   ----------  ----------
A04     8000      KSAD
A03     7000      ANAND
A05     6000      VVN
A02     6000      KSAD
A01     5000      VVN
```

Group By and Having:

- The GROUP BY clause groups records based on distinct values for specified columns.

- The HAVING clause filters groups based on the specified condition.

Example:

Display total balance of different accounts for ‘vvn’ branch.

Input:

```
SELECT BNAME, SUM(BALANCE) "TOTAL BALANCE" FROM ACCOUNT
GROUP BY BNAME
HAVING BNAME='VVN';
```

Output:

```
BNAME       TOTAL BALANCE
----------  -------------
VVN          22000
```

❖ Differentiate ORDER BY v/s GROUP BY.

ORDER BY	GROUP BY
It sorts the result set either in ascending or descending order.	It is used to group the rows that have the same values.
It controls the presentation of columns.	It controls the presentation of rows.
The attribute can be under aggregate function under GROUP BY statement.	The attribute cannot be under aggregate function under GROUP BY statement.
It is always used after the GROUP BY clause in the SELECT statement.	It is always used before the ORDER BY clause in the SELECT statement.
It is not mandatory to use aggregate functions in ORDER BY.	It is mandatory to use aggregate functions in the GROUP BY.
Here, the result-set is sorted based on the column's attribute values, either ascending or descending order.	Here, the grouping is done based on the similarity among the row's attribute values.
It is not allowed in CREATE VIEW statement.	It may be allowed in CREATE VIEW statement.

❖ Write short note on: Dual table, Sysdate, Arithmetic Operators, Logical Operators

Answer:

Dual Table:

- Dual table is a dummy table. It is provided by Oracle itself, i.e. it is in-built table.
- It contains only one row and one column with single value **x**.
- The dual table can be used to display output of operations which do not refer to any table.

Example:

Calculate the result of mathematical expression 5*3.

Input:

```
SELECT 5 * 3 FROM dual;
```

Output:

```
5 * 3
------
5
```

Sysdate:

- SYSDATE is a “pseudo column” that contains the current date and time.
- A pseudo column is a column that returns a value when it is selected, but it is not an actual column in the table, i.e. it does not belong to any specific table.

Example:

Display the current date.

Input:

```
SELECT SYSDATE FROM dual;
```

Output:

```
SYSDATE
--------------
14-FEB-13
```

Arithmetic Operators:

- Arithmetic operations can be performed while viewing table data, or while manipulating table data with insert, update or delete operation.
- SQL supports following operators for arithmetic operation.

+	Addition	-	Subtraction
*	Multiplication	/	Division
( )	Enclosed operation		

Example:

Input:

```
SELECT 5 * 3 FROM dual;
```

Output:

```
5 * 3
------
5
```

Logical Operators:

AND Operator:

- The AND operator is used to combine two or more conditions in WHERE and HAVING clauses.
- AND requires all the conditions to be true to consider entire clause true.

Example:

Find out the customer who belongs to ‘Anand’ city and has salary less than 17000.

Input:

```
SELECT * FROM EMPLOYEE
WHERE CITY = 'ANAND' AND SALARY < 17000;
```

Output:

```
EID    ENAME       BIRTHDATE    SALARY    CITY
----   ----------  ---------    ----------  ----------
E02    GOPI        15-AUG-83     15000      ANAND
```

OR Operator:

- The OR operator is also used to combine two or more conditions in WHERE and HAVING clauses.
- OR requires any one condition to be true to consider entire clause true.

Example:

Find out the customer who belongs to either ‘Anand’ city or ‘Surat’ city.

Input:

```
SELECT * FROM EMPLOYEE
```

```
WHERE CITY='ANAND' OR CITY='SURAT';
```

Output:

```
EID    ENAME         BIRTHDATE    SALARY    CITY
----   ----------    ---------    ----------  ----------
E02    GOPI          15-AUG-83    15000     ANAND
E04    VAISHALI      23-MAR-85    25000     SURAT
E05    LAXMI         14-FEB-83    18000     ANAND
```

NOT Operator:

- The NOT operator is used to negate the result of any condition or group of conditions.

Example:

Find out the customers who do not belong to ‘Anand’ city.

Input:

```
SELECT * FROM EMPLOYEE
WHERE NOT (CITY='ANAND');
```

Output:

```
EID    ENAME         BIRTHDATE    SALARY    CITY
----   ----------    ---------    ----------  ----------
E01    TULSI         26-JAN-82    12000     AHMEDABAD
E03    RAJSHREE      31-OCT-84    20000     VADODARA
E04    VAISHALI      23-MAR-85    25000     SURAT
```

❖ **Write short note on: Relational Operators, Range Searching Operators, Set Searching Operators, Character Operators**

Answer:

Relational Operators:

- SQL supports following relational operators to perform comparisons among values.

=	Equals	!= or < >	Not equals
<	Less than	< =	Less than or equal to
>	Greater than	> =	Greater than or equal to

- These relation operators compare expressions and return one of three values: True, False or Unknown.

Example:

Find out the customer who has salary less than 17000.

Input:

```
SELECT * FROM EMPLOYEE
WHERE SALARY < 17000;
```

Output:

```
EID    ENAME         BIRTHDATE    SALARY    CITY
----   ----------    ---------    ----------  ----------
E02    GOPI          15-AUG-83    15000     ANAND
```

Range Searching Operators:

BETWEEN Operator:

- Selects rows that contain values within a specified lower and upper limit.
- The lower and upper limit values must be linked with the keyword AND.

Example:

Display employees having salary between 15000 and 20000.

Input:

```
SELECT * FROM EMPLOYEE
WHERE SALARY BETWEEN 15000 AND 20000;
```

Output:

```
EID   ENAME       BIRTHDATE   SALARY    CITY
----  ----------  ---------   ----------  ----------
E02   GOPI        15-AUG-83    15000    ANAND
E03   RAJSHREE    31-OCT-84    20000    VADODARA
E05   LAXMI       14-FEB-83    18000    ANAND
E06   SHIVALI     05-SEP-84    20000
```

- To select data that do not fall in some particular range, NOT can be used with BETWEEN operator.

Example:

Input:

```
SELECT * FROM EMPLOYEE
WHERE SALARY NOT BETWEEN 15000 AND 20000;
```

Output:

```
EID   ENAME       BIRTHDATE   SALARY    CITY
----  ----------  ---------   ----------  ----------
E01   TULSI       26-JAN-82   12000     AHMEDABAD
E04   VAISHALI    23-MAR-85   25000     SURAT
```

Set Searching Operator:

IN Operator:

- Selects rows that contain any value given in a set.
- Can be used when there is a need to use multiple OR conditions.

Example:

Display employees coming from city ‘Ahmedabad’, ‘Vadodara’ or ‘Surat’.

Input:

```
SELECT * FROM EMPLOYEE
WHERE CITY IN ('AHMEDABAD','VADODARA','SURAT');
```

Output:

```
EID   ENAME       BIRTHDATE   SALARY    CITY
----  ----------  ---------   ----------  ----------
E01   TULSI       26-JAN-82   12000     AHMEDABAD
E03   RAJSHREE    31-OCT-84    20000    VADODARA
E04   VAISHALI    23-MAR-85    25000    SURAT
```

- To select data that do not belong to some particular set, NOT can be used with IN operator.

Example:

Display employees not coming from city ‘Ahmedabad’, ‘Vadodara’ or ‘Surat’.

Input:

```
SELECT * FROM EMPLOYEE
WHERE CITY NOT IN ('AHMEDABAD','VADODARA','SURAT');
```

Output:

```
EID   ENAME      BIRTHDATE    SALARY    CITY
----  ---------- ---------    ----------  ----------
E02   GOPI       15-AUG-83     15000     ANAND
E05   LAXMI      14-FEB-83     18000     ANAND
```

Character Operators:

LIKE Operator:

- Select rows that contain values similar to a given pattern.
- Can be used with character data type.
- Pattern can be formed by using two wild cards characters:
    i. % (Modulo) allows matching with any string having any number of characters, including zero.
    ii. _ (Underscore) allows matching with any single character.

Example:

Display employees whose name have ‘a’ as a second character.

Input:

```
SELECT * FROM EMPLOYEE
WHERE ENAME LIKE '_A%';
```

Output:

```
EID   ENAME      BIRTHDATE    SALARY    CITY
----  ---------- ---------    ----------  ----------
E03   RAJSHREE   31-OCT-84     20000     VADODARA
E04   VAISHALI   23-MAR-85     25000     SURAT
E05   LAXMI      14-FEB-83     18000     ANAND
```

❖ **Write sort note on (with example): Aggregate Functions, Numeric Functions**

Answer:

Aggregate Functions:

1. MAX (columnName)

- Returns maximum value for a given column.

Input:

```
SELECT MAX(SALARY) "MAX SALARY" FROM EMPLOYEE;
```

Output:

```
MAX SALARY
----------
     25000
```

2. MIN (columnName)

- Returns minimum value for given column.

Input:

```
SELECT MIN(SALARY) "MIN SALARY" FROM EMPLOYEE;
```

Output:

```
MIN SALARY
----------
     12000
```

3. **SUM ([distinct / all] columnName)**

- Returns sum of all values for a given column.

Example:

```
SELECT SUM(SALARY), SUM(DISTINCT SALARY) FROM EMPLOYEE;
```

Output:

```
SUM(SALARY)       SUM(DISTINCTSALARY)
-----------       -------------------
     110000       90000
```

4. **AVG ([distinct / all] columnName)**

- Returns average of all values for a given column.

Example:

Input:

```
SELECT AVG(SALARY), AVG(DISTINCT SALARY) FROM EMPLOYEE;
```

Output:

```
AVG(SALARY)       AVG(DISTINCTSALARY)
-----------       -------------------
 18333.3333       18000
```

5. **COUNT (*)**

- Returns number of rows in a table including duplicates and having null values.

Example:

Input:

```
SELECT COUNT(*) FROM EMPLOYEE;
```

Output:

```
COUNT(*)
----------
         6
```

6. **COUNT ([distinct / all] columnName)**

- Returns number of rows where column does not contain null value.

Example:

Input:

```
SELECT COUNT(*) "C*", COUNT (CITY) "CC", COUNT(DISTINCT CITY) "CDC"
FROM EMPLOYEE;
```

Output:

```
        C*         CC        CDC
---------- ---------- ----------
         6          5          4
```

Numeric Function:

1. **ABS (n)**
    - Returns absolute value of 'n' This means, negative numbers are converted to positive numbers.

Example:

Input:

```
SELECT ABS(-25), ABS(25) FROM DUAL;
```

Output:

```
ABS(-25)    ABS(25)
----------  ----------
      25          25
```

2. **SQRT (n)**
    - Returns square root of 'n'. Here, 'n' cannot be a negative number.

Example:

Input:

```
SELECT SQRT(25), SQRT(35) FROM DUAL;
```

Output:

```
SQRT(25)    SQRT(35)
----------  ----------
       5    5.91607978
```

3. **POWER (m, n)**
    - Returns m raised to nth power, i.e, returns m^{n}
    - Also, n must be an integer value.

Example:

Input:

```
SELECT POWER(3,2), POWER(4,3) FROM DUAL;
```

Output:

```
POWER(3,2)  POWER(4,3)
----------  ----------
       9          64
```

4. **MOD (m, n)**
    - Returns remainder of a m divided by n operation.

Example:

Input:

```
SELECT MOD(5,2), MOD(5,3) FROM DUAL;
```

Output:

```
MOD(5,2)    MOD(5,3)
----------  ----------
       1           2
```

5. **CEIL (n)**
    - Returns the largest integer value that is greater than or equal to n.

Example:

Input:

```
SELECT CEIL(25.2), CEIL(25.7), CEIL(-25.2) FROM DUAL;
```

Output:

```
CEIL(25.2)  CEIL(25.7)  CEIL(-25.2)
----------  ----------  -----------
```

```
            26                     26                  -25
```

6. **FLOOR (n)**
   - Returns the smallest integer value that is less than or equal to n.

Example:

Input:

```
SELECT FLOOR(25.2), FLOOR(25.7), FLOOR(-25.2) FROM DUAL;
```

Output:

```
FLOOR(25.2)          FLOOR(25.7)          FLOOR(-25.2)
-----------          -----------          ------------
       25                      25                      -26
```

7. **ROUND (m, n)**
   - Returns m, rounded to n places to the right of a decimal point.
   - If n is omitted, m is rounded to 0 places.
   - If n is negative, m is rounded to n places to the left of a decimal point.
   - n must be an integer value.

Example:

Input:

```
SELECT ROUND(157.732,2), ROUND(157.732), ROUND(157.732,-2) FROM DUAL;
```

Output:

```
ROUND(157.732,2)   ROUND(157.732)     ROUND(157.732,-2)
----------------   --------------     -----------------
         157.73                        158                     200
```

8. **TRUNC (m, n)**
   - If n is positive, m is truncated to n places to the right of a decimal point.
   - If n is omitted, m is truncated to 0 places.
   - If n is negative, m is truncated to n places to the left of a decimal point.

Example:

Input:

```
SELECT TRUNC(157.732,2), TRUNC(157.732), TRUNC(157.732,-2) FROM DUAL;
```

Output:

```
TRUNC(157.732,2)   TRUNC(157.732)     TRUNC(157.732,-2)
----------------   --------------     -----------------
         157.73              157                     100
```

9. **EXP (n)**
   - Returns e raised to nth power, i.e., returns e, where e = 2.071828183

Example:

Input:

```
SELECT EXP(2), EXP(3) FROM DUAL;
```

Output:

```
EXP(2)          EXP(3)
----------      ----------
 7.3890561      20.0855369
```

10. **LN (n)**
    - Returns natural, or base e, logarithm of n.

Example:

Input:

```
SELECT LN(10), LN(2) FROM DUAL;
```

Output:

```
LN(10)          LN(2)
----------      ----------
2.30258509      0.693147181
```

11. LOG (b, n)

- Returns logarithm of the n in the base of b, or return Log $_b$ n.

Example:

Input:

```
SELECT LOG(10,5), LOG(10,100) FROM DUAL;
```

Output:

```
 LOG(10,5)              LOG(10,100)
----------              -----------
0.698970004                   2
```

12. COS (n), SIN (n), TAN (n)

- Returns trigonometric cosine, sine and tangent values for n; where n is an angle in radius.

Example:

Input:

```
SELECT COS(3.1415), SIN(3.1415), TAN(3.1415) FROM DUAL;
```

Output:

```
COS(3.1415)             SIN(3.1415)             TAN(3.1415)
-----------             -----------             -----------
      -1  .             0. 000092654            -0.00009265
```

13. COSH (n), SINH (n), TANH (n)

- Return hyperbolic cosine, sine and tangent values for n; where n is in radius.

Example:

Input:

```
SELECT COSH(3.1415), SINH(3.1415), TANH(3.1415) FROM DUAL;
```

Output:

```
COSH(3.1415)            SINH(3.1415)            TANH(3.1415)
------------            ------------            ------------
  11.5908833            11.5476654              0.996271387
```

14. SIGN (n)

- Returns -1, if n is negative; 0 if n is zero; and 1 if n is positive.

Example:

Input:

```
SELECT SIGN(-25), SIGN(0), SIGN(25) FROM DUAL;
```

Output:

```
 SIGN(-25)      SIGN(0)         SIGN(25)
----------      ----------      ----------
      -1                0               1
```

❖ **Write short note on (with example): Character Functions, Date Functions**

Answer:

Character Function:

1. Length (str)

- Returns length, i.e. number of characters, of str.

Example:

Input:

```
SELECT LENGTH('INDIA') FROM DUAL;
```

Output:

```
LENGTH('INDIA')
---------------
              5
```

2. **Lower (str)**

- Returns str with all letters in lower case.

Example:

Input:

```
SELECT LOWER('SACHIN Ramesh tendulkar') FROM DUAL;
```

Output:

```
LOWER('SACHINRAMESHTEND
-----------------------
sachin ramesh tendulkar
```

3. **Upper (str)**

- Returns str with all letters in upper case.

Example:

Input:

```
SELECT UPPER ('SACHIN Ramesh tendulkar') FROM DUAL;
```

Output:

```
UPPER('SACHINRAMESHTEND
-----------------------
SACHIN RAMESH TENDULKAR
```

4. **Initcap (str)**

- Returns str with the first letter of each word in upper case.

Example:

Input:

```
SELECT INITCAP ('SACHIN Ramesh tendulkar') FROM DUAL;
```

Output:

```
INITCAP('SACHINRAMESHTE
-----------------------
Sachin Ramesh Tendulkar
```

5. **Substr (str, pos, length)**

- Returns a portion of str, beginning at pos and going up to length characters.

Example:

Input:

```
SELECT SUBSTR ('SACHIN Ramesh tendulkar',8,6) FROM DUAL;
```

Output:

```
SUBSTR
------
```

```
Ramesh
```

6. **Lpad (str, n, str2)**
   - Returns str, left padded with str2 up to length n.

Example:

Input:

```
SELECT LPAD ('INDIA',10,'*') FROM DUAL;
```

Output:

```
LPAD('INDI
----------
*****INDIA
```

7. **Rpad (str, n, str2)**
   - Returns str, right padded with str2 up to length n.

Example:

Input:

```
SELECT RPAD ('INDIA',10,'*') FROM DUAL;
```

Output:

```
RPAD('INDI
----------
INDIA*****
```

8. **Ltrim (str, str2)**
   - Remove characters from the left of str. Characters will be removed up to the first character not in set.

Example:

Input:

```
SELECT LTRIM('Sumita','uSae') FROM DUAL;
```

Output:

```
LTRI
----
mita
```

9. **Rtrim (str, set)**
   - Remove characters from the right of str. Characters will be removed up to the first character not in set.

Example:

Input:

```
SELECT RTRIM('Sumita','tab') FROM DUAL;
```

Output:

```
RTRI
----
Sumi
```

10. **Translate (str, from_set, to_set)**
    - Characters of str that occur in from set are translated to the corresponding characters in the to_set.

Example:

Input:

```
SELECT TRANSLATE ('abc12efg3','1234','XYZW') FROM DUAL;
```

Output:

```
TRANSLATE
---------
abcXYefgZ
```

11. Replace (str, from_set, to_set)

- Replace is similar to translate, but it works on strings rather than set of characters.
- It replaces entire sub-string instead of individual characters as in translate.

Example:

Input:

```
SELECT REPLACE ('abc123efg','123','XYZ') FROM DUAL;
```

Output:

```
REPLACE('
---------
abcXYZefg
```

12. Ascii (char)

- Returns the ASCII code of a char.

Example:

Input:

```
SELECT ASCII ('a'), ASCII ('A') FROM DUAL;
```

Output:

```
ASCII('a')    ASCII('A')
----------    ----------
        97            65
```

Date Function:

1. ADD_MONTHS (date, n)

- Returns new date after adding n months in date specified by date.
- If n is a negative, then n month will be subtracted.

Example:

Input:

```
select add_months(SYSDATE, 3), add_months (SYSDATE, -3) from dual;
```

Output:

```
ADD_MONTH     ADD_MONTH
---------     ---------
23-OCT-19     23-APR-19
```

2. MONTHS_BETWEEN (date1, date2)

- Returns number of months between date1 and date2.
- Subtract date2 from date1 to find out the difference of months.

Example:

Input:

```
SELECT MONTHS_BETWEEN ('31-MAR-13', '31-DEC-12') FROM DUAL;
```

Output:

```
MONTHS_BETWEEN('31-MAR-13','31-DEC-12')
---------------------------------------
                                      3
```

3. LAST_DAY (date)

- Returns the last date of the month specified by date.

Example:

Input:

```
SELECT LAST_DAY ('14-FEB-13') FROM DUAL;
```

Output:

```
LAST_DAY(
---------
28-FEB-13
```

4. **NEXT_DAY (date, day)**

- Returns the date of next named week-day specified by day relative to date.

Example:

Input:

```
SELECT NEXT_DAY('31-JUL-13', 'SUNDAY') FROM DUAL;
```

Output:

```
NEXT_DAY(
---------
04-AUG-13
```

5. **ROUND (date, format)**

- Returns rounded date according to format.

Example:

Input:

```
SELECT ROUND (TO_DATE('31-DEC-12 03:30:45 PM', 'DD-MON-YY HH:MI:SS
PM')) FROM DUAL;
```

Output:

```
ROUND(TO_
---------
01-JAN-13
```

6. **TRUNC (date, format)**

- Returns truncated date according to format.

Example:

Input:

```
SELECT TRUNC (TO_DATE('31-DEC-12 03:30:45 PM', 'DD-MON-YY HH:MI:SS
PM')) FROM DUAL;
```

Output:

```
TRUNC(TO_
---------
31-DEC-12
```

7. **NEW_TIME (date, zone1, zone2)**

- Returns the date after converting it from time zone 1 to time zone 2.

Example:

Input:

```
SELECT NEW_TIME (TO_DATE('31-DEC-12 12:00:00 AM', 'DD-MON-YY
HH:MI:SS PM'), 'GMT', 'PST')  FROM DUAL;
```

Output:

```
NEW_TIME(
---------
30-DEC-12
```

- **Write sort note on (with example): Conversion Functions, Miscellaneous Functions**

Answer:

Conversion Function:

1. **Converting Character to Number:**

   TO_NUMBER (str):

   - Converts a value of a character data type, expressing a number, to NUMBER data type, i.e. converts CHAR or VARCHAR2 to NUMBER.
   - Returns equivalent numeric value to str.

Example:

Input:

```
SELECT TO_NUMBER ('1234.56') FROM DUAL;
```

Output:

```
TO_NUMBER('1234.56')
--------------------
             1234.56
```

2. **Converting Number to Character:**

   TO_CHAR (str):

   - Converts a numeric value to a character data type, using optional format.

Example:

Input:

```
SELECT TO_CHAR (123456,'09,99,999') FROM DUAL;
```

Output:

```
TO_CHAR(12
----------
 01,23,456
```

3. **Converting Date to Character:**

   TO_CHAR (date, format)

   - Converts a DATE value date to a CHAR value, using format.

Example:

Input:

```
SELECT TO_CHAR (SYSDATE, 'DD Month, YYYY') FROM DUAL;
```

Output:

```
TO_CHAR(SYSDATE,'DDMONTH,YYYY')
----------------------------------------
23 July     , 2019
```

4. **Converting Character to Date:**

   TO_DATE (str, format)

   - Converts a character value, i.e. str, to a DATE value.

Example:

Input:

```
SELECT TO_DATE ('31 December, 2012', 'DD-MON-YY') FROM DUAL;
```

Output:

```
TO_DATE('
---------
31-DEC-12
```

Miscellaneous Function:

1. UID

- Returns an integer value corresponding to the UserID of the user currently logged in.

Example:

Input:

```
SELECT UID FROM DUAL;
```

Output:

```
       UID
----------
         5
```

2. USER

- Returns the user name of the user currently logged in.

Example:

Input:

```
SELECT USER FROM DUAL;
```

Output:

```
USER
------------------------------
SYSTEM
```

3. GREATEST (exp1, exp2, …, expN)

- Returns the greatest expression.
- Can be used with numeric, character as well as date related data.

Example:

Input:

```
SELECT GREATEST (11,32,7), GREATEST ('11','32','7') FROM DUAL;
```

Output:

```
GREATEST(11,32,7) G
----------------- -
                32 7
```

4. LEAST (exp1, exp2, …, expN)

- Similar to GREATEST function, but returns the least expression.

Example:

Input:

```
SELECT LEAST (11,32,7), LEAST ('11','32','7') FROM DUAL;
```

Output:

```
LEAST(11,32,7) LE
-------------- --
```

```
        7 11
```

5. **DECODE (value, if1, then1, if2, then2, …,else)**
   - DECODE is similar to if-then-else construct in programming languages.
   - A value is the value to be tested. It is compared with if part, and whenever match is found, corresponding then part is returned. If match not found with any of the if, then else will be returned.

Example:

Input:

```
SELECT DECODE ('SATUR', 'SUN', 'HOLIDAY', 'SATUR', 'HALFDAY', 'FULL DAY') FROM DUAL;
```

Output:

```
DECODE(
-------
HALFDAY
```

6. **NVL (exp1, exp2)**
   - Returns exp2, if exp1 is null. Returns exp1, if it is not null.

Example:

Input:

```
SELECT NVL (5,10), NVL(null,10) FROM DUAL;
```

Output:

```
 NVL(5,10)    NVL(NULL,10)
----------    ------------
     5            10
```

7. **VSIZE (exp)**
   - Returns the storage size of exp in Oracle.

Example:

Input:

```
SELECT VSIZE(25), VSIZE('INDIA'), VSIZE(TO_DATE('31-DEC-12')) FROM DUAL;
```

Output:

```
 VSIZE(25)    VSIZE('INDIA')        VSIZE(TO_DATE('31-DEC-12'))
----------    --------------------  ----------------------------------------
2                   5                                  7
```

❖ **What are constraints? Give classification of Constraints.**

Answer:

"A constraint is a rule that restricts the values that may be present in the database."

- The data stored in database should be valid, correct and consistent. So, when data is manipulated, some rules (constraints) should be followed.
- Oracle provides various kinds of constraints to ensure validity, correctness and consistency of data in a database.
- These constraints can be broadly classified into two categories.
  1. Entity Integrity Constraint
  2. Referential Integrity Constraint

1. Entity Integrity Constraint

"Entity integrity constraints are constraints that restrict the values in row of an individual table."

These constraints are further divided into two sub-categories:

1. **Domain Integrity Constraint**
2. **Key Constraint**

Domain Integrity Constraint:

- Specifies that the value of each column must belong to the domain of that column.
- For example, a balance for any account in banking system should not be negative value.
- Oracle provides two constraints – NOT NULL and CHECK – to provide domain integrity.

Key Constraint:

- Ensure that there must be some way to distinguish two different rows of the same table uniquely.
- Oracle provides two constraints – UNIQUE and PRIMARY KEY – to ensure this kind of uniqueness between two different rows.

2. Referential Integrity Constraint

"Referential integrity constraint specifies that a row in one table that refers to another row must refer to an existing row in that table."

- Oracle provides FOREIGN KEY constraint to ensure such kind of consistency.

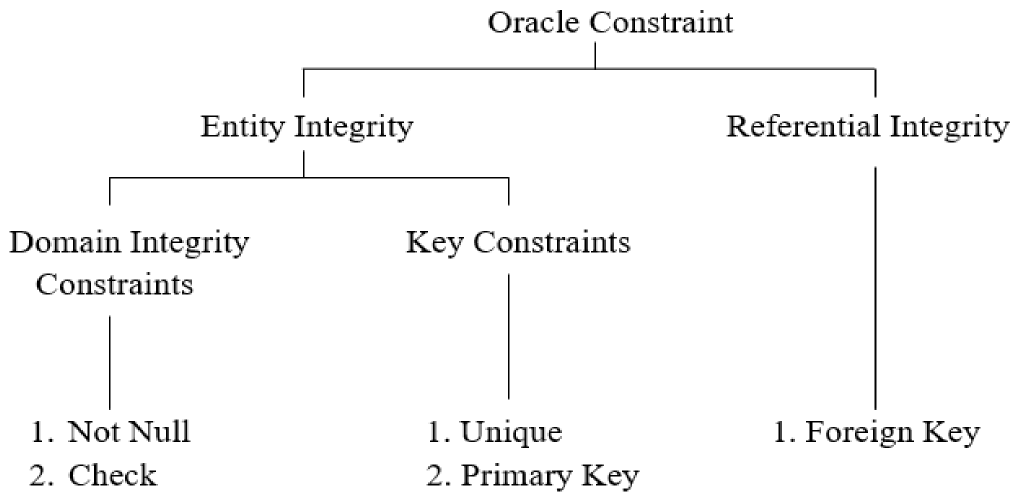


Figure: Classification of Oracle Constraints

❖ **Explain concept of Naming a constraint with example.**

Answer:

- It is also possible to assign some user defined name to a constraint.

Example:

Create an Account table having *ano* as a primary key and assign name 'prime_key' to this constraint.

Input:

```
CREATE TABLE ACCOUNT(
ANO VARCHAR2(3)  CONSTRAINT PRIME_KEY PRIMARY KEY,
BALANCE NUMBER (9),
BNAME CHAR(10)
);
```

Output:

```
Table created.
```

Example:

Create an Account table, having constraint named 'for_key' defining *bname* as a foreign key, referring to Branch table, defined at table level.

Input:

```
CREATE TABLE ACCOUNT(
ANO VARCHAR2(3) PRIMARY KEY,
BALANCE NUMBER (9),
BNAME CHAR(10),
constraint FOR_KEY FOREIGN KEY (BNAME) REFERENCES BRANCH (BNAME)
);
```

Output;

```
Table created.
```

- **Explain how to alter schema in case of constraints.**

Answer:

- ALTER TABLE can also be used to add, modify or drop constraint in an existing table.

Add Constraint:

- This command adds a new constraint in an existing table.

Example:

Consider that Account table doesn’t have any primary key. Alter this table by adding primary key on column *ano*.

Input:

```
ALTER TABLE ACCOUNT
ADD PRIMARY KEY (ANO);
```

Output:

```
Table altered.
```

Drop Constraint:

- A constraint can be dropped directly, or can be dropped by using pre-assigned name to the constraint.

Example:

Drop primary key from an Account table.

Input:

```
ALTER TABLE ACCOUNT
DROP PRIMARY KEY;
```

Output:

```
Table altered.
```

- **Explain Domain Integrity Constraints with example.**

Answer:

- Specifies that the value of each column must belong to the domain of that column.
- Oracle provides two constraints – NOT NULL and CHECK – to provide domain integrity.

NOT NULL Constraint:

- There may be records in table that do not contain any value for some fields.
- In other words, a NULL value represents an empty field.
- A NULL value indicates ‘not applicable’, ‘missing’, or ‘not known’.
- A NULL value is distinct from zero or other numeric value for numerical data.
- A NULL value is also distinct from a blank space for character data.

- But, sometimes it is required that a field cannot be left empty. For example, a balance column in Account table must have some value.
- A column, defined as a NOT NULL, cannot have a NULL value. In other words, such column becomes a mandatory column and cannot be left empty for any record.

Example:

Create an Account table having column balance as a mandatory column.

Input:

```
CREATE TABLE ACCOUNT(
ANO VARCHAR2(3),
BALANCE NUMBER (9) NOT NULL,
BNAME CHAR(10)
);
```

Output:

```
Table created.
```

Input:

```
INSERT INTO ACCOUNT
VALUES ('A01',NULL,'ANAND');
```

Output:

```
ERROR at line 2:
ORA-01400: cannot insert NULL into ("SYSTEM"."ACCOUNT"."BALANCE")
```

CHECK Constraint:

- The CHECK constraint is used to implement business rule. This constraint is also referred as business rule constraint.
- The CHECK constraint is bound to a particular column.
- Once a CHECK constraint is implemented, any insert or update operation on that table must follow this constraint, i.e. condition provided by this constraint.

Example:

Create an Account table which doesn’t allow negative values as a balance for any account.

Input:

```
CREATE TABLE ACCOUNT(
ANO VARCHAR2(3),
BALANCE NUMBER (9) CHECK (NOT (BALANCE < 0)),
BNAME CHAR(10)
);
```

Output:

```
Table created.
```

Input:

```
INSERT INTO ACCOUNT VALUES ('A01',-5000,'VVN');
```

Output:

```
ERROR at line 1:
ORA-02290: check constraint (SYSTEM.SYS_C006994) violated
```

Example:

Create an Account table which accepts ‘vvn’,’ksad’ or ‘anand’ only as a branch name.

Input:

```
CREATE TABLE ACCOUNT(
ANO VARCHAR2(3),
BALANCE NUMBER (9),
BNAME CHAR(10),
CHECK ( BNAME IN ('VVN','KSAD','ANAND'))
);
```

Output:

```
Table created.
```

Input:

```
INSERT INTO ACCOUNT VALUES ('A01',5000,'AHMEDABAD');
```

Output:

```
ERROR at line 1:
ORA-02290: check constraint (SYSTEM.SYS_C006995) violated
```

❖ Explain Key Constraints with example.

Answer:

- Ensure that there must be some way to distinguish two different rows of the same table uniquely.
- Oracle provides two constraints – UNIQUE and PRIMARY KEY – to ensure this kind of uniqueness between two different rows.

Unique:

- There may be requirement that a column must have unique values. In other words, it is required that a column cannot contain duplicate values.
- For example, in Account table, all account numbers must be unique. This is required to identify all accounts, i.e. records stored in table, uniquely.
- A column, defined as a UNIQUE, cannot have duplicate values across all records.

Example:

Create Account table having column *ano* as a UNIQUE column.

Input: (Column Level)

```
CREATE TABLE ACCOUNT(
ANO VARCHAR2(3) UNIQUE,
BALANCE NUMBER (9),
BNAME CHAR(10)
);
```

Table Level:

```
CREATE TABLE ACCOUNT(
ANO VARCHAR2(3),
BALANCE NUMBER (9),
BNAME CHAR(10),
UNIQUE(ANO)
);
```

Output:

```
Table created.
```

Input:

```
INSERT INTO ACCOUNT
VALUES ('A04',8000,'KSAD')
```

Output:

```
ERROR at line 1:
ORA-00001: unique constraint (SYSTEM.SYS_C006988) violated
```

Primary Key:

"A primary key is a set of one or more columns used to identify each record uniquely in a column."

- A column, defined as a PRIMARY KEY, cannot have duplicate values across all records.
- A column, defined as a PRIMARY KEY, cannot have NULL value.
- Thus, a PRIMARY KEY constraint is a combination of NOT NULL and UNIQUE constraints.

Example:

Create Account table having column *ano* as a PRIMARY KEY column.

Input:(Column Level)

```
CREATE TABLE ACCOUNT(
ANO VARCHAR2(3) PRIMARY KEY,
BALANCE NUMBER (9),
BNAME CHAR(10)
);
```

Table Level:

```
CREATE TABLE ACCOUNT(
ANO VARCHAR2(3),
BALANCE NUMBER (9),
BNAME CHAR(10),
PRIMARY KEY(ANO)
);
```

Output:

```
Table created.
```

Input:

```
INSERT INTO ACCOUNT VALUES ('A01',5000,'VVN');
```

Output:

```
ERROR at line 1:
ORA-00001: unique constraint (SYSTEM.SYS_C006990) violated
```

Input:

```
INSERT INTO ACCOUNT VALUES (NULL,5000,'VVN');
```

Output:

```
ERROR at line 1:
ORA-01400: cannot insert NULL into ("SYSTEM"."ACCOUNT"."ANO")
```

- ❖ **Explain Referential Integrity Constraints with example also explain use of On Delete Cascade.**

Answer:

- A FOREIGN KEY constraint, also referred as referential integrity constraint, is specified between two tables. This constraint is used to ensure consistency among records of the two tables.
- Consider the following two tables – Account and Branch – as shown in following.

Account:

Branch:

ano	**balance**	**Bname**
A01	5000	Vvn
A02	6000	Ksad
A03	7000	Anand
A04	8000	Ksad
A05	6000	vvn

bname	**Baddress**
vvn	Mota bazaar, VVNagar
ksad	Chhota bazaar, Karamsad
anand	Nana bazaar, Anand

- Here, bname is common between both tables and provides link between Account and Branch. It represents which account is related to which branch.
- A FOREIGN KEY constraint is used to maintain such kind of data consistency between two tables. To server this purpose, it uses concept of foreign key.
- “A foreign key is a set of one or more columns whose values are derived from the primary key or unique key of other table.”

- In our case, an attribute *bname* in Account table is referred as foreign key as its values are derived from the branch table where it is defined as a primary key.
- The table, in which a foreign key is defined, is called a **foreign table**, **detail table** or **child table**. The table, of which primary key or unique key is referred, is called a **primary table**, **master table** or **parent table**.

Example:

Create an Account table in such a way that it contains column *bname* as a FOREIGN KEY referring to Branch table.

Input:

```
CREATE TABLE ACCOUNT(
ANO VARCHAR2(3) PRIMARY KEY,
BALANCE NUMBER (9),
BNAME CHAR(10) REFERENCES BRANCH(BNAME)
);
```

Output:

```
Table created.
```

- A table name in REFERENCES clause refers to the Master table.
- If *columnName* from REFERENCES clause is omitted, then the primary key is automatically referred of a table specified in this clause.

ON DELETE CASCADE Option:

- This option overcomes the restriction enforced on Master table for delete operation.
- With this option, when any record from Master table is deleted, all the corresponding records from Detail (Child) table are also deleted automatically.
- Consider that, this option has already been provided while defining a foreign key in Account table. Now, if record with branch name ‘vvn’ is deleted from Branch table, then all the records with ‘vvn’ as a branch name from Account table will also be deleted.

Example:

Create an Account table in such a way that it contains column *bname* as a FOREIGN KEY referring to Branch table defined at table level.

Input:

```
        CREATE TABLE ACCOUNT(
        ANO VARCHAR2(3) PRIMARY KEY,
        BALANCE NUMBER (9),
        BNAME CHAR(10),
        FOREIGN KEY (BNAME) REFERENCES BRANCH (BNAME)
        ON DELETE CASCADE
);
```

Unit-4
Relational Model

❖ Basic terms of Relational Model

- **Domain**: It contains a set of atomic values that an attribute can take.

- **Attribute**: It contains the name of a column in a particular table. Each attribute Ai must have a domain.

- **Tuple:** A row or a record in a relation is referred as a Tuple.

- **Relational schema**: A Relational schema contains the name of the Relation and name of all columns or attributes.

- **Relational key**: In the Relational key, each row has one or more attributes. It can identify the row in the Relation uniquely.

- **Relational instance**: In the Relational database system, the relational instance is represented by a finite set of tuples. Relation instances do not have duplicate tuples.

- **Arity of a Relation:** The total numbers of attributes in relation is referred as Arity of Relation.
- For example, the arity of STUDENT relation is 4.

- **Cardinality of a Relation:** The total numbers of tuples in relation is referred as Cardinality of a Relation.
- For example, the cardinality of a STUDENT relation is 3.

- **Example**:
- STUDENT Relation

ENROLL_NO	**NAME**	**CONTACT_NO**	**CITY**
101	JIYA	7687864543	AHMEDABAD
102	KIYA	9897654512	ANAND
103	HIYA	8767675412	VVN

ENROLL_NO, NAME, CONTACT_NO, ADDRESS are attributes.
The total no. of attributes in a relation i.e arity of relation.
A row in relation is referred as tuple.
The total numbers of tuple in a relation are cardinality of a relation.
A domain of an attribute is a set of a permitted values for that attribute.

❖ Keys in Relational Model

- Keys in DBMS is an attribute or set of attributes which helps you to identify a row(tuple) in a relation(table). They allow you to find the relation between two tables.
- Keys help you uniquely identify a row in a table by a combination of one or more columns in that table. Key is also helpful for finding unique record or row from the table. Database key is also helpful for finding unique record or row from the table.

- **Super Key:**
- "A super key is a set of one or more attributes that allows to identify each tuple uniquely in a relation."
- For example, consider a relation Customer with attributes cid, cname, address and contact_no. In this relation, each tuple represents an individual customer. Here, the cid attribute can distinguish each tuple from another. So, cid is a super key for Customer relation.

- **Relation Key:**
- "A super key for which no subset is a super key is called a relation key."
- In other words, relation key is a minimal super key
- A relation key is sufficient to identify each and every tuple uniquely within a relation.
- A relation key cannot have null or duplicate values, as well as it does not contain any redundant attribute.
- Sometimes, there can be more than one relation keys for the same relation. For example, for Customer relation, {cid} as well as {cnam address} are relation keys.

- **Candidate Key:**
- "If a relation contains more than one relation keys, then, they each are called candidate key."
- For example, for Customer relation, {cid} as well as {cname, address} are relation keys. And so, they each are called candidate keys.

- **Primary Key:**
- "A primary key is a candidate key that is chosen by the database designer to identify tuples uniquely in a relation."
- For example, if database designers choose candidate key cid to identify all customers uniquely, then, cid is a primary key of the relation Customer.

- **Foreign Key:**
- "A foreign key is a set of one or more attributes whose values are derived from the key attributes of another relation."
- For example, consider the Account and Branch relations. Here, values for attribute *bname* of Account relation are derived from the Branch relation. So, an attribute *bname* in Account table is referred as foreign Key.

- **Alternate Key:**
- "An alternate key is a candidate key that is not chosen by the database designer to identify tuples uniquely in a relation."
- For example, if database designers choose candidate key cid to identify all customers uniquely, then, another candidate key {cname, address} is an alternate key.

❖ **Sub queries**

- A sub query is an SQL statement which appears inside another SQL statement.
- Here, a SELECT statement appears as a part of WHERE clause of some other SQL statement, and so, it is referred as a sub query.
- The statement containing a sub query is called a **parent** or **outer** query.

- The result of the parent query depends upon the result of the sub query.
- A sub query is also referred as a **nested query**.
- Consider following tables.

Customer:

cid	**ano**
C01	A01
C02	A02
C03	A03
C04	A04
C05	A05
C02	A04

Account_Holder:

ano	**balance**	**bname**
A01	5000	vvn
A02	6000	ksad
A03	7000	anand
A04	8000	ksad
A05	6000	vvn

Account:

cid	**name**
C01	Riya
C02	Jiya
C03	Diya
C04	Taral
C05	Saral

Example:

Find out the balance of an account which belongs to customer 'C01'.

Input:

```
SELECT BALANCE FROM ACCOUNT
WHERE ANO IN(
SELECT ANO FROM ACCOUNT_HOLDER
WHERE CID='C01'
);
```

Output:

```
BALANCE
----------
     5000
```

Example:

Find out the balance of accounts which belongs to customer 'Jiya'.

Input:

```
SELECT BALANCE FROM ACCOUNT
WHERE ANO IN(
SELECT ANO FROM ACCOUNT_HOLDER
WHERE CID IN(
SELECT CID FROM CUSTOMER
WHERE NAME='JIYA'
)
);
```

Output:

```
BALANCE
----------
     6000
     8000
```

Correlated Subquery:

- A sub query, which refers a column from a table in the parent query, is called a correlated sub query.
- A correlated sub query is evaluated once for each row processed by the parent statement.

Example:

Find out the accounts having maximum balance in branch to which it belongs. Display account number, balance and branch name for such accounts.

Input:

```
SELECT ANO, BALANCE, BNAME FROM ACCOUNT ACC
WHERE BALANCE IN (
SELECT MAX(BALANCE) FROM ACCOUNT
WHERE BNAME=ACC.BNAME
);
```

Output:

```
ANO   BALANCE    BNAME
---   ----------  ----------
A03    7000       ANAND
A04    8000       KSAD
A05    6000       VVN
```

❖ Relational Algebra

Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either **unary** or **binary**. They accept relations as their input and yield relations as their output. Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

The fundamental operations of relational algebra are as follows −

- **Select**
- **Project**
- **Union**
- **Set different**
- **Cartesian product**
- **Rename**

- **Select Operation (σ)**

It selects tuples that satisfy the given predicate from a relation.

Notation − $\sigma_p(r)$

Where **σ** stands for selection predicate and **r** stands for relation. *p* is prepositional logic formula which may use connectors like **and, or,** and **not**. These terms may use relational operators like − =, ≠, ≥, < , >, ≤.

For example −

$\sigma_{subject = "database"}(Books)$

Selects tuples from books where subject is 'database'.

$\sigma_{\text{subject = "database" and price = "450"}}(\text{Books})$

Selects tuples from books where subject is 'database' and 'price' is 450.

$\sigma_{\text{subject = "database" and price = "450" or year < "2022"}}(\text{Books})$

Selects tuples from books where subject is 'database' and 'price' is 450 or those books published before 2022.

- **Project Operation (∏)**

It projects column(s) that satisfy a given predicate.

Notation − $\prod_{A1}$, A_2, A_n (r)

Where A_1, A_2 , A_n are attribute names of relation **r**.

Duplicate rows are automatically eliminated, as relation is a set.

For example −

$\prod_{subject, author}$ (Books)

Selects and projects columns named as subject and author from the relation Books.

- **Union Operation (∪)**

It performs binary union between two given relations and is defined as −

r ∪ s = { t | t ∈ r or t ∈ s}

Notation − r U s

Where **r** and **s** are either database relations or relation result set (temporary relation).

For a union operation to be valid, the following conditions must hold −

- **r**, and **s** must have the same number of attributes.
- Attribute domains must be compatible.
- Duplicate tuples are automatically eliminated.

$\prod_{author}$ (Books) ∪ $\prod_{author}$ (Articles)

Projects the names of the authors who have either written a book or an article or both.

- **Set Difference (−)**

The result of set difference operation is tuples, which are present in one relation but are not in the second relation.

Notation − **r** − **s**

Finds all the tuples that are present in **r** but not in **s**.

$\prod_{author}$ (Books) − $\prod_{author}$ (Articles)

Provides the name of authors who have written books but not articles.

- **Cartesian Product (X)**

Combines information of two different relations into one.

Notation − r X s

Where **r** and **s** are relations and their output will be defined as −

r X s = { q t | q ∈ r and t ∈ s}

$\sigma_{author = 'tutorialspoint'}$(Books X Articles)

Yields a relation, which shows all the books and articles written by tutorialspoint.

- **Rename Operation (ρ)**

The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter **rho** ρ.

Notation − ρ_x (E)

Where the result of expression **E** is saved with name of **x**.

❖ Joins

A SQL Join statement is used to combine data or rows from two or more tables based on a common field between them. Different types of Joins are:

- Cross Join
- Inner Join
- Outer Join
- Self Join

❖ Cross Join (Cartesian Product, or Simple Join)

- The cross join combines every row from one table with every row in other table.
- The output of cross join is similar to Cartesian product, and so, this operation is also referred as Cartesian product.

Example:

Combine information from Account and Branch table.

Input:

```
SELECT * FROM ACCOUNT,BRANCH;
```

Output:

```
ANO   BALANCE    BNAME      BNAME      BADDRESS
---   ----------  ----------  -------    ------------------------------
A01     5000       VVN        VVN        Mota Bazar, VVNagar
A02     6000       KSAD       VVN        Mota Bazar, VVNagar
A03     7000       ANAND      VVN        Mota Bazar, VVNagar
A04     8000       KSAD       VVN        Mota Bazar, VVNagar
A05     6000       VVN        VVN        Mota Bazar, VVNagar
A01     5000       VVN        KSAD       Chhota Bazar, Karamsad
A02     6000       KSAD       KSAD       Chhota Bazar, Karamsad
A03     7000       ANAND      KSAD       Chhota Bazar, Karamsad
A04     8000       KSAD       KSAD       Chhota Bazar, Karamsad
A05     6000       VVN        KSAD       Chhota Bazar, Karamsad
A01     5000       VVN        ANAND      Nana Bazar, Anand
A02     6000       KSAD       ANAND      Nana Bazar, Anand
A03     7000       ANAND      ANAND      Nana Bazar, Anand
A04     8000       KSAD       ANAND      Nana Bazar, Anand
A05     6000       VVN        ANAND      Nana Bazar, Anand
```

❖ Inner Join (Equi and Non-equi Joins)

- The inner join combines only those records which contain common values in both tables.
- Thus, in contrast to cross join, inner join produces only consistent records in output.

Example:

Combine only consistent information from Account and Branch table.

Input:

```
select ANO, BALANCE, ACCOUNT.BNAME, BADDRESS from ACCOUNT,
BRANCH
WHERE ACCOUNT.BNAME = BRANCH.BNAME;
```

Output:

```
ANO   BALANCE    BNAME      BADDRESS
---   ---------- ---------- -----------------------------
A01    5000      VVN        Mota Bazar, VVNagar
A02    6000      KSAD       Chhota Bazar, Karamsad
A03    7000      ANAND      Nana Bazar, Anand
A04    8000      KSAD       Chhota Bazar, Karamsad
A05    6000      VVN        Mota Bazar, VVNagar
```

❖ Self Join:

- Sometimes, it is necessary to join a table with itself, as like, joining two separate tables. Such kind of join is called a self-join.
- Self-join is similar to inner-join, except that, a table is joined with itself. In self-join, each row of a table is combined with other rows of the same table to produce result.

Example:

Display employee id and name along with name of their manager.

Input:

```
SELECT EMP.EID, EMP.NAME "EMPNAME", MNGR.NAME "MNGRNAME"
FROM EMPLOYEE EMP, EMPLOYEE MNGR
WHERE EMP.MNGR_ID = MNGR.EID;
```

Output:

```
EID   EMPNAME   MNGRNAM
---   -------   -------
E05   SARAL     ZALAK
E04   TARAL     ZALAK
E03   FALAK     ZALAK
E01   PALAK     ZALAK
```

❖ Outer Join:

- The outer join operation is an extension of the inner join operation.

Example:

Combine the academic and hostel related information for all students.

Input:

```
SELECT ID, COLLEGE.NAME "NAME", DEPARTMENT, HOSTEL_NAME,
ROOM_NO FROM COLLEGE, HOSTEL
WHERE COLLEGE.NAME = HOSTEL.NAME;
```

Output:

```
ID  NAME      DEPARTMENT      HOSTEL_NAME        ROOM
--- --------  ----------      ------------------ ---
S02 Anisha    Computer        Kaveri Hostal      K01
S03 Nisha     I.T.            Godavari Hostal    G07
```

The outer join operation can be divided into three different forms:

1. Left outer join
2. Right outer join
3. Full outer join

1. Left Outer Join

- The left outer join retains all the records of the left table even though there is no matching record in the right table.

Example:

Input:

```
SELECT * FROM COLLEGE, HOSTEL
WHERE COLLEGE.NAME = HOSTEL.NAME(+);
```

Output:

```
NAME     ID  DEPARTMENT NAME    HOSTEL_NAME        ROOM
-------- --- ---------- ------- ------------------ ---
Anisha   S02 Computer   Anisha  Kaveri Hostal      K01
Nisha    S03 I.T.       Nisha   Godavari Hostal    G07
Manisha  S01 Computer
```

2. Right Outer Join

- The right outer join retains all the records of the right table even though there is no matching record in the left table.

Example:

Input:

```
SELECT * FROM COLLEGE, HOSTEL
WHERE COLLEGE.NAME(+) = HOSTEL.NAME;
```

Output:

```
NAME     ID  DEPARTMENT NAME    HOSTEL_NAME          ROOM
-------- --- ---------- ------- -------------------- ---
Anisha   S02 Computer   Anisha  Kaveri Hostal        K01
Nisha    S03 I.T.       Nisha   Godavari Hostal      G07
                        Isha    Kaveri Hostal        K02
```

3. Full Outer Join

- The full outer join retains all the records of both of the tables. It also pads null values whenever required.

Example:

Input:

```
SELECT * FROM COLLEGE, HOSTEL
WHERE COLLEGE.NAME = HOSTEL.NAME(+)
UNION
SELECT * FROM COLLEGE, HOSTEL
WHERE COLLEGE.NAME(+) = HOSTEL.NAME;
```

Output:

```
NAME      ID   DEPARTMENT NAME     HOSTEL_NAME          ROOM
--------  ---  ----------  -------   ------------------   ---
Anisha    S02  Computer    Anisha    Kaveri Hostal        K01
Manisha   S01  Computer
Nisha     S03  I.T.         Nisha    Godavari Hostal      G07
                            Isha     Kaveri Hostal        K02
```

❖ **Set operations**

- Consider following two tables Customer and Employee.

Customer:

cid	name
C01	Riya
C02	Jiya
C03	Diya
C04	Taral
C05	Saral

Employee:

eid	name	mngr_id
E01	Palak	E02
E02	Zalak	null
E03	Falak	E02
E04	Taral	E02
E05	Saral	E02

Union:

Query1 Union Query2

- The UNION clause merges the output of query1 and query2.
- The output will be as a single set of rows and columns.
- Output will contain all the records from query and query 2, but the common records will appear only once.

Example:

List out name of persons who are either customer or employee.

Input:

```
SELECT name FROM Customer
UNION
SELECT name FROM Employee;
```

Output:

```
Name
--------
Diya
Falak
Jiya
Palak
Riya
Saral
Taral
Zalak
```

Intersect:

query1 INTERSECT query2

- The INTERSECT clause combines the output of query1 and query2.
- Output will contain those records which are common to query1 and query2.

Example:

List out name of persons who are customer as well as employee.

Input:

```
SELECT name FROM Customer
INTERSECT
SELECT name FROM Employee;
```

Output:

```
NAME
------------
Saral
Taral
```

Minus:

query1 MINUS query2

- The MINUS clause combines the output of query1 and query2.
- Output will contain those records which are present in query1 but not present in query2.

Example:

List out name of persons who are customers but not employee.

Input:

```
SELECT name FROM Customer
MINUS
SELECT name FROM Employee;
```

Output:

```
NAME
-----------
Riya
Jiya
Diya
```

Unit-5

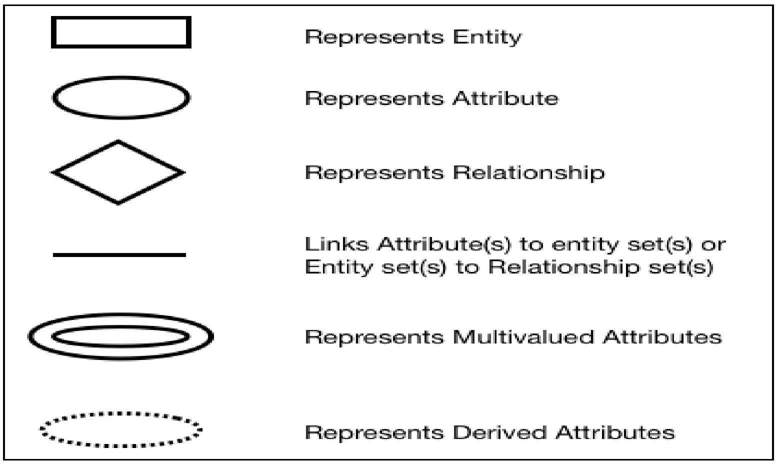
Entity Relationship Model

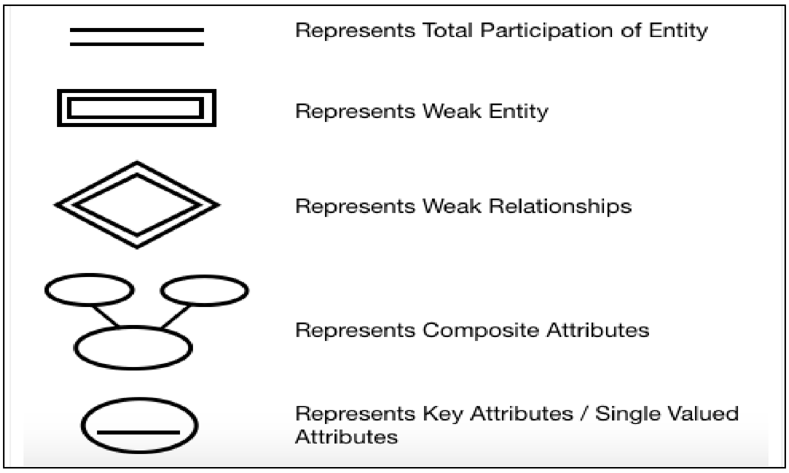
➢ Introduction

- ER model stands for an Entity-Relationship model. It is a high-level data model. This model is used to define the data elements and relationship for a specified system.
- Entity-relationship model describes data involves in real world in terms of object and their relationships.
- It is widely used for initial Database design.
- It describes overall structure of Database. E-R model is in fact, semantic data model which describes the meaning of data.
- It has a capability to map the meanings and interactions of real world objects on to the conceptual schema.

➢ Basic Terms of E-R Model

- **Entity**
  An entity may be any object, class, person or place. In the ER diagram, an entity can be represented as rectangles.
- **Attribute**
  The attribute is used to describe the property of an entity. Eclipse is used to represent an attribute.
- **Relationship**
  A relationship is used to describe the relation between entities. Diamond or rhombus is used to represent the relationship.


Represents Entity
Represents Attribute
Represents Relationship
Links Attribute(s) to entity set(s) or Entity set(s) to Relationship set(s)
Represents Multivalued Attributes
Represents Derived Attributes


Represents Total Participation of Entity
Represents Weak Entity
Represents Weak Relationships
Represents Composite Attributes
Represents Key Attributes / Single Valued Attributes

Types of Entity

- **Concrete Entity**
  A concrete entity is a physically existing entity, such as a customer, an employee, or a book.
- **Abstract Entity**
  An abstract entity has no physical existence, such as an account, a loan, or a subject.
- **Strong Entity**
- The strong entity has a primary key. Weak entities are dependent on strong entity. Its existence is not dependent on any other entity.
- **Weak Entity**
  The weak entity in DBMS do not have a primary key and are dependent on the parent entity. It mainly depends on other entities.

Types of Attributes

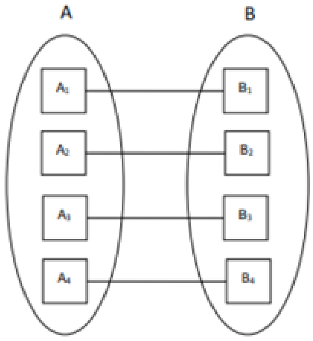
- **Single Attribute**
  Single valued attributes have single data value for a particular entity. For example: SSN or Gender.
- **Multi-valued Attribute**
  Multi-valued attributes have multiple data values for a particular entity. For example: Contact_No.
- **Composite Attribute**
  Composite attributes can be divided into smaller sub-parts.
- **Stored Attribute**
  Attribute that cannot be derived from other attributes are called as stored attributes. For example: Date of Birth.
- **Derived Attribute**
  The value for derived attribute can be derived from the values of the other related attributes.
- **Key Attribute**
  The value of attribute that can be uniquely identify i.e known as Key Attribute. For example: License no. of vehicle.

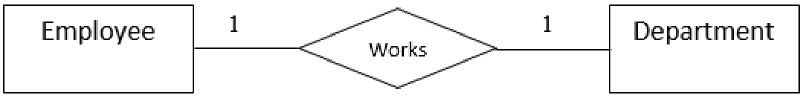
Cardinality

- A Cardinality is a data constraint that expresses the number of entities to which another entity can be related via a relationship set.
- It is most useful in describing the relationship sets that involve more than two entity sets.
- For binary relationship set R on an entity set A and B, there are four possible mapping cardinalities.
- Cardinality (cardinality constraints)
- It represents the number of entities of another entity set which are connected to an entity using a relationship set.
- It is most useful in describing binary relationship sets.
- For a binary relationship set the Cardinality must be one of the following types:
- **One to One**
- **One to Many**
- **Many to One**
- **Many to Many**

One to One

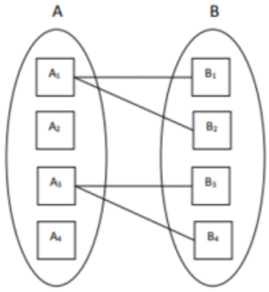
An entity in A is associated with at most (only) one entity in B and an entity in B is associated with at most (only) one entity in A.

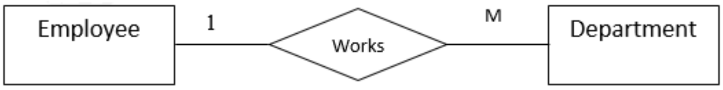




One to many

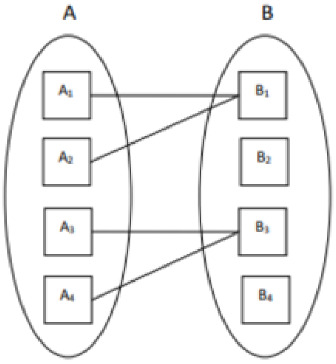
An entity in A is associated with any number (zero or more) of entities in B and an entity in B is associated with at most (only) one entity in A.

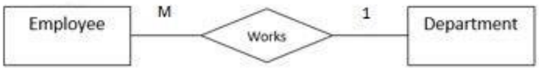




➢ Many to One

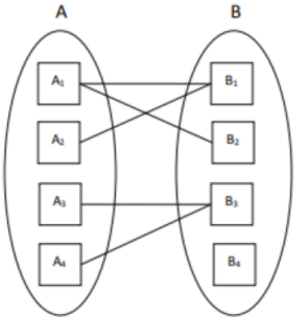
An entity in A is associated with at most (only) one entity in B and an entity in B is associated with any number (zero or more) of entities in A.

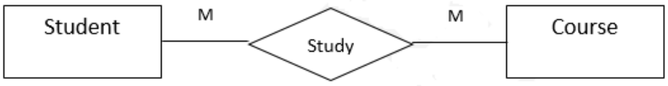




➢ Many to Many

An entity in A is associated with any number (zero or more) of entities in B and an entity in B is associated with any number (zero or more) of entities in A.





➢ Design E-R Model

- Here are some best practice or example for Developing Effective E-R Diagrams.
- Eliminate any redundant entities or relationships.
- You need to make sure that all your entities and relationships are properly labelled.
- There may be various valid approaches to an ER diagram. You need to make sure that the ER diagram supports all the data you need to store.
- You should assure that each entity only appears a single time in the E-R Diagram.
- Name every relationship, entity, and attribute are represented on your diagram.
- Never connect relationships to each other.
- You should use colors to highlight important portions of the E-R Diagram.

➢ Problem E-R Model

- The E-R model can result problems due to limitations in the way the entities are related in the relational Databases. These problems are called connection traps. These problems often occur due to a misinterpretation of the meaning of certain relationships.
- Two main types of connection traps are called fan traps and chasm traps.
- **Fan trap**
  Fan trap occurs when two relationship sets, having Cardinality one to many, converge to single entity set.
- **Chasm trap**
  Chasm trap occurs when an E-R Diagram suggests the existence of a relationship between entity sets, but in real it doesn't exist.

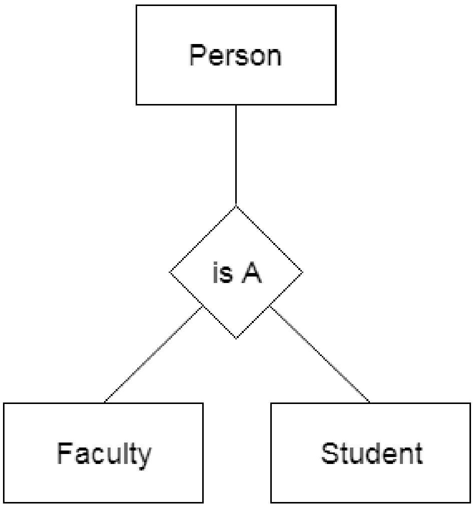
➢ Basics of Enhanced E-R Model

- EER is a high-level data model that incorporates the extensions to the original ER model. It is a diagrammatic technique for displaying the following concepts:
- Sub Class and Super Class
- Specialization and Generalization
- **Super Class:**
  "A super-class is a generic entity set, which has relationship with one or more sub-classes."
- **Sub Class:**
  "A Sub Class is a subset of entities in an entity set, which has attributes distinct from those in other subsets."

➢ Generalization

- Generalization is like a bottom-up approach in which two or more entities of lower level combine to form a higher level entity if they have some attributes in common.
- In generalization, an entity of a higher level can also combine with the entities of the lower level to form a further higher level entity.
- Generalization is more like subclass and superclass system, but the only difference is the approach. Generalization uses the bottom-up approach.
- In generalization, entities are combined to form a more generalized entity, i.e., subclasses are combined to make a superclass.

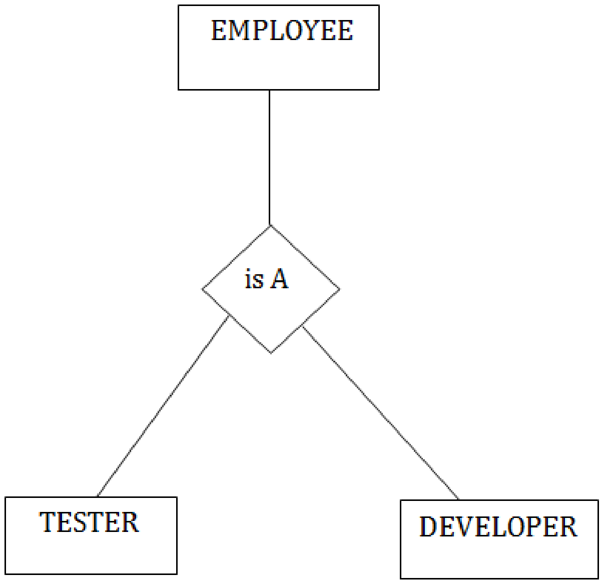
- **For example,** Faculty and Student entities can be generalized and create a higher level entity Person.



➢ Specialization

- Specialization is a top-down approach, and it is opposite to Generalization. In specialization, one higher level entity can be broken down into two lower level entities.
- Specialization is used to identify the subset of an entity set that shares some distinguishing characteristics.
- Normally, the superclass is defined first, the subclass and its related attributes are defined next, and relationship set are then added.

- **For example:** In an Employee management system, EMPLOYEE entity can be specialized as TESTER or DEVELOPER based on what role they play in the company.


EMPLOYEE
is A
TESTER
DEVELOPER