

ABOUT THE SQL TUTORIAL

SQL Tutorial:

SQL tutorial provides basic and advanced concepts of SQL. Our SQL tutorial is designed for both beginners and professionals. Our SQL tutorial helps you learn SQL (Structured Query Language) in simple and easy steps so that you can start your database programming quickly. It covers most of the important concepts related to SQL for a basic to advanced understanding of SQL and to get a feel of how SQL works. This tutorial will give you quick start with SQL.

Audience:

This reference has been prepared for the beginners to help them understand the basic to advanced concepts related to SQL languages. No prior DB experience is required. It is good to have SQL Server installed on your computer, as it might assist you in executing the examples yourself and get to know how it works.

Prerequisites:

Before you start doing practice with various types of examples given in this reference, I'm making an assumption that you are already ware about what is database, especially RDBMS.

Note:

SQL is not a database system, but it is a query language. Suppose you want to perform the queries of SQL language on the stored data in the database. You are required to install any database management system in your systems, for example, Oracle, MySQL, MongoDB, PostgreSQL, SQL Server, DB2, etc.

SQL - COURSE INDEX PAGE

S.NO	Module #	Module Name	Page #
1	Module-1	Introduction to Basic Database Concepts	7-17
2	Module-2	E-R Modelling and Normalization	18-47
3	Module-3	Introduction to SQL [Create Database using ORACLE & SQL Commands]	48-65
4	Module-4	INTEGRITY CONSTRAINTS SQL Constraints SQL Data Integrity	66-69 70-73
5	Module-5	SQL DUAL TABLE SQL Operators SQL Clauses	74-75 76-85 86-99
6	Module-6	SQL Functions	100-118
7	Module-7	SQL Views SQL Indexes	119-128 129-132
8	Module-8	SQL Sub Queries SQL Joins	133-136 137-146
9	Module-9	SQL Sequences SQL Synonym	147-149 150-152
10	Module-10	Table Alias Triggers Flashback and Purge Commands	153-155 156-156 157-160
11		SQL Interview Questions	161-205

SQL [ORACLE-DB] -CONTENTS

Module 1: Introduction to Basic Database Concepts

- Data, Information, Table, Field, Record.
- Database and database Types.
- Introduction to DBMS and RDBMS.
- Limitations of File Management System.
- Advantages of Data Base Approach.
- Understanding Client and Server Architecture.
- Database Administrator (DBA)
- Exploring Relational DBMS

Module 2: E-R Modelling and Normalization

- Attribute and Types of Attributes
- Entity and Types of Entity's
- Anomaly and Types of Anomalies
- Relationship and Types of Relationships
- Degree of Relationships and Types
- Key and Types of Keys
- ER-MODELS and Basic Symbols
- Conversion of E-R Diagrams into Tables
- Dependency and Types of Database Dependencies

Normalization

- First Normal Form
- Second Normal Form
- Third Normal Form
- Boyce-Codd Normal Form (BCNF)
- Fourth Normal Form Practically Normalizing Tables

Module 3: Introduction to SQL

- What is SQL? Why SQL? History of SQL.
- How to Install and Run SQL[ORACLE]
- Datatypes in SQL

SQL Commands:

- DDL - Data Definition Language
- DML - Data Manipulation Language

- DCL - Data Control Language
- DQL - Data Query Language
- TCL - Transaction Control Language

Create Database using ORACLE

- SQL CREATE TABLE Statement
- SQL ALTER TABLE Statement
- SQL INSERT INTO Statement
- SQL UPDATE Statement
- SQL SELECT Statement
- Datatypes in SQL
- Creating Users & Roles
- Granting & Revoking of Roles & privileges

Module 4: INTEGRITY CONSTRAINTS

SQL Constraints

- NOT NULL Constraint
- DEFAULT Constraint
- Drop Default Constraint
- UNIQUE Constraint
- DROP a UNIQUE Constraint
- Primary Key
- Default Check Foreign Key

SQL Data Integrity

- Entity integrity
- Domain integrity
- Referential integrity
- Types of constraints
- Data Integrity

Module 5: Implementation of SQL Clauses and Data integrity

SQL Clauses

- SQL DISTINCT Clause
- SQL WHERE Clause
- SQL AND/OR Clause
- SQL IN Clause
- SQL BETWEEN Clause
- SQL Like Clause
- SQL ORDER BY Clause
- SQL GROUP BY Clause
- SQL COUNT Clause

- SQL HAVING Clause

Data integrity

- Entity integrity
- Domain integrity
- Referential integrity
- Types of constraints
- Data Integrity

Module 6: SQL Functions

- **Aggregate Functions**
 - AVG SUM MAX MIN COUNT
- **Scalar Functions**
 - Character functions
 - Number (Numeric) functions
 - Data functions
- **Conversion functions**
 - TO_CHAR
 - TO_NUMBER
 - TO_DATE

Module 7: Implementing SQL Views and SQL Indexes

- Introduction & Advantages of Views
- Creating, Altering, Dropping Views
- Inserting Rows into a View
- Deleting Rows into a View
- Dropping Views
- Advance Options while Creating a View
- SQL Views With CHECK OPTION

SQL Indexes

- The Create Index
- Single-Column Indexes
- Unique Indexes
- Composite Indexes
- Implicit Indexes
- Drop Indexes

Module 8: Sub Queries and Joins

SQL Sub Queries

- Single Row Sub Queries
- Multi Row Sub Queries
- Subqueries with the SELECT Statement
- Subqueries with the INSERT Statement
- Subqueries with the UPDATE Statement
- Subqueries with the DELETE Statement

Introduction to Joins and SQL Join Types

- Inner Join
- Outer Join
- Self - Join
- Left JOIN
- Right Join
- Full Join
- Cross Joins
- Set Operations using Unions, Intersect and Except

Module 9: Implementing Sequences and Synonyms

- SQL – Using Sequences
- Using AUTO_INCREMENT column
- Obtain AUTO_INCREMENT Values
- Starting a Sequence at a Particular Value

Module 10: Table Alias, Triggers and Flashback and Purge Commands

- Table Alias
- Flashback
- Purge Commands

Triggers

- Introduction to triggers
- Constraints vs Triggers

Module 1: INTRODUCTION TO BASIC DATABASE CONCEPTS

Data:

Data is a collection of characters (or) raw material. Data consists of facts, text, image, sound graphical & video segments that have meaning in the user environment. In the context of computing, data refers to digital information that can be stored, processed, and analysed using computers and other digital devices. Data can be structured or unstructured, and can be generated from various sources, including sensors, social media, transactions, and surveys. It can be used for a wide range of purposes, including research, decision-making, analysis, and prediction. With the rise of big data, data has become an increasingly valuable resource for businesses, governments, and individuals alike. The term data referred to known that it would be recorded & stored on computer media.

The term 'Data' refers to collection of raw facts. A collection of characters, which doesn't have any meaning is called as Data.

Data is represented with the help of characters such as alphabets (A-Z, a-z), digits (0-9) or special characters (+,-,/,*,<,>,= etc.)

For Example: Student data: Ramu, Java, Software etc., **Employee data:** Bharat, Accounts Department, 25000/-

Information:

Processed data is known as information (or) data that have been processed in such a way as to increase the knowledge of the person who user the data. Information is data that has been processed in such a way as to be meaningful to the person who receives it. it is anything that is communicated.

Information is a collection of processed or organized data that has meaning and relevance to a specific context, purpose, or audience. It is the result of processing, analyzing, or interpreting data in a way that can be understood and used by people or machines. Information can take many forms, such as text, images, sounds, or videos, and can be communicated through various channels, such as books, websites, social media, or email. It can be used for various purposes, such as decision-making, learning, communication, or entertainment. In order for data to become information, it needs to be processed in a way that adds value and makes it

useful for a specific context or purpose. This may involve filtering, sorting, analyzing, or summarizing the data, or presenting it in a way that is easy to understand and interpret.

Student ID	First Name	Last Name	Date of Birth	Gender	GPA
1001	John	Smith	01/01/2002	Male	3.5
1002	Jane	Doe	05/12/2001	Female	3.9
1003	Mark	Johnson	10/23/2002	Male	3.2
1004	Emily	Davis	03/14/2003	Female	4.0
1005	David	Brown	07/09/2001	Male	2.8

Difference between Data and Information?

DATA: - Data means raw facts. Data are the unprocessed information .it has the following characters:

- a) Data are not useful for decision making
- b) Data are collected from various sources
- c) Data may be collection of characters or a single character.

Alphabetic (a-z or A-Z) or Alpha Numeric (0,1 9 and A(a) Z(z)) items.

For example: - students individual names. i.e., Ramu Raj.....

Information: - Information is the processed data, it has the following characteristics

- (a). It is useful in decision making.
- (b). Information is the output of data

Table:

Table is a collection of data, organized in terms of rows and columns. Tables are database objects that contain all the data in a database. In tables, data is logically organized in a row-and-column format similar to a spreadsheet [Microsoft Excel]. Table is the simple form of data storage. A table is also considered as a convenient representation of relations. In SQL, a table is a database object that stores data in rows and columns. Each row represents a unique record, and each column represents a field in the record. It is the fundamental component of a relational database, and it consists of one or more columns, each of which has a data type, and zero or more rows of data. In SQL, tables are created using the CREATE TABLE statement, which specifies the name of the table, the columns it contains, and their data types. In SQL databases a table is also known as a relation.

Example: The below table named "customers" with four columns: "id", "name", "age" and "email".

ID	Name	Age	Email
0148	Krishna	30	Krishna123@gmail.com
5627	Madhu	25	Madhu@gmail.com
8372	Suresh	40	Suresh@gmail.com

Fields (Columns):

Tables contain rows and columns, where the columns are known as fields. In database terminology, field is often used to refer to the individual cells within a row or column. However, it can also refer to the whole column itself. A field is a column in a table that is designed to maintain specific information about every record in the table. A record, also called a row, is each individual entry that exists in a table

A column/field is a set of data values of a particular type (like numbers or alphabets), one value for each row of the database,

For example: Age, Student_ID, or Student_Name.

Record:

In a database, a record (sometimes called a row) is a group of fields within a table that are relevant to a specific entity. A record is also known as a tuple. Tables contain rows and columns, where the rows are known as records and the columns are known as fields. A record in a database refers to a single instance or row of data that contains a set of related fields or attributes. Each record typically represents a unique entity or object within the database, such as a customer, an order, or a product.

The data within a record is organized into fields, which represent specific attributes or characteristics of the entity being stored. For example, a customer record might contain fields for the customer's name, address, phone number, and email address.

In a relational database, records are typically organized into tables, with each table containing multiple records. Each record in a table is identified by a unique value in a field known as the primary key. This key is used to link related records across different tables in the database.

For example, a customer record may include items, such as first name, physical address, email address, date of birth and gender.

NULL value:

A NULL value in a table is a value in a field that appears to be blank, which means a field with a NULL value is a field with no value. It is very important to understand that a NULL value is different than a zero value or a field that contains spaces. A field with a NULL value is the one that has been left blank during a record creation.

Data Base: Organized collection of logically related data is known as “Data base”.

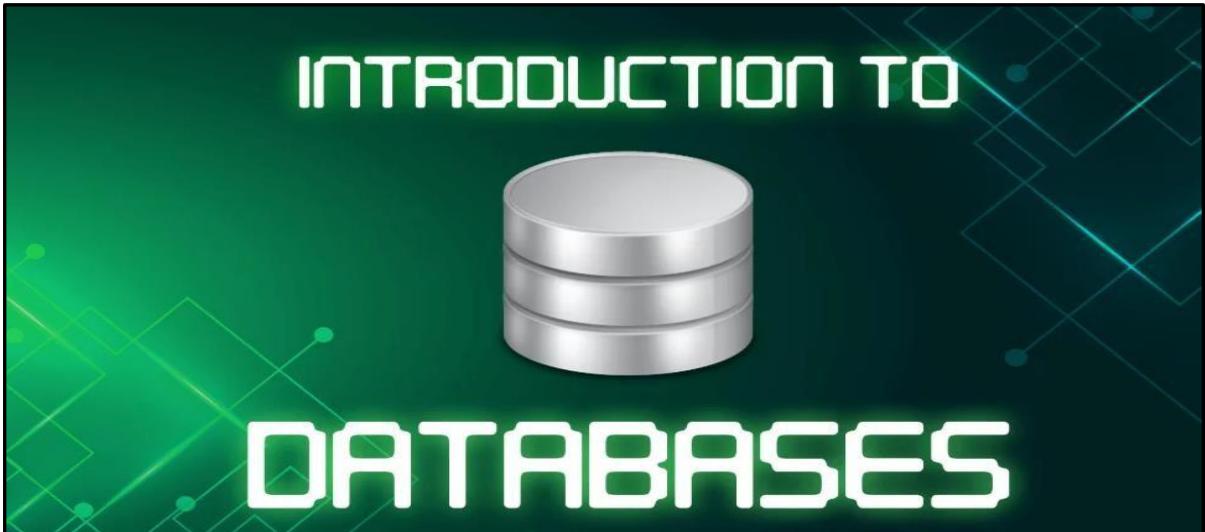
Or

A database is an organized collection of structured information, or data, typically stored electronically in a computer system. A database is usually controlled by a database [11]

management system (DBMS). A ‘database’ is a collection of data items. Database is also called as a centralized location i.e., “Repository” Database is a collection of interrelated files-and data, stored in a standard format and shared by multiple users. Or The arranging of related information in a systematic manner is called ‘database’.

Example:

Oracle, MS SQL Server, MySQL, MongoDB, Cassandra, Redis, etc.



Types of Databases:

Depending upon the usage requirements, there are following types of databases available in the market.

1. Centralized Database:

A centralized database (CDB) is a database that is located, stored, and maintained in a single location. This location is most often a central computer or database system, for example a desktop or server CPU, or a mainframe computer.

2. Distributed Database:

A distributed database is a collection of multiple interconnected databases, which are spread physically across various locations that communicate via a computer network.

3. Personal Database:

Personal database system is the local database system which is only for one user to store and manage the data and information on their own personal system. Personal database management system requires only one application to store and manage data in personal computer.

4. End-user Database:

An end user database is simply a software which helps store data created by an end user.

5. Commercial Database:

A commercial database is one created for commercial purposes only and it's available at a price. Unlike open- source databases, commercial databases can only be viewed or modified by authorized users.

6. Operational Database:

The Operational Database is the source of information for the data warehouse. It includes detailed information used to run the day-to-day operations of the business. The data frequently changes as updates are made and reflect the current value of the last transactions.

Operational Database Management Systems also called as OLTP (Online Transactions Processing Databases), are used to manage dynamic data in real-time.

7. Relational Database:

A relational database is a type of database that focuses on the relation between stored data elements. It allows users to establish links between different sets of data within the database and use these links to manage and reference related data.

Many relational databases use SQL (Structured Query Language) to perform queries and maintain data.

8. Cloud Database:

A cloud database is a database that is built, deployed, and accessed in a cloud environment, such as a private, public, or hybrid cloud.

9. Object-Oriented Database.

An object-oriented database (OODBMS) or object database management system (ODBMS) is a database that is based on object-oriented programming (OOP). The data is represented and stored in the form of objects.

10. Graph Database:

A graph database is defined as a specialized, single-purpose platform for creating and manipulating graphs. Graphs contain nodes, edges, and properties, all of which are used to represent and store data in a way that relational databases are not equipped to do.

DBMS:

DBMS is referred as Data Base Management System. To maintain database in computers, it needs a software called as DBMS.

A DBMS is a collection of interrelated data and a set of programs to access those data.

Or

DBMS is a software designed for maintaining and using large collections of data.

RDBMS:

RDBMS stands for Relational Database Management System. It is a software system designed to manage relational databases, which are a type of database that organizes data into one or more tables or relations, with each table consisting of rows and columns. A Relational Database Management System (RDBMS) is a software system that is used to create, manage, and maintain relational databases. Relational databases organize data into tables with rows and columns, where each table represents a unique entity or relationship between entities.

One of the key features of an RDBMS is its ability to enforce relationships between tables through the use of primary and foreign keys. A primary key is a unique identifier for each record in a table, while a foreign key is a field in one table that refers to the primary key in another table. This allows for the creation of complex data models that can represent real-world relationships between different entities. RDBMSs also provide a variety of security and access control features to ensure that only authorized users can access and modify data. They typically support transaction processing, which ensures that multiple changes to the database are executed as a single unit of work, and provide backup and recovery capabilities to protect against data loss.

Examples of popular RDBMSs include Oracle, Microsoft SQL Server, MySQL, and PostgreSQL.

By using an RDBMS to manage your customer database, you can easily add, update, and retrieve information about your customers and their orders. RDBMSs also provide features for ensuring data consistency, security, and reliability. Examples of RDBMSs include MySQL, Oracle, Microsoft SQL Server, and PostgreSQL.

Disadvantages of traditional file processing system?

(or)

Limitations of file-based system?

(or)

Limitations of manual processing System?

The computer systems are used for business applications to store and retrieve the data. To perform these operations, a large file processing system is needed. We maintain files like paper or books for data in olden days. As business applications became complicate the file processing system unable to meet the business requirements. So, for this purpose database has been developed.

The following are the disadvantages of file processing system:

- i. Program – Data dependence,
- ii. Redundancy of data,
- iii. Limited sharing data,
- iv. Lengthy program development time.
- v. Excessive program maintenance,
- vi. Data isolation
- vii. Security problems

i. Program-Data Dependence:

The file descriptions are stored within each application program. For example, in the above diagram, accounts department having two files namely customer master file and inventory pricing file and two application programs (Program-A, Program-B,) . To access these files, the corresponding changes must be made in application programs

ii. Redundancy of data (Duplication of data):

Since, application programs are often developed independently in file processing systems, the duplication of data is existing as a rule.

For example, the accounting department having two files namely customer master file and inventory pricing file, and orders department also having same these two files. So, we can say that duplication is existing in file processing system.

This duplication of data leads to wastage of memory, inconsistency and integrity problems

iii. Limited sharing data:

With traditional file processing system, each application has its own private file and then the departments cannot use other departments data.

For example, orders department cannot access account department files.

iv. Lengthy development time:

With traditional file processing system, there are little opportunities to use previous development efforts. The application programs are needed to develop from the beginning.

So, it takes lengthy time in developing application programs.

v. Excessive Program Maintenance:

The above all factors combined together creates a heavy program maintenance.

vi. Data isolation:

The data are scattered in various files, and files may be in different formats. So, to retrieve the appropriate data is difficult and difficult to write new program.

vii. Security problems:

Not every user of the data base system should be able to access all the data. They should access only required data and remaining data is hided. But, in file processing system, it is difficult to keep security rules.

Advantages of DataBase approach:

(or)

Advantages over Traditional file processing system:

Data base approach integrates and shares the data throughout the organization.

Advantages of Data base approach:

- i. Program data independence,
 - ii. Minimal data redundancy,
 - iii. Data administration,
 - iv. Data consistency,
 - v. Reduced application development time,
 - vi. Enforcement of standards,
 - vii. Data integrity and security problems,
 - viii. Improved data quality,
 - ix. Improved data access & good response,
 - x. Improved data sharing.
-
- i. **Program data independence:** The Separation of data description from application programs is called data independence. In data base approach, data descriptions are stored in a centralized location called “Repository” “Applications programs are independent as possible.
 - ii. **Minimal data redundancy:** The design goal with the data base is integrating and storing in one logical structure i.e., separate files are stored in one structure. Data base approach does not avoid redundancy completely. But it minimizes (reduces) Some

- redundancy.
- iii. **Data administration:** When several users share the data, then centralizing the administration of data can offer more improvements. Experienced professionals, who understand the nature of the data being managed.
 - iv. **Data consistency:** By eliminating or controlling redundancy we reduce data inconsistency. So, we can decrease wastage of memory space.
 - v. **Reduced application development time:** Stored data may need to change for some reason. For example, if a customer address is changed that must be updated with new address. This can be done with data base management system facilities very easily. And maintenance time of a program will be very less compared to traditional file processing system.
 - vi. **Enforcement of standards:** Data base Administrator (DBA) should enforce the standards. The standards may include adding and deleting records, some access of data etc., These standards must be followed by each and every user of the data base.
 - vii. **Improved data Quality:** With data base approach, these are two methods to improve the quality of data.
 - The data designer will specify some rules regarding the data.
 - Date warehouse requires cleaning up the data.
 - viii. **Data integrity and security:** If the data is always accessed through the DBMS, the DBMS can enforce integrity rules.
 - ix. **Improved data access and good response:** With data base approach, the DBMS allows the user to access the data without any knowledge of programming languages.
 - x. **Improved data sharing:** The data base system is an integrated centralized database, so, same file can be used in different applications.

Client-Server Architecture in DBMS:

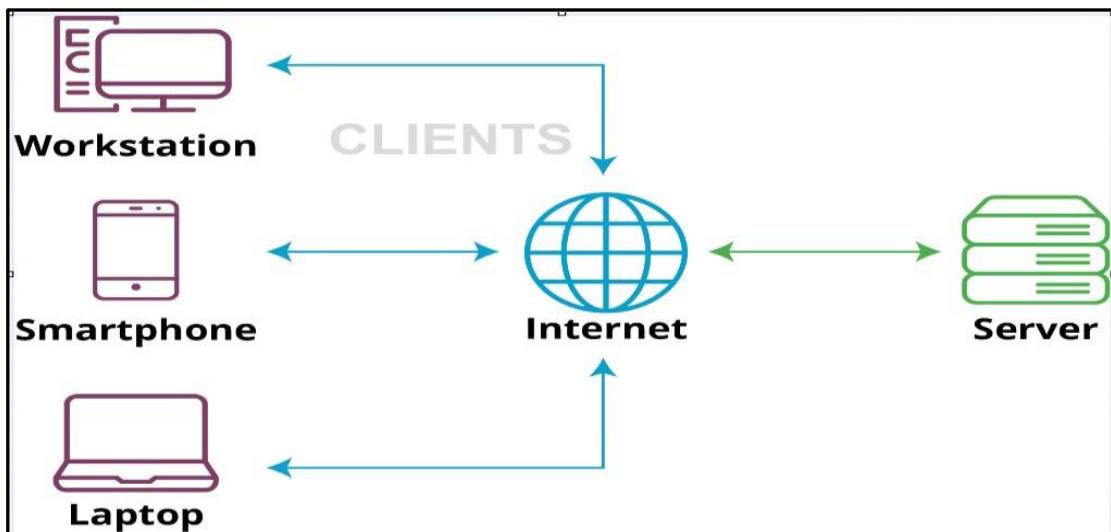
In client-server architecture many clients connected with one server. The server is centerlines.it provides services to all clients. All clients request to the server for different Service. The server displays the results according to the client's request. Client/server architecture is a computing model in which the server hosts (computer), send and manages most of the resources and works to be required by the client. In this type of architecture has one or more client computers attached to a central server over a network. This system shares different resources. Client/server architecture is also called as a networking computing model and client- server network because all the requests and demands are sent over a network.

Client: Where the request initiated. A piece of software or application that takes the input and sends request to the servers. Clients are the ones who request services.

Server: Where SQL Services got installed and databases reside. A piece of software that receives and processes requests from clients. A server is the one who provides requested

services.

“The main function of client-server architecture is as a storage system for data. In this approach, all data and applications on storage devices are kept stored on the remote server. Whenever a client requires getting access a specific file or application then it sends a request to server.”



Working of Client-server Architecture in DBMS:

Basically, client-server model defines how the server provides services to clients. Server is a centralized computer that provides services to all attached clients. For example, file server, web server, etc. Each the basic work of server to provide services to each client. The client can be a laptop computer, tablets, and smartphones, etc. The server has many types of relationships with clients. Many servers have one too many relationships with clients. In one too many relationships many clients connected with one server. When one client wants to communicate with the server. The server may accept or reject the request of clients. When the server computer accepts the request of clients then server maintains a connection according to a defined protocol. **Another example of a client-server model is online gaming.**

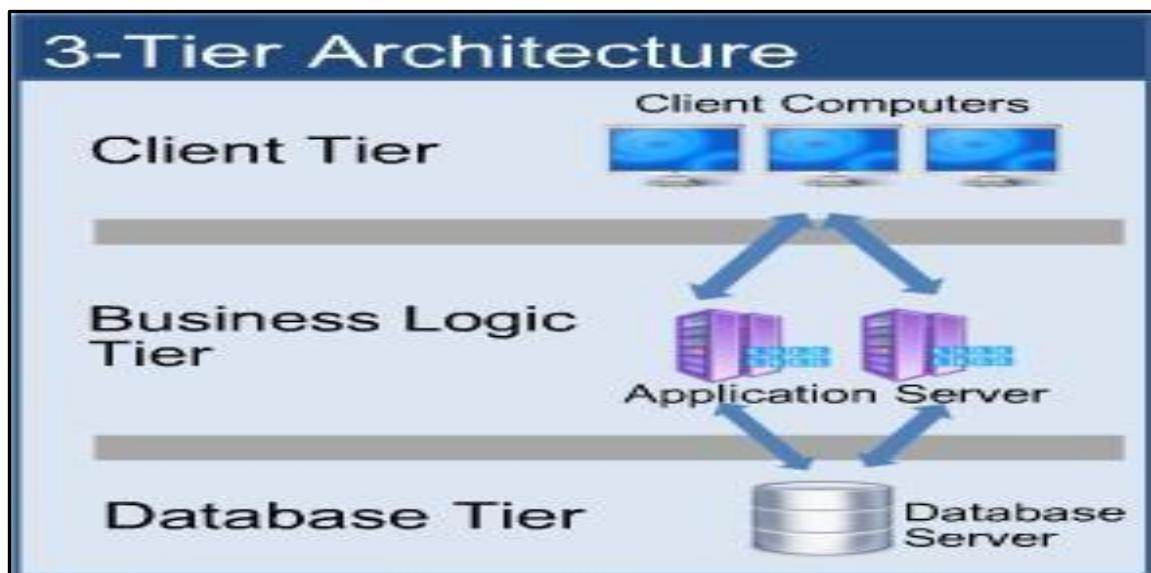
Structure of Client-server Database Architecture in DBMS:

By using this architecture structure this software is divided into three different tiers:

1. **Presentation Tier:** This is the first and topmost level of the application. The basic work of this layer provides user interface. The interface is a graphical user interface. The

graphical user interface is an interface that consists of menus, buttons, and icons, etc. The presentation tier presents information related to such work as browsing, sales purchasing, and shopping cart contents. It attaches with other tiers by computing results to the browser/client tier and all other tiers in the network.

2. **Logic Tier:** The logical tier is also known as data access tier and middle tier. It lies between the presentation tier and the data tier. It basically controls the application's functions by performing processing. The components that build this layer exist on the server, assist the resources sharing these components also define the business rules like different government legal rules, data rules, and different business algorithm. which are designed to keep data structure consistent.
3. **Data Tier:** This is basically the DBMS (database management system) layer. This layer consists of database. It can be used through the business services layer. In this layer, data is stored and retrieved. The responsibility of this layer to keep data consistent and independent.



Advantages of Client-server Architecture in DBMS:

- All the data and resources are controlled by server in this way all data and resources are very consistent.
- You can easily increase the number of clients in this architecture at any time. This all increases the scalability of the network.
- This is very easy to maintain you can easily repair, replace or add clients in this network.
- This network is very easy to use and it is not complicated.

Disadvantages of Client-server Architecture in DBMS:

- Traffic is a big problem in this network.
- When you add large numbers of the client with server this network will be more complicated.
- When the server goes down all the clients are not able to send their request. The whole work will be stopped
- The hardware and software are very expensive.
- The client does not have resources for each resource they need to request the server. Because of all resources exist on server

Database Administrator (DBA):

A Database Administrator (DBA) is an individual or person responsible for controlling, maintaining, coordinating, and operating a database management system. Managing, securing, and taking care of the database systems is a prime responsibility. They are responsible and in charge of authorizing access to the database, coordinating, capacity, planning, installation, and monitoring uses, and acquiring and gathering software and hardware resources as and when needed. Their role also varies from configuration, database design, migration, security, troubleshooting, backup, and data recovery. Database administration is a major and key function in any firm or organization that is relying on one or more databases.

A DBA, or Database Administrator, is a professional responsible for designing, implementing, and maintaining an organization's databases. DBAs may also be responsible for developing and implementing database policies and procedures, monitoring and tuning database performance, and working with developers to optimize database design and performance.

DBAs typically work with a variety of database management systems, such as Oracle, Microsoft SQL Server, MySQL, among others. They may also work with different database technologies such as cloud-based database solutions.



DATABASE ADMINISTRATOR ROLES AND RESPONSIBILITIES:

Database Design:

DBAs are responsible for designing, creating, and maintaining the organization's databases. They work closely with developers and stakeholders to ensure that the database meets the organization's needs.

Database Accessibility:

Setting up employee access is a critical component of database security. DBAs decide who has access and what kind of access they have. They create a subschema to control database accessibility. They also determine which users will have access to the database and which users will use data. Without the permission of the DBA, no user has the authority to access the database.

Security:

DBAs are responsible for maintaining the security of the database. This includes setting up user accounts and roles, assigning privileges, and monitoring for potential security breaches.

Takes Care of Data Extraction, Transformation, and Loading:

DBAs are responsible for Data extraction, transformation, and loading, also known as (ETL), which refers to the efficient import of large amount of data extracted from multiple systems into a data warehouse environment. The external data is cleaned and transformed to fit the required format before being imported into a central repository.

Backup and Recovery:

DBAs are responsible for setting up and managing backup and recovery procedures to ensure that data can be restored in the event of a disaster or system failure.

Performance Tuning:

DBAs are responsible for monitoring the performance of the database and optimizing it for maximum efficiency. This includes identifying and resolving performance issues, tuning queries, and managing resources.

Data Integration:

DBAs are responsible for integrating data from different sources and ensuring that the data is accurate and consistent across the organization.

Database Upgrades and Migration: DBAs are responsible for upgrading the database software and migrating data to new systems when necessary.

Disaster Recovery:

DBAs are responsible for developing and implementing disaster recovery plans to ensure that

the organization can recover from a catastrophic event or system failure.

Software Installation and Maintenance:

A DBA is frequently involved in the initial installation and configuration of a new Oracle, SQL Server, or other databases. The system administrator configures the database server's hardware and implements the operating system, after which the DBA installs and configures the database software. The DBA is in charge of ongoing maintenance, such as updates and patches. In addition, if a new server is implemented, the DBA is in charge of transferring data from the existing system to the new platform.

Documentation:

DBAs are responsible for documenting all aspects of the database, including database design, security, backup and recovery procedures, and performance tuning.

Technical Support:

DBAs are responsible for providing technical support to users of the database, including developers, management, and end-users.

Provides Support to Users: If a user requires assistance at any time, it is the DBA's responsibility to assist him. The DBA provides complete support to users who are new to the database.

Module-2: E-R MODELING and NORMALIZATION ATTRIBUTES and TYPES

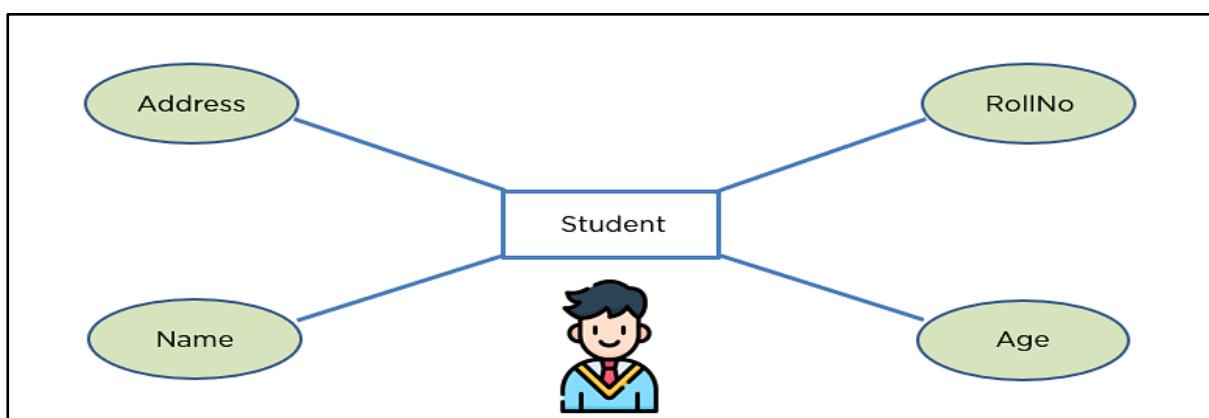
Attribute:

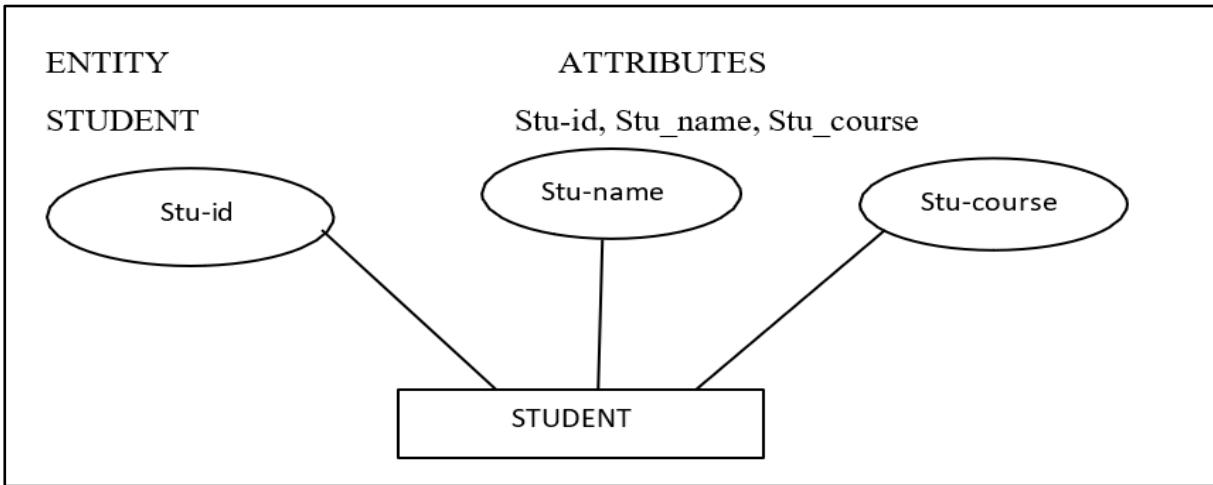
An attribute is a property or characteristic of each entity set. It represents the field or column of a table. An attribute is represented by a symbol ellipse. ‘  ‘. An attribute exhibits the properties of an entity. You can illustrate an attribute with an oval shape in an ER diagram.

Rules for declaring attribute:

- i. Each attribute should be given meaningful name and starts with an alphabet.
- ii. Initial letter of each word in an attribute should be in upper case.
- iii. An attribute should not have any blank spaces between the characters.
- iv. Each word of an attribute should be separated by under base.
- v. In a diagram, each attribute should be connected to an entity type with a line.

For example, In STUDENT entity, it contains stu_Id, stu_name, stu_course attribute.





In the above example, STUDENT is an entity type contains properties i.e, attribute the should, Stu-id, Stu-name, Stu-course.

Types of Attributes with diagrams:

1. Simple (or) Atomic Attribute,

2. Composite Attribute

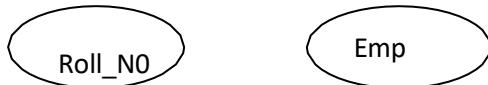
3. Single Valued Attribute,

4. Multi Valued Attribute,

5. Derived Attribute.

1. **Simple (or) Atomic Attribute:** An attribute that cannot be broken down into smaller components is called “Simple” or atomic attribute.

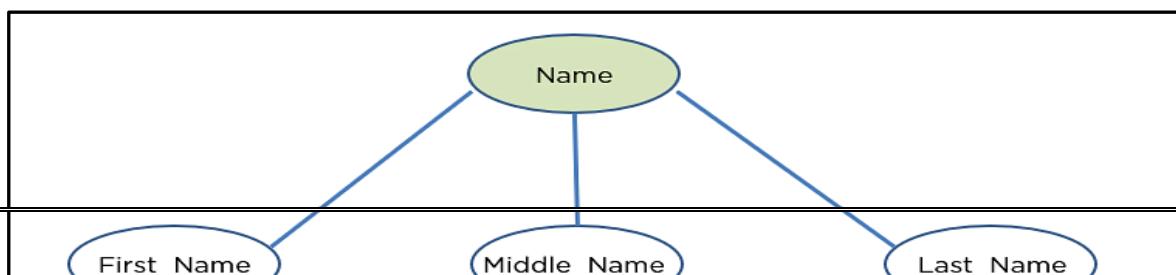
Example:



In the above example, Roll-No is an attribute that belongs to simple or atomic type. It cannot be further subdivided into parts.

2. **Composite Attribute:** An attribute that can be broken down into smaller components or parts is called composite attribute. An attribute that is composed of several other attributes is known as a composite attribute. An oval showcases the composite attribute, and the composite attribute oval is further connected with other ovals.

Example:



In the above example, Name is an attribute. It can be further sub-divided into smaller parts First_ Name, Middle & Last name.

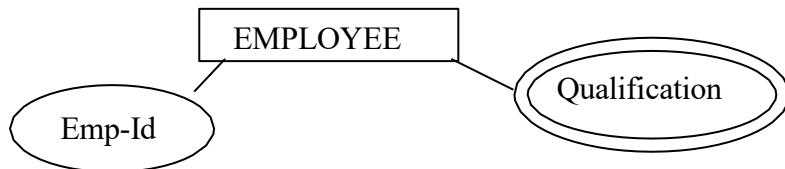
3. **Single Valued Attribute:** If an attribute takes only one value for each entity i.e, if the value is not repeated in the entity set then it is called as single valued attribute.

Example: Emp _Id Student _No Product _Id

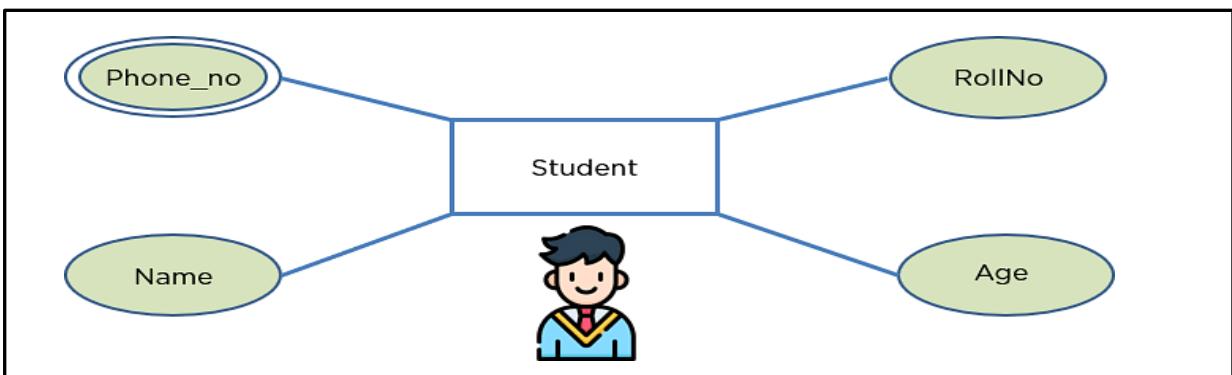
In the above example, Emp_id is an attribute, which belongs to single valued attribute because Emp_Id value cannot be EM repeated in EMPLOYEE entity set.

4. Multi Valued Attribute: An attribute that takes on more than one value for an entity set them it is called as multi valued attribute. Some attributes can possess over one value, those attributes are called multivalued attributes. The double oval shape is used to represent a multivalued attribute.

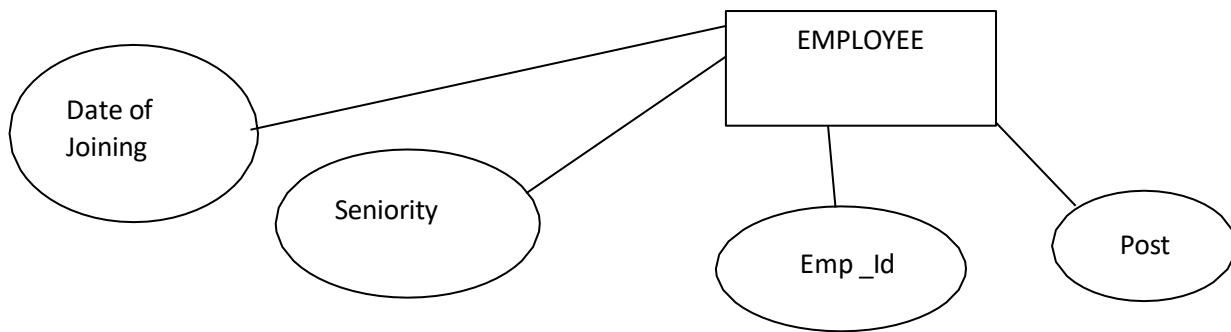
Example:



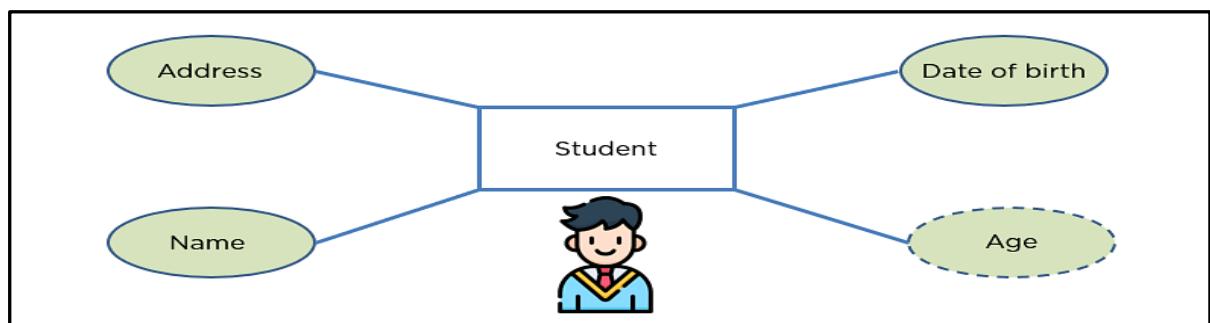
In the above example, Qualification is a multi-valued attribute represented by ellipse with double lines. In EMPLOYEE entity type, employee may have more than one qualification like M.Com, M.B.A. etc.,



5. Derived Attribute: An attribute whose value can be calculated or derived from another related value (i.e, related attribute) is called derived attribute. It is represented by an ellipse with dotted line. An attribute that can be derived from other attributes of the entity is known as a derived attribute. In the ER diagram, the dashed oval represents the derived attribute.



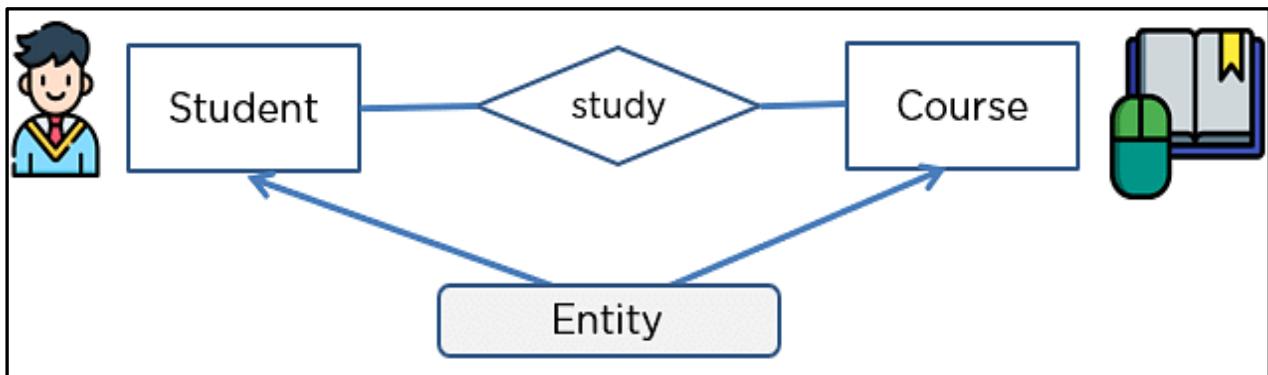
In the above, Seniority is derived attribute, which depends on other attribute Date of Join.



ENTITY and TYPES

Entity:

An entity can be either a living or non-living component. It showcases an entity as a rectangle in an ER diagram. For example, in a student study course, both the student and the course are entities. An entity may be a person, place, thing, event or concept.



The following is an example for entity:

PERSON	ENTITY
Person	: STUDENT, CUSTOMER, EMPLOYEE, etc..
Place	: COUNTRY, STATE, CITY etc..
Object	: MACHINE, BUILDING, PRODUCT etc..
Event	: ORDER, SALE, PURCHASE, etc..
Concept	: COURSE, ACCOUNT, POST etc..

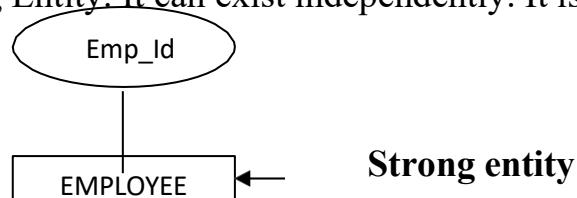
An entity must be in capitalized. An entity may be classified into two types :

Types of Entities:

1. Strong Entity

2. Weak Entity.

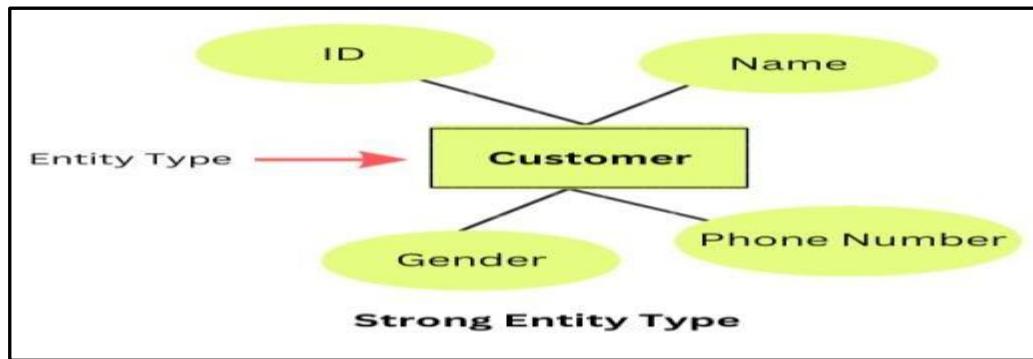
1. **Strong Entity:** An entity set that does not depend on another entity set (which has primary key) , is called Strong Entity. It can exist independently. It is represented by rectangle as shown below.



It is an entity that has its own existence and is independent. The entity relationship diagram represents a strong entity type with the help of a single rectangle. Below is the ERD of the

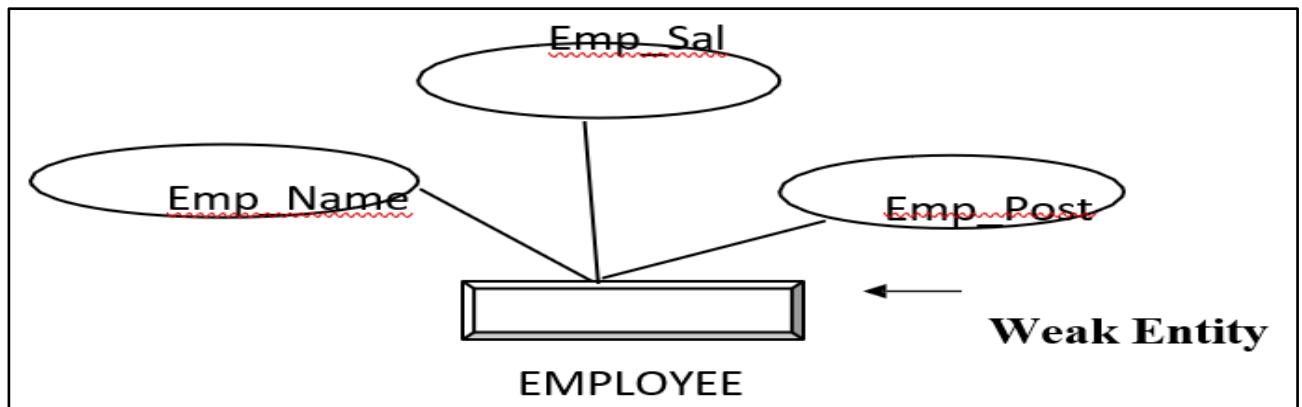
strong entity type:

In the below example, the "Customer" is the entity type with attributes such as ID, Name, Gender, and Phone Number. Customer is a strong entity type as it has a unique ID for each customer.



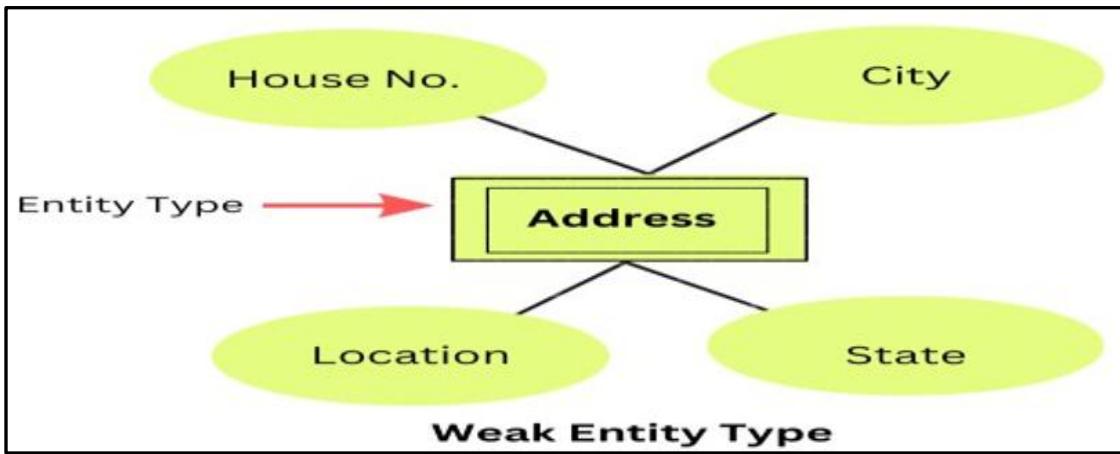
2. **Weak Entity:** An entity set which depend on other entity set (Which does not have primary key), is called Weak Entity. Weak Entity is represented by rectangle with double lines. It contains partial identifiers.

Example:

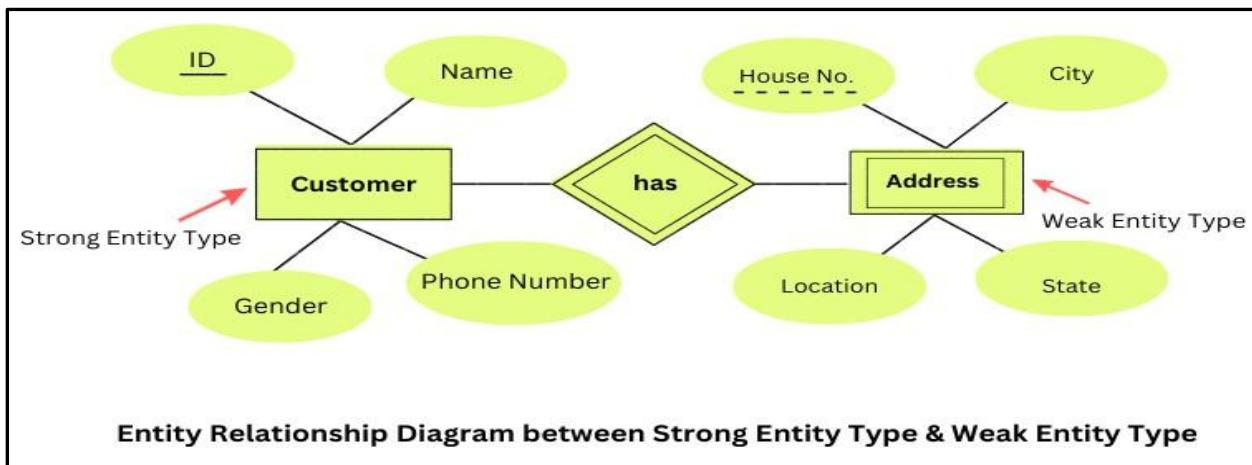


It is an entity that does not have its own existence and relies on a strong entity for its existence. The Entity Relationship Diagram represents the weak entity type using double rectangles. Below is the ERD of the weak entity type:

In the below example, "Address" is a weak entity type with attributes such as House No., City, Location, and State.



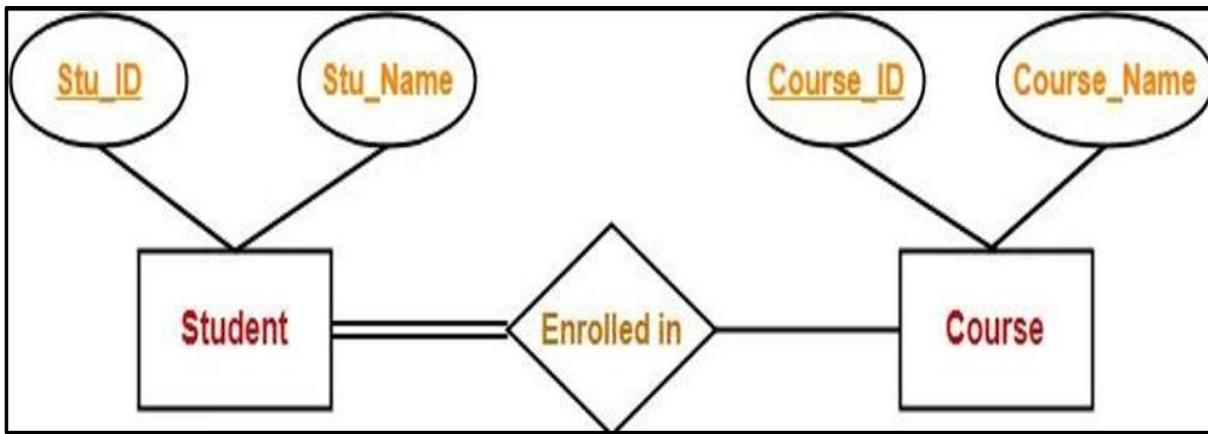
The relationship between a strong and a weak entity type is known as an identifying relationship. Using a double diamond, the Entity-Relationship Diagram represents a relationship between the strong and the weak entity type.



Entity Set:

An entity set in DBMS is a collection of similar entities. An entity is described using a set of attributes. Therefore, all entities in a given entity set have the same attributes. It is a named collection of related data. A collection of entities that shares common properties or characteristics is known as entity set of records.

An ER diagram shows the relationship among entity sets. An entity set is a group of similar entities and these entities can have attributes. In terms of DBMS, an entity is a table or attribute of a table in database, so by showing relationship among tables and their attributes, ER diagram shows the complete logical structure of a database. For example, an entity set of students, an entity set of motorbikes, an entity of smartphones, an entity of customers, etc.



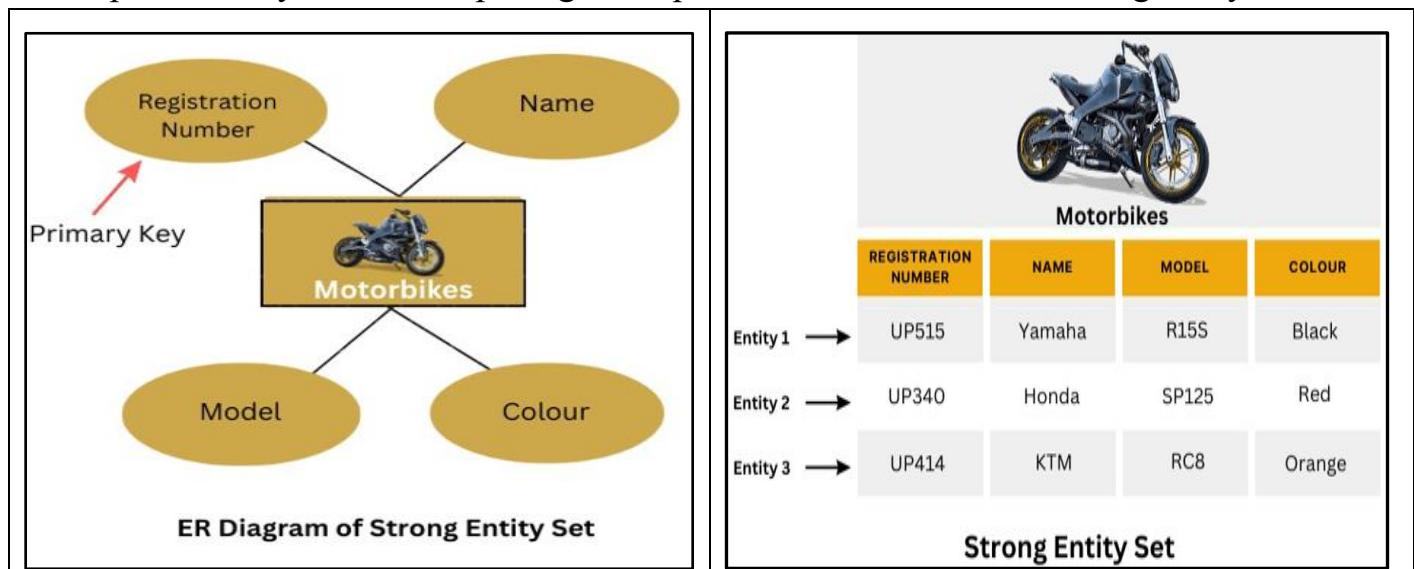
Entity sets can be classified into **Two Types**:

1. Strong Entity Set:

In a DBMS, a strong entity set consists of a primary key. For example, an entity of motorbikes with the attributes, motorbike's registration number, motorbike's name, motorbike's model, and motorbike's colour.

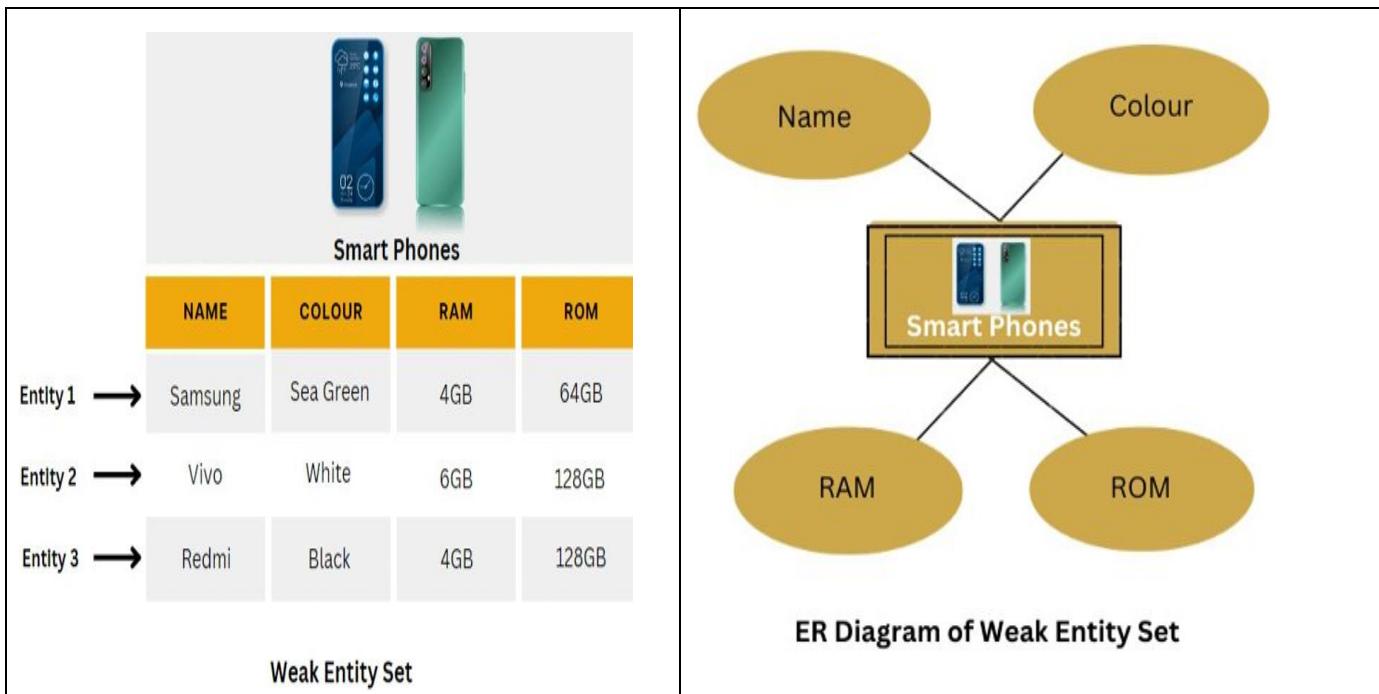
Below is the representation of a strong entity set in tabular form:

Example of Entity Relationship Diagram representation of the above strong entity set



Weak Entity Set:

In a DBMS, a weak entity set does not contain a primary key. For example, An entity of smartphones with its attributes, phone's name, phone's colour, and phone's RAM. Below is the representation of a weak entity set in tabular form:



Example of Entity Relationship Diagram representation of the above weak entity set:

ANOMALIES and TYPES

Anomaly:

An anomaly is an error or inconsistency that may result when user attempts update, insert and delete a data in a table that contains redundancy. Anomalies cause if there is no data integrity. There can be various reasons for anomalies to occur in the database. For example, if there is a lot of redundant data present in our database then DBMS anomalies can occur. If a table is constructed in a very poor manner, then there is a chance of database anomaly. Due to database anomalies, the integrity of the database suffers. Anomalies in the relational model refer to inconsistencies or errors that can arise when working with relational databases, specifically in the context of data insertion, deletion, and modification.

Anomalies means problems or inconsistency which happened during the operations performed on the table. There can be many reasons that anomaly occurs for example, It occurs when data is stored multiple times unnecessarily in the database i.e. redundant data is present or it occurs when all the data is stored in a single table. normalization is used to overcome the anomalies. The other reason for the database anomalies is that all the data is stored in a single table. So, to remove the anomalies of the database, normalization is the process which is done where the splitting of the table and joining of the table (different types of joins) occurs.

Types of Anomalies:

1. Modification or Updation Anomalies:

When we update some rows in the table, and if it leads to the inconsistency of the table then this anomaly occurs. This type of anomaly is known as an updation anomaly. In a table, with data redundancy, the modification of an attribute value must be made in more than one place. If this modification is not done at all multiple places, then, it is modification anomaly. These anomalies occur when modifying data in a database and can result in inconsistencies or errors.

For example, if a database contains information about employees and their salaries, updating an employee's salary in one record but not in all related records could lead to incorrect calculations and reporting.

2. Insertion anomalies:

In a table, with data redundancy, the insertion of an attribute value must be made in more than one place. If this insertion is not done at all multiple places, then it is insertion anomaly. These anomalies occur when it is not possible to insert data into a database because the required fields are missing or because the data is incomplete.

For example, if a database requires that every record has a primary key, but no value is

provided for a particular record, it cannot be inserted into the database.

3. Deletion anomalies:

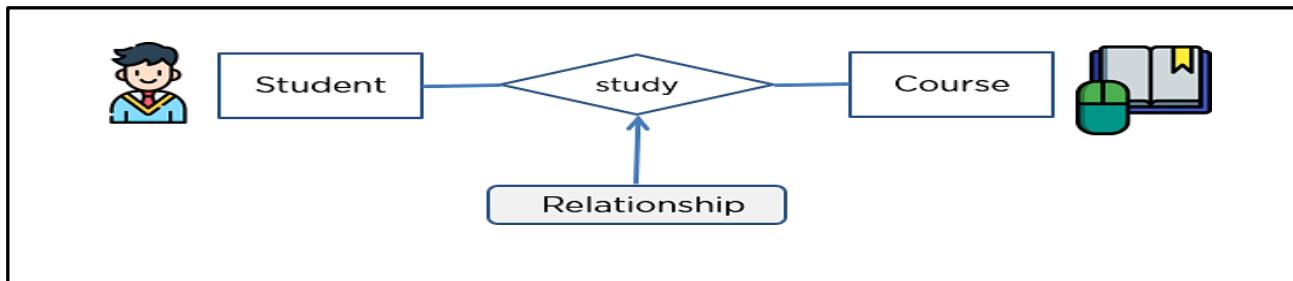
In a table, with data redundancy, the deletion of an attribute value must be made in more than one place. If this deletion is not done at all multiple levels or multiple places, then it is deletion anomaly. These anomalies occur when deleting a record from a database and can result in the unintentional loss of data.

For example, if a database contains information about customers and orders, deleting a customer record may also delete all the orders associated with that customer.

RELATIONSHIP and TYPES

Relationship:

An Association between two or more tables in database is called as Relationship. Relationship is created between or among the tables using foreign key. In the example below, both the student and the course are entities, and study is the relationship between them.



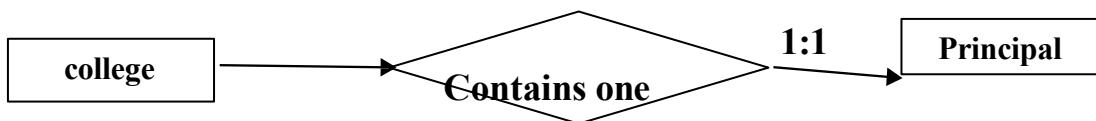
Types of Relationships:

1. One –to -One relationship (1:1)
2. One-to-Many Relationship(1:M)
3. Many-to-One Relationship(M:1)
4. Many-to-Many Relationship (M:M)

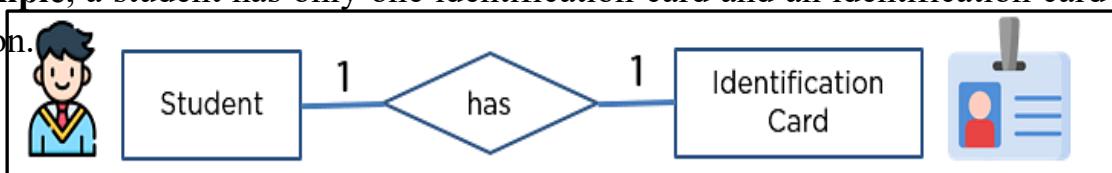
1. One –to -One Relationship (1:1):

One entity in 'X' relation is related with only one entity in 'Y' relation, in the same way one entity of 'Y' relation is related with only one entity of 'X' relation then it is said to be one to one relationship in this the relation 'Y' also indicates one side.

For example, each college contains only one principal and every principal belongs to only one college, hence the relationship between college and principal is one to one.



For example, a student has only one identification card and an identification card is given to one person.



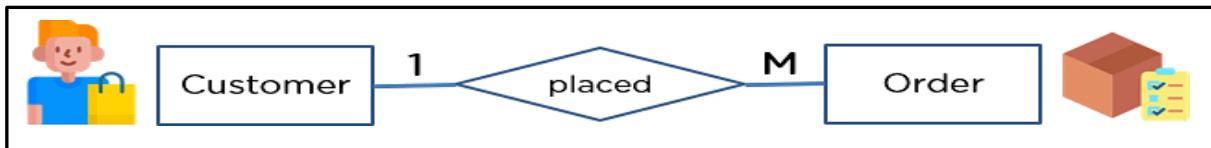
2. One to Many Relationship (1:M):

One entity in 'X' relation is related with many other entities in 'Y' relation, in the same way many entities of 'Y' relation are related with only one entity of 'X' relation then it is said to be one to many relationships. In this the relation 'X' indicates one side and the relation 'Y' indicates many sides.

For example, a university contains many colleges, but each college is affiliated to only one university, hence the relationship between university and colleges is one to many.



For example, a customer can place many orders, but an order cannot be placed by many customers.



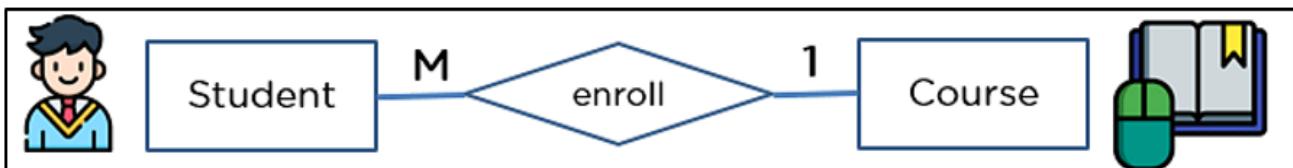
3. Many to One Relationship (M:1):

Many entities in 'X' relation is related with only one entity in 'Y' relation, in the same way one entity of 'Y' relation is related with many other entities of 'X' relation then it is said to be many to one relationship. In this the relation 'X', indicates many sides and the relation 'Y' indicates one side.

For example, many Students many joins One Course and each Course may contain many students hence the relationship between student a course is many to one.



For example, students have to opt for a single course, but a course can have many students.

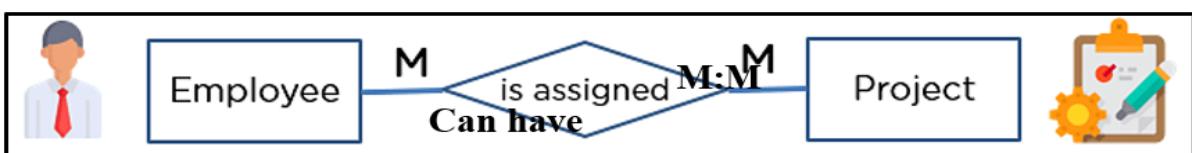


4. Many to Many Relation Ship (M:M):

Many entities in 'X' relation is related with many other entities in 'Y' relation in the same way many entities of 'Y' relation are related with many entities of 'X' relation then it is said to be Many to Many relationships. In this the relation 'X' indicates many sides and the relation 'Y' also indicates many sides.

For example, an employee can have many degrees (Qualifications) and each degree can be done by many employees, hence the relationship between employee and degree is many to many.

For example, you can assign an employee to many projects and a project can have many employees.



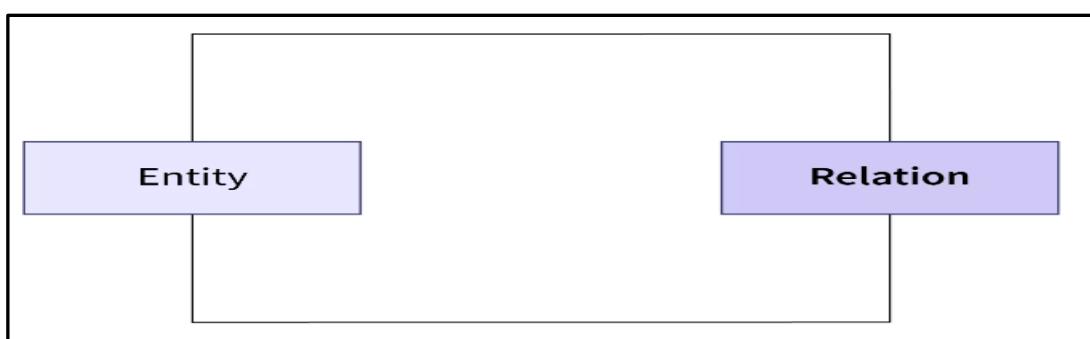
Degree of Relationship:

In DBMS, a degree of relationship represents the number of entity types that associate in a relationship. For example, we have two entities, one is a student and the other is a bag and they are connected with the primary key and foreign key.

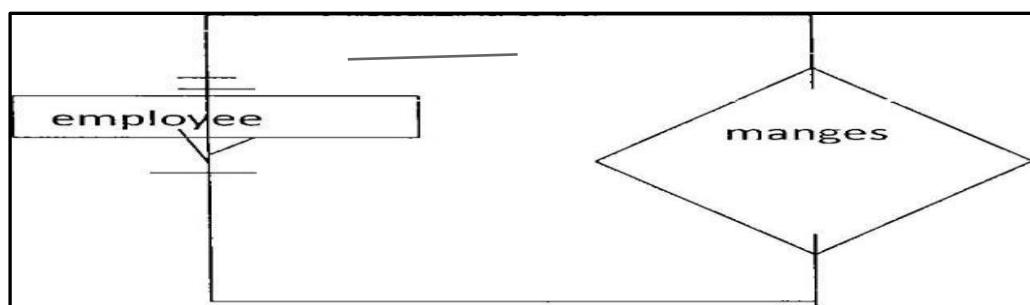
The degree of relationship can be classified into three types. They are:

- 1) **Unary relationship.**
- 2) **Binary relationship.**
- 3) **Ternary relationship.**

- 1) **Unary relationship:** A unary relationship is a relationship between the instances of a single entity type. It is also called as " Recursive relationship"

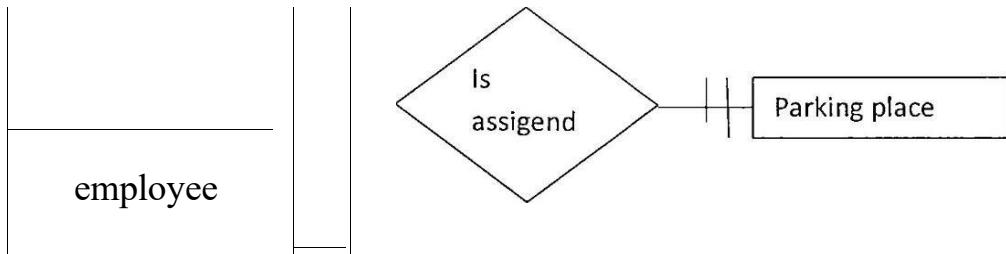


Example: In the college presidential elections, a president is chosen among the students. He/she leads the entire community and looks after the student-centric activities. Even though he/she is the president, but after all is a student. So, we can say that there is only one entity i.e., the student.

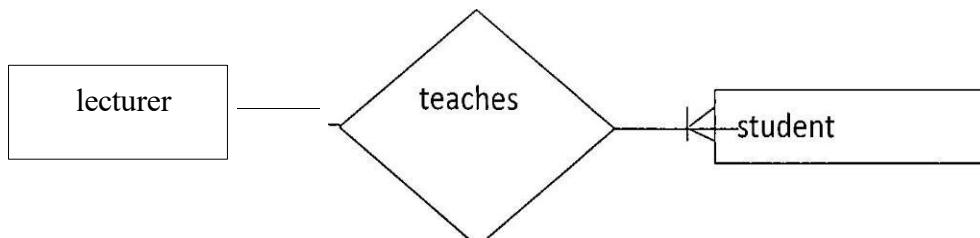


In the above example, "manages" is a "one to many" relationship between instances of an "Employee" entity type.

- 2) **Binary Relationship:** Binary relationship is a relationship between the instances of two entity types. In a Binary relationship, there are two types of entity associates. So, we can say that a Binary relationship exists when there are two types of entity and we call them a degree of relationship is 2. Or in other words, in a relation when two entity sets are participating then such type of relationship is known as a binary relationship.

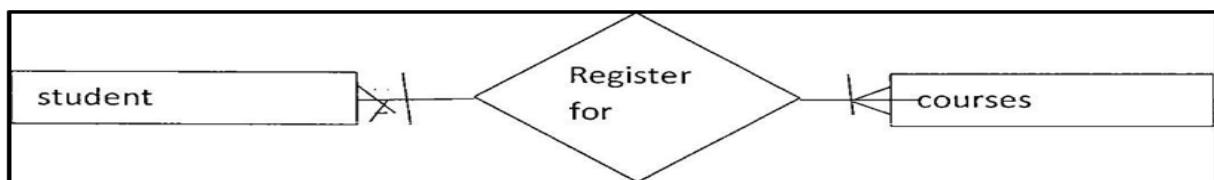


In the above example, it is a one-to-one relationship because an employee is assigned one parking place & each parking place is assigned to one employee.



(one to many)

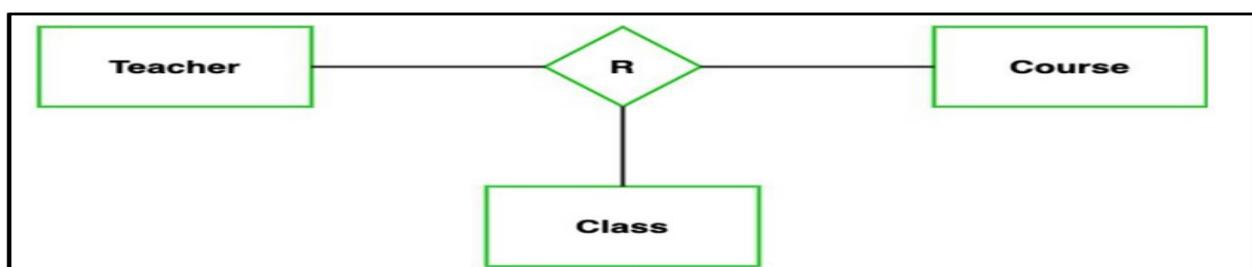
In the above example, it is a one-to-many relationship because a lecturer teaches to many students and many students are assigned to one lecturer.



(Many to Many)

In the above example, it is a many-to-many relationship because students may register for many courses & many courses may have many students.

3) Ternary Relationship: In the Ternary relationship, there are three types of entity associates. So, we can say that a Ternary relationship exists when there are three types of entity and we call them a degree of relationship is 3. A ternary relationship is a relationship among the instances of three entity types.



Example: We have three entity types 'Teacher', 'Course', and 'Class'. The relationship between these entities is defined as the teacher teaching a particular course, also the teacher teaches a particular class. So, here three entity types are associating we can say it is a ternary relationship.

KEY and TYPES

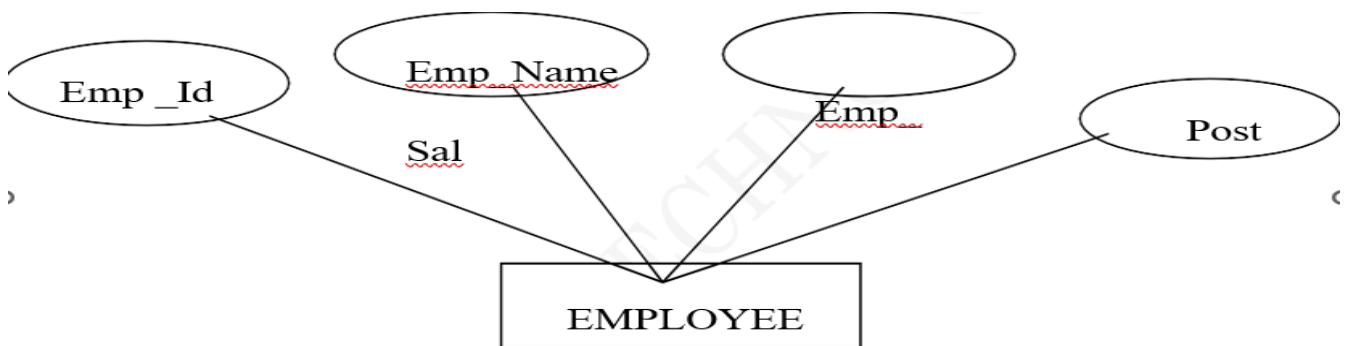
Key: A key refers to an attribute/a set of attributes that help us identify a row (or tuple) uniquely in a table (or relation). A key is also used when we want to establish relationships between the different columns and tables of a relational database. The individual values present in a key are commonly referred to as key values.

Key Attribute: Database is a collection of several records. To identify or retrieve particular record, we need some identifiable items. These identifiable items are known as a 'KEY' there are five different types of keys in the database. These are the components of the database table.

Types of Keys:

1. **Primary Key:** The primary key is used to identify a specific row in a table. It is an attribute that which uniquely identifies each record this key does not allow duplicate or null values and allows only unique values. The unique key is used to ensure that there is only one entry in a specific table together are used to identify a row. These keys help you to identify a particular column of a row of a table accurately and uniquely. Hence, it is very important to use the accurate columns as a key as per your use case. Primary key cannot consist of the same values reappearing/repeating for any of its rows.

Example: Emp_Id in EMPLOYEE table; Stu_Ht no in STUDENT table;



In the above example, an entity set or table EMPLOYEE contain an attribute Emp_Id is a primary key.

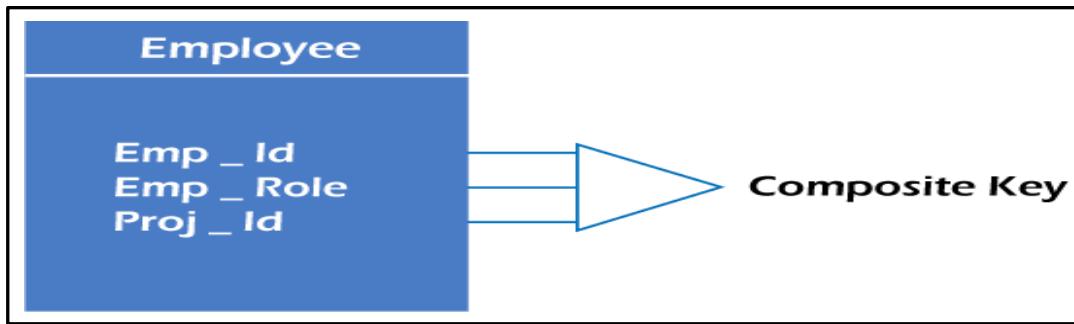
Rules for Primary Key: An attribute, which is declared as primary key, should follow two rules.

- (i) It should not have any NULL values.
- (ii) Data should not be repeated in that column.

2. **Composite Key:** A primary key that consists of a combination of attributes is known as composite primary key. And all the primary key rules are applied to the composite key

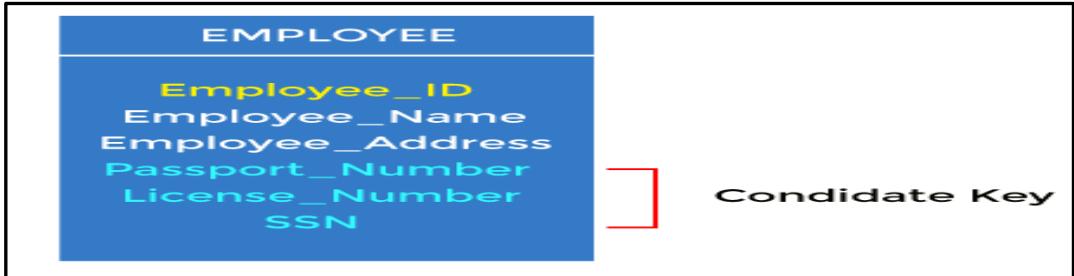
primary key. Whenever a primary key consists of more than one attribute, it is known as a composite key. This key is also known as Concatenated Key.

For example, in employee relations, we assume that an employee may be assigned multiple roles, and an employee may work on multiple projects simultaneously. So the primary key will be composed of all three attributes, namely Emp_ID, Emp_role, and Proj_ID in combination. So these attributes act as a composite key since the primary key comprises more than one attribute.



3. **Candidate Key:** An attribute or combinations of attributes that acts as like primary key i.e, A candidate key is similar to as primary key. But it is not selected as a primary key. Candidate key does not take the attribute of primary key, but candidate key follows the same rules of primary key. A candidate key is an attribute or set of attributes that can uniquely identify a tuple. **Except for the primary key, the remaining attributes are considered a candidate key.** The candidate keys are as strong as the primary key.

For example: In the EMPLOYEE table, id is best suited for the primary key. The rest of the attributes, like SSN, Passport_Number, License_Number, etc., are considered a candidate key.



4. **Secondary Key:** A non-primary key (i.e, which is not a primary key) is known as a Secondary key. The secondary Key is nothing but the Key which was not chosen to be the primary Key. From the set of candidate keys, one Key is chosen to be the primary Key, and others are called secondary Keys. They are also called alternate Keys as they uniquely identify a row in a table. Thus, an unselected candidate key used as a primary key is called a secondary key. The Key that uniquely identifies a row but is not selected as the primary Key is known as a secondary or alternate Key.

For example: Suppose you have a relational database for an online bookstore, and you have a table called "Books" with the following columns:

BookID (Primary Key) Title Author PublicationYear ISBN Genre Price

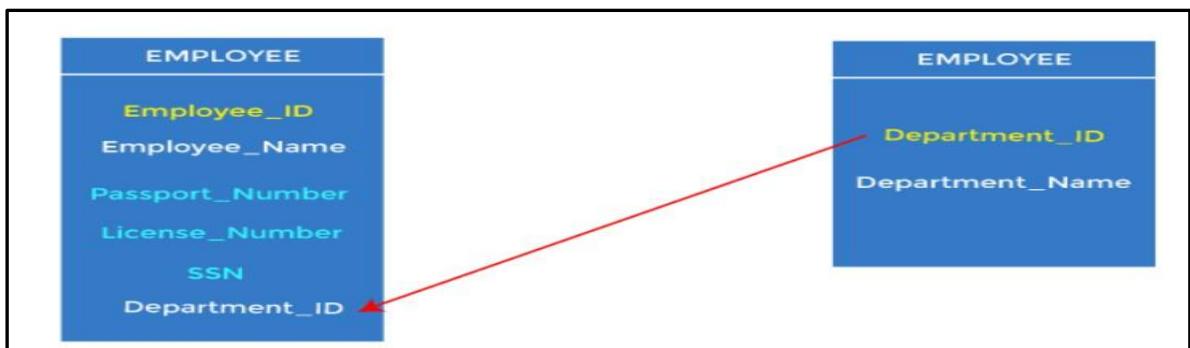
In this example, BookID is the primary key of the table, which uniquely identifies each book. However, you might frequently need to search for books by their Author or Genre. To improve the performance of these queries, you can create secondary keys on the Author and Genre columns.

5. **Foreign Key:** An attribute that establishes the relationship between relational data base tables is called as Foreign Key. A foreign key is an attribute or primary key in one table, foreign key is used to create relationship among the tables in a same database. foreign key is also called as a “Reference Key”. It is indicated with dashed line. Foreign keys are the column of the table used to point to the primary key of another table.

Every employee works in a specific department in a company, and employee and department are two different entities. So, we can't store the department's information in the employee table. That's why we link these two tables through the primary key of one table.

We add the primary key of the DEPARTMENT table, Department_Id, as a new attribute in the EMPLOYEE table.

In the EMPLOYEE table, Department_Id is the foreign key, and both the tables are related.



6. **Unique Key:** A unique key refers to a column/a set of columns that identify every record uniquely in a table. All the values in this key would have to be unique. Remember that a unique key is different from a primary key. It is because it is only capable of having one null value. A primary key, on the other hand, cannot have a null value.

ER-MODELS

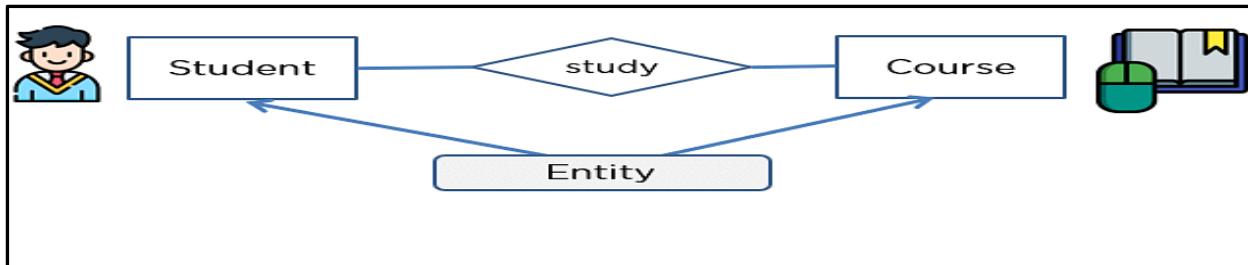
ER (Entity-Relationship) models are used to represent the entities, relationships, and attributes of a system in a visual and graphical way. It is used to design and describe the relationships between entities in a system, which helps in understanding the data requirements of the system. ER models are represented using various symbols and notations to create a visual representation of the relationships between entities. An Entity-Relationship Model represents the structure of the database with the help of a diagram. ER Modelling is a systematic process to design a database as it would require you to analyze all data requirements before implementing your database.

Why Use ER Diagrams in DBMS?

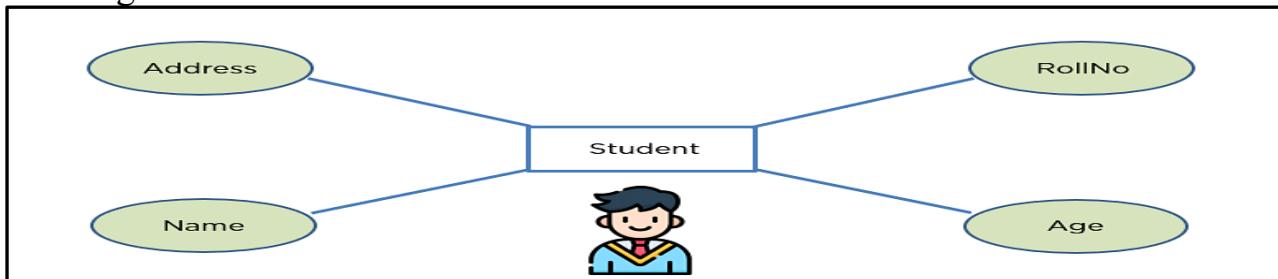
- ER Diagram helps you conceptualize the database and lets you know which fields need to be embedded for a particular entity.
- ER Diagram gives a better understanding of the information to be stored in a database.
- It reduces complexity and allows database designers to build databases quickly.
- It helps to describe elements using Entity-Relationship models.
- It allows users to get a preview of the logical structure of the database.

Components of ER Diagram:

Entities are objects or concepts that are relevant to the system being modelled. For example, in an Employee Management System, the entities would be Employee, Department, Project, etc. Entities are represented by rectangles in an ER diagram.

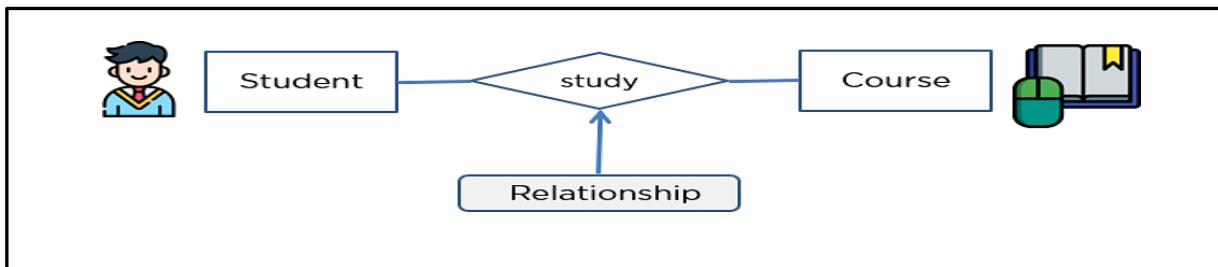


Attributes are characteristics or properties of an entity. For example, an employee entity would have attributes like Name, Address, Salary, etc. Attributes are represented by ovals in an ER diagram.



Relationships define the association between entities. For example, an employee works for a department and a department has many employees. So, there is a relationship between

Employee and Department entities. Relationships are represented by diamond shapes in an ER diagram.



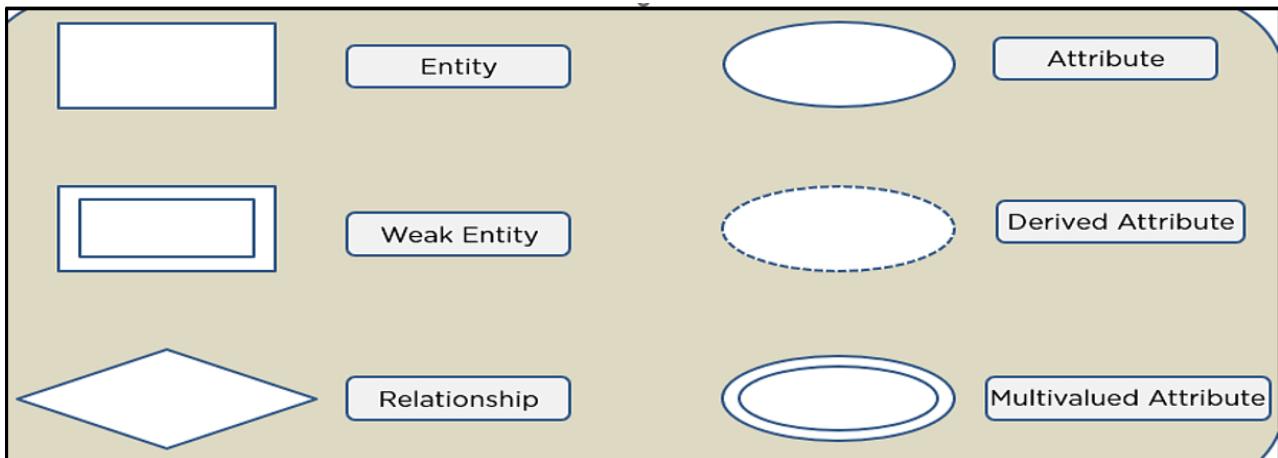
The symbols used in the ER model diagram are:

ER models consist of three main components: **Entities, Attributes, and Relationships**. Rectangles represent entities.

Ellipses represent attributes.

Diamonds represent relationships.

Lines represent connections between entities.

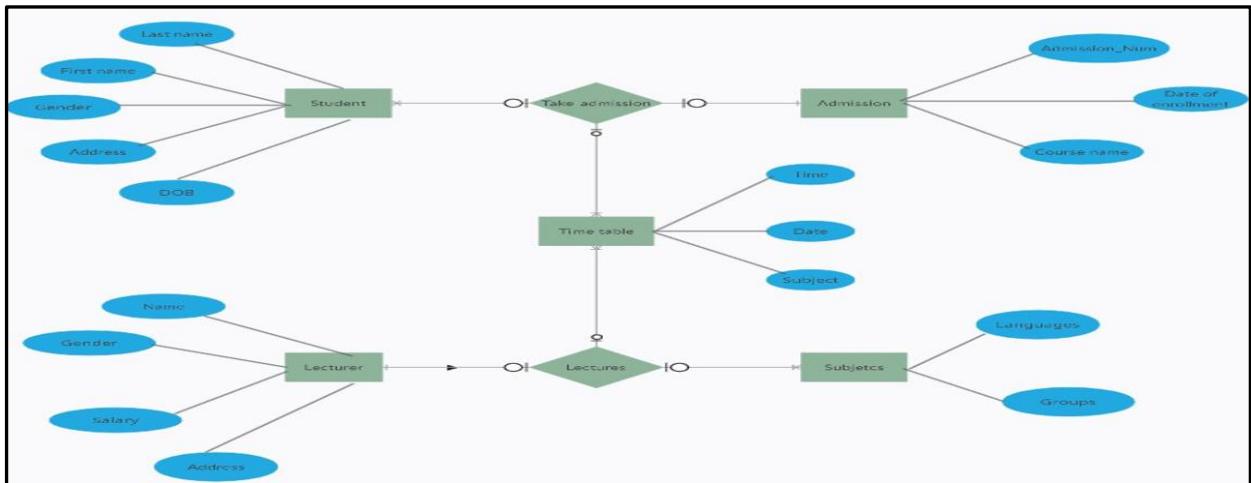


ER models are a useful tool for database designers to visualize and design the logical structure of a database before implementing it in SQL.

How to Draw an ER Diagram:

- First, identify all the Entities. Embed all the entities in a rectangle and label them properly.
- Identify relationships between entities and connect them using a diamond in the middle, illustrating the relationship. Do not connect relationships with each other.
- Connect attributes for entities and label them properly.
- Eradicate any redundant entities or relationships.
- Make sure your ER Diagram supports all the data provided to design the database.
- Effectively use colours to highlight key areas in your diagrams.

COLLEGE DATA BASE E-R DIAGRAM:



Entity-Relationship (ER) Diagram is a diagram that displays the relationship of entity sets stored in a database. As shown in the College ER Diagram, the entity-relationship diagrams help to explain the logical structure of a college that includes Admissions, Lecturers, Students, and Subjects. The attached diagram contains different symbols that use **Rectangles** to represent **Entities**, **Ovals** to define **Attributes**, and **Diamond** shapes to represent **Relationships**. As represented in the diagram, a college ER diagram is a blended learning approach of the college system that combines different attributes of the **Student** like FirstName, LastName, Gender, Address, DOB. At the same time, the **Admission** like Admission_Num, DateOfEnrollment and CourseName etc.....

□ Entity type becomes a Table:

In the given ER diagram, STUDENT, ADMISSION, TIMETABLE, LECTURE, and SUBJECTS forms individual tables.

Convert ER Diagrams to Tables:

STUDENT:

FirstName	LastName	Gender	Address	DOB

ADMISSION:

Admission_Num	DateOfEnrollment	CourseName

TIMETABLE:

Time	Date	Subject

SUBJECTS:

Name	Gender	Salary	Address

Languages	Groups

DEPENDENCY and TYPES

Dependency:

A dependency is a relationship between two or more attributes in a database. It describes how the value of one attribute is determined by the value of another attribute. A dependency is a constraint that governs or defines the relationship between two or more attributes. A database dependency is a constraint that defines the relationship between attributes. It happens when information stored in the same database table uniquely determines other information stored in the same table. It's important to understand what database dependencies are because they provide the basic building blocks for database normalization. Dependencies in DBMS is a relation between two or more attributes.

It is a constraint or rule between two attributes. In a table, if the attribute 'X' is functionally dependent on the attribute 'Y' and the values of attribute 'Y' uniquely determines the value of attribute 'X', then functional dependency will occur. It is denoted as $Y \rightarrow X$, where the attribute set on the left side of the arrow, Y is called Determinant, and X is called the Dependent.

If the values of attribute 'Z' is also dependent on 'Y' attribute then it can be written as: $Y \rightarrow X, Z$

It can read as 'Y' attribute implies 'X' and 'Z' attributes.

Note: The attributes on left-hand side of arrow are called as determinant, where as the right-hand side of arrow are called as dependents.

Determinant: Attribute 'X' of relation R is said to be determinant if and only if it uniquely defines the values of attributes 'Y' in relation R.

To say an attribute as determinant, attribute need not be a key attribute.

Ex: Marks attribute decide/determine Grade attribute. So, Marks attribute is determinant.

Ex: Emp-Id \rightarrow Emp-Name, Emp-Sal, Emp-Post.

Emp-Id is determinant. Emp-Name, Emp-Sal, Emp-Sal, Emp-Post are dependents.

Types of Database Dependencies:

- Functional Dependency**
- Partial Dependency**
- Transitive Dependency**
- Multivalued Dependency**

1. Functional Dependency:

In a relational database, an attribute (or a set of attributes) is said to be functionally dependent on another attribute (or set of attributes) if the value of the first attribute uniquely determines the value of the second attribute. It is denoted as $A \rightarrow B$, where A determines B.

A functional dependency occurs when the information stored in a table can uniquely determine other information in the same table. Think of it as an association between two attributes of the same relation. A functional dependency is a relationship between two or more attributes of a table, where the value of one attribute (or a set of attributes) determines the value of another attribute (or a set of attributes). For example, in a table of employees, the employee ID (EID) functionally determines the name, department, and salary of each employee. This means that for any given EID, there is only one possible name, department, and salary. We can write this as $EID \rightarrow \text{name, department, salary}$. **Example:** Consider a relation schema R with attributes $\{\text{StudentID, Course, Instructor}\}$ representing information about students, the courses they take, and the instructors who teach those courses.

StudentID	Course	Instructor
-----------	--------	------------

In this example, the combination of $\{\text{StudentID, Course}\}$ uniquely determines the value of the instructor.

2. Partial Dependency:

A partial dependency occurs when a non-prime attribute is functionally dependent on part of the candidate key. When a non-prime attribute (an attribute that is not part of any candidate key) is functionally dependent on only part of a candidate key, it is referred to as a partial dependency. One or more non-key attributes that are functionally dependent on part of the primary key, but not fully on primary key then it is called Partial dependency.

For example, let's consider a relation with attributes A, B, and C. Suppose $\{A, B\}$ is a candidate key. If attribute C is functionally dependent only on attribute B and not on the entire candidate key $\{A, B\}$, then we have a partial dependency. This is because the value of C depends on only a part of the candidate key.

Example: Consider a table representing information about students and their courses. The attributes are StudentID, CourseID, StudentName, CourseName, and Instructor.

StudentID	CourseID	StudentName	CourseName	Instructor
-----	-----	-----	-----	-----

In this example, the candidate key could be the combination of StudentID and CourseID. Both together uniquely identify each row in the table. Now, let's say we have a situation where the Instructor is functionally dependent only on the CourseID but not on the entire candidate key (StudentID and CourseID). This is an example of partial dependency.

3. Transitive Dependency:

Transitive dependencies occur when there is an indirect relationship that causes a functional dependency. Transitive dependencies occur when a non-key element determines the value for another non-key element. If A, B, and C are attributes of relation R, such that $A \rightarrow B$, and $B \rightarrow C$, then C is transitively dependent on A, unless either B or C is a candidate key or

part of a candidate key. Equivalently, a transitive dependency exists when a nonprime attribute determines another nonprime attribute.

Example: Consider a table representing information about employees

EmployeeID	EmployeeName	Department	ManagerID	ManagerName
------------	--------------	------------	-----------	-------------

In this example, there is a transitive dependency between EmployeeID and ManagerName through the intermediate attribute ManagerID. The dependency can be expressed as follows:

EmployeeID \rightarrow ManagerID (Direct dependency)
ManagerID \rightarrow ManagerName (Direct dependency)

Therefore, EmployeeID \rightarrow ManagerName (Transitive dependency)

Note: If it depends on the whole primary key, it is a direct functional dependency; if it depends on part of the primary key, it is a partial dependency; and if it depends on another non-key attribute, it is a transitive dependency.

4. Multivalued Dependency:

Multi-valued dependency (MVD) is a concept that deals with dependencies among attributes in a relation where a set of attributes collectively determines another set of attributes, and the dependency involves multiple values rather than just single values. Occurs when a value in one attribute uniquely determines a set of values in another attribute.

The notation for a multi-valued dependency is $X \rightarrow\!\!\!> Y$, where X and Y are sets of attributes.

Let R be a relation schema and let X, Y, and Z be sets of attributes in R. The multi-valued dependency $X \rightarrow\!\!\!> Y$ holds in R if, for every valid instance r of R, whenever two tuples t1 and t2 in r agree on all attributes in X, they must also agree on all attributes in Y.

Example: Consider a relation schema for a database that stores information about employees and their projects.

EmployeeID	ProjectID	Skills
------------	-----------	--------

In this example, the multi-valued dependency EmployeeID $\rightarrow\!\!\!> Skills$ holds because if we take any two tuples with the same EmployeeID (e.g., the first and third tuples), they share the same set of skills. This ensures that the values in the "Skills" attribute are uniquely determined by the "EmployeeID" attribute.

NORMALIZATION

Normalization is a step-by-step procedure, that is the process of decomposing a relation into smaller and well-structured relations and also grouping the required attributes into a relation. Normalization is the process to eliminate data redundancy and enhance data integrity in the table. Normalization also helps to organize the data in the database. It is a multi-step process that sets the data into tabular form and removes the duplicated data from the relational tables. Normalization organizes the columns and tables of a database to ensure that database integrity constraints properly execute their dependencies. It is a systematic technique of decomposing tables to eliminate data redundancy (repetition) and undesirable characteristics like Insertion, Update, and Deletion anomalies.

- ❖ The primary goal of normalization is to remove dependencies, which causes data redundancy and anomalies.
- ❖ Database normalization is the process of efficiently organizing data in a database. There are two reasons of this normalization process:
- ❖ Eliminating redundant data. For example, storing the same data in more than one table.
- ❖ Ensuring data dependencies make sense.

Both these reasons are worthy goals as they reduce the amount of space a database consumes and ensures that data is logically stored. Normalization consists of a series of guidelines that help guide you in creating a good database structure. Normalization guidelines are divided into normal forms; think of a form as the format or the way a database structure is laid out. The aim of normal forms is to organize the database structure, so that it complies with the rules of first normal form, then second normal form and finally the third normal form.

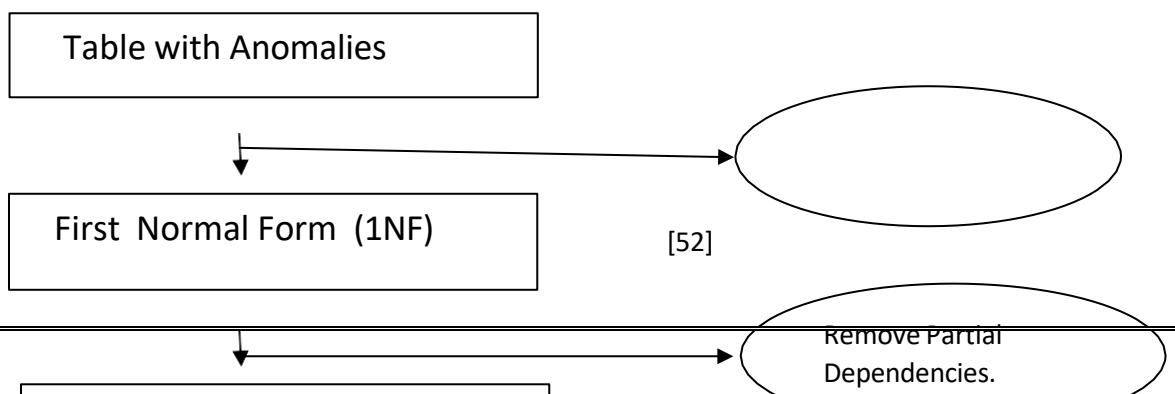
Normal Form: In a normalization step-by-step procedure, each step or rule applied on a state of a relation, and it is called a “Normal form” a normal form applying simple rules regarding functional dependencies.

Purpose of Normalization:

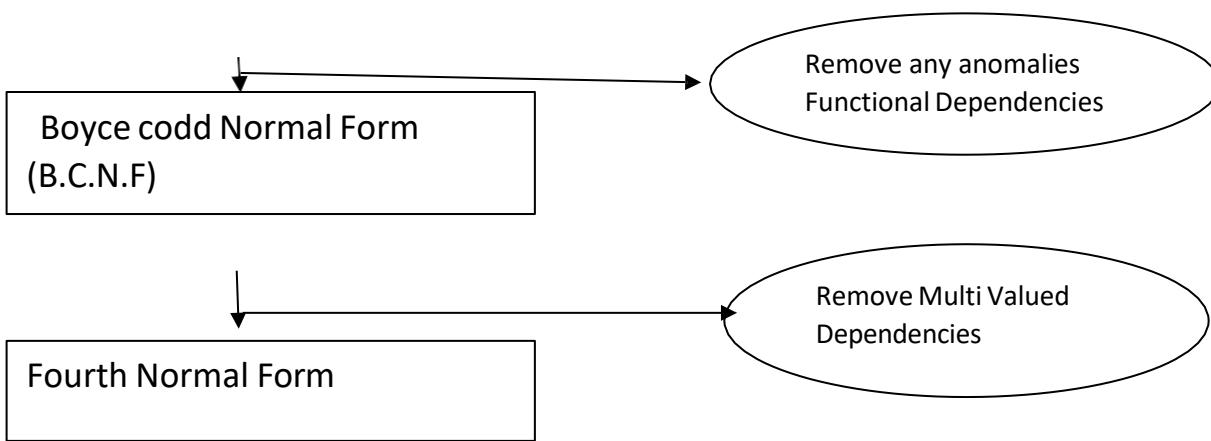
To structure the data, so that there is no repetition of data.

To permit simple retrieving the data in response to query and report requests. To simplify the maintenance of data i.e , updatons , insertions, deletions.

Steps of Normalization:



Remove
Multi valued
Attributes.

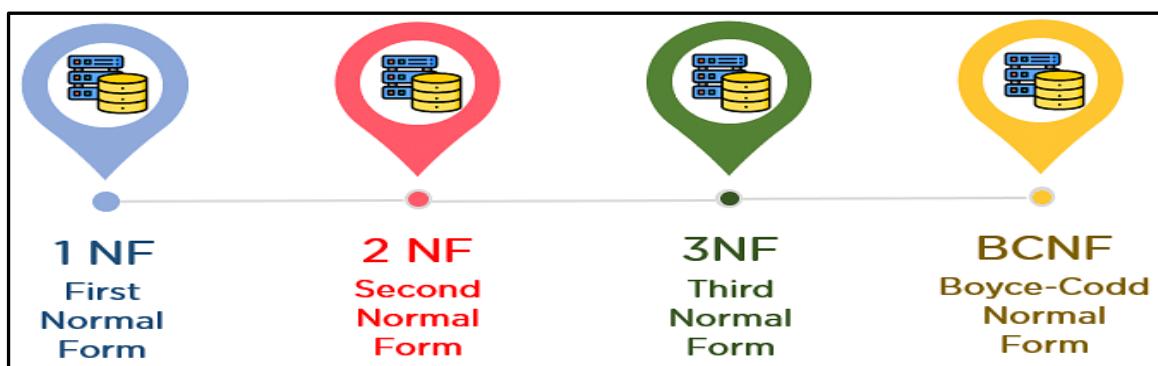


Steps of Normalization:

There are five types of normal forms:

1. **First Normal form (1 NF)**
2. **Second normal form (2 NF)**
3. **Third normal form (3 NF)**
4. **Boyce codd Normal form (BCNF)**
5. **Fourth normal form (4 NF)**

- 1 NF, 2 NF, 3NF, are called as basic normal forms.
- BCNF and 4NF are called as advanced normal forms .



Now let's understand the types of Normal forms with the help of examples. Consider the following table with anomalies:

Table: STUDENT

Rno	Sname	Group	Fee	Skills	Faculty
101	krishna	B.Sc	7000	C DBM S C++	Bhadram Vineela Gowtha mi

102	uday	B.Com	6000	Java VB	Chary Naresh
103	vicky	B.Tech	10000	java	chary
104	Srinivas	B.Sc	7000	Asp C	Sathyam

The above table STUDENT multi valued attributes. By removing multi valued attributes, it is in First Normal Form.

1. First Normal Form (1 NF) :

A table is said to be in 1 NF if it satisfies (meets) following conditions:

- If any multi valued attributes it should be removed and also there must be an atomic (single value) value at the intersection of each row and column.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- The Primary Key requirements are met (PK),

A table is referred to as being in its First Normal Form if atomicity of the table is 1. Here, atomicity states that a single cell cannot hold multiple values. It must hold only a single-valued attribute. The First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

In the **STUDENT** table, you can see that the course column has two values. Thus, it does not follow the First Normal Form. Now, if you use the First Normal Form to the above table, you get the below table as a result.

Table: STUDENT

R no	Sname	Group	Fee	Skills	Faculty
101	krishna	B.Sc	7000	C	Bhadram
101	krishna	B.Sc	7000	DBMS	Vineela
101	krishna	B.Sc	7000	C++	Gowthami
102	uday	B.com	6000	Java	Chary
102	uday	B.com	6000	VB	Naresh
103	vicky	B. TECH	10,000	Java	Chary
104	Srinivas	B.Sc	7000	ASP	Satyam

104	Srinivas	B.Sc	7000	C	Bhadram
-----	----------	------	------	---	---------

Table: EMPLOYEE

EMPLOYEE table:			
EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

The decomposition of the EMPLOYEE table into 1NF has been shown below:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	Up
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

By applying the First Normal Form, you achieve atomicity, and also every column has unique values.

2. Second Normal Form (2 NF):

A table is said to be in second normal form if it meets following conditions:

- A table must be in 1 NF.
- Every non-key attribute is fully functionally dependent on the primary key attributes.
- Primary key (PK) consists only one attribute.
- Any partial dependency should be removed.

Roll No	S Name	Group	Fee	Skills	Facult
P.F.D					y

↑
P.F.D

To bring the table to Second Normal Form, you need to split the table into two parts. This will give you the below tables:

Table: STUDENT

Roll No	S name	Group
101	Guptha	B.Sc
102	Ramu	B.Com
103	Raju	B.Tech
104	Srinivas	B.Sc

Table: SKILLS

Roll No	Skills	Faculty
101	C	Bhadram
101	DBMS	Vineela
101	C++	Gowthami
102	Java	Chary
102	VB	Naresh
[56]103	Java	Chary
104	ASP	Sathyam
104	C	Bhadram

Example: Let's assume, a school can store the data of teachers and the subjects they teach. In a school, a teacher can teach more than one subject.

TEACHER table		
TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30
47	English	35
83	Math	38
83	Computer	38

In the given table, non-prime attribute TEACHER_AGE is dependent on TEACHER_ID which is a proper subset of a candidate key. That's why it violates the rule for 2NF.

To convert the given table into 2NF, we decompose it into two tables:

TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38
TEACHER SUBJECT table:	
TEACHER_ID	SUBJECT
25	Chemistry
25	Biology
47	English
83	Math
83	Computer

3. Third Normal Form(3 NF):

The table is said to be in 3 NF, if it meets following conditions:

- The following relation (i.e, table) must be in 2 nd normal form,
- It has no transitive dependencies.
- If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

Ex: Table : **Group**

Group	Fee
-------	-----

Table : **STUDENT**

Roll No	S name	Group
---------	--------	-------

Table : **SKILLS**

Roll No	Skills	Faculty
---------	--------	---------

Now to change the table to the third normal form, you need to divide the table as

R No	Skills	Faculty
101	C	Bhadram
101	DBMS	Vineela

shown below: Table : **GROUP**

Table : **STUDENT**

Table : **SKILLS**

Group	Fee
B.Sc	7000
B.Com	6000
B.Tech	10,000
B.Sc	7000

R No	Sname	Group
101	Guptha	B.Sc
102	Ramu	B.Com
103	Raju	B.Tech
104	Srinivas	B.Sc

Example2: EMPLOYEE_DETAIL table:

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago
555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

Candidate key: {EMP_ID}

Non-prime attributes: In the given table, all attributes except EMP_ID are non-prime.

Here, EMP_STATE & EMP_CITY dependent on EMP_ZIP and EMP_ZIP dependent on EMP_ID. The non-prime attributes (EMP_STATE, EMP_CITY) transitively dependent on super key(EMP_ID). It violates the rule of third normal form.

That's why we need to move the EMP_CITY and EMP_STATE to the new <EMPLOYEE_ZIP> table, with EMP_ZIP as a Primary key.

EMPLOYEE table:		
EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

EMPLOYEE_ZIP table:		
EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago
06389	UK	Norwich
462007	MP	Bhopal

4. Boyce codd Normal Form (BCNF):

The table is said to be in BCNF, if it meets following conditions:

- The table (or Relation) must be in 3NF,

- Should not contain any other functional dependencies.

A relation R is BCNF, iff (if and only if) or vice-versa every determinant must be a candidate key. B.C.N.F is stronger than 3N.F, (Third Normal form). Every BCNF relation is also in 3 NF, but not every 3 NF relation is a B.C.N.F relation.

Consider the relation,

RESULT (Stu_No, Email_Id, Course_Id, Marks)

In the above relation RESULT, we have two candidate keys (Stu_No, Course_Id) & (Course_Id, Email_Id). Here, course_id is overlapping among these candidate keys. Hence, these candidate keys are called as overlapping candidate keys.

In the RESULT relation, the non-key attribute “Marks” is non- transitively and fully functionally dependent on key attributes. Hence, this is in 3 NF. But this is not in BCNF, because, there are 4 determinants in this relation. They are:

(i) Stu_No \square Email_Id (Stu_No determines

Email_Id) (ii) Email_Id \square Stu_No (Email_Id determines Stu_No)

(iii) (Stu_No, Course_Id) (Stu_No, Course_Id which determines remaining attributes of relation) (iv) (Course_Id, Email_Id) (Course_Id, Email_Id which determines remaining attributes of relation)

All above determinants are not candidate keys. Email_Id decides (or) determines Stu_No, but Email_Id on its own is not a candidate key. Only combination of (Stu_No, Course_Id) and (Course_Id, Email_Id) are candidate keys.

To make this table BCNF, we need to split the table into the following structure.

Stu-No	Email-Id	Stu-No	Course-Id	Marks
STUDENT		COURSE		

Now, both the tables are not only in 3 NF, but also in BCNF because, all the determinants are candidate keys and no functional dependencies.

In the COURSE relation, (Stu-No, Course-Id) is only determinant & candidate key.

If the relation has only one non-composite candidate key & it is 3 NF, then it is also in B.C.N.F.

Example2: Let's assume there is a company where employees work in more than one department.

EMPLOYEE table:				
EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232

In the above table Functional dependencies are as follows: $EMP_ID \rightarrow EMP_COUNTRY$
 $EMP_DEPT \rightarrow \{DEPT_TYPE, EMP_DEPT_NO\}$

Candidate key: $\{EMP_ID, EMP_DEPT\}$

The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys. To convert the given table into BCNF, we decompose it into three tables.

5. Fourth Normal Form (4 NF):

A table is said to be in 4 NF if it meets the following conditions

- A table (or relation) must be in B.C.N.F.
- Should not contain Multi – Valued Dependencies.

Multi-valued Dependency: This type of dependency exists when there are at least three attributes in a relation like A, B, & C. For each value of 'A' there is a well-defined set of values of 'B' attribute and well-defined set of values of 'C' attribute.

The set of values of 'B' is independent of set of values of 'C' and vice-versa

Example: **COURSE (a)** **COURSE(b)**

Course	Instructor	Text book
C	Rahul Vimal	Let Us C C-Prog
C++	Dinesh	Master C++ C++ -Prog

Course	Instructor	Text book
C	Rahul	Let Us C
C	Vimal	C-Prog
C	Rahul	C-Prog
C	Vimal	Let Us C
C++	Dinesh	Master C++
C++	Dinesh	C++ - Prog

(i) In the table, COURSE (b) converted into 1 NF, removed multi-valued attributes. This table is in 1 NF and the table COURSE (b) doesn't have determinants dependencies, so it is in BCNF.

(ii) But, COURSE (b) table have redundancy because of multi-valued dependencies.

(iii) To remove these multi-valued dependencies from a relation, divide the table into 2 new relations as shown below.

INSTRUCTOR

Instructor	Course
Rahul	C
Vimal	C
Dinesh	C++

TEXT BOOK

Text book	Course
Let Us C	C
C - Prog	C
Master C++	C++
C++ - Prog	C++

There are 18 cells of data in the table COURSE (b) and 14 cells of data in both the tables INSTRUCTOR and TEXT BOOK. So, 4 cells are saved using 4 NF. And multi – valued dependencies are removed.

Example2:

STUDENT		
STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Math	Singing
34	Chemistry	Dancing
74	Biology	Cricket
59	Physics	Hockey

The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entities. Hence, there is no relationship between COURSE and HOBBY.

In the STUDENT relation, a student with STU_ID, 21 contains two courses, Computer and Math and two hobbies, Dancing and Singing. So, there is a multi-valued dependency on STU_ID, which leads to unnecessary repetition of data.

So, to make the above table into 4NF, we can decompose it into two tables:

STUDENT_COURSE	
STU_ID	COURSE
21	Computer
21	Math
34	Chemistry
74	Biology
59	Physics

STUDENT_HOBBY	
STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket
59	Hockey

Module-3: Introduction to SQL

- SQL stands for Structured Query Language
- SQL was initially developed at IBM in the 1970s
- SQL is the standard language to communicate with relational database management systems like Oracle, MS Access, MS SQL Server, MySQL, DB2, Sybase Etc...

History of SQL:

"A Relational Model of Data for Large Shared Data Banks" was a paper which was published by the great computer scientist "E.F. Codd" in 1970.

The IBM researchers Raymond Boyce and Donald Chamberlin originally developed the SEQUEL (Structured English Query Language) after learning from the paper given by E.F. Codd. They both developed the SQL at the San Jose Research laboratory of IBM Corporation in 1970.

At the end of the 1970s, relational software Inc. developed their own first SQL using the concepts of E.F. Codd, Raymond Boyce, and Donald Chamberlin. This SQL was totally based on RDBMS. Relational Software Inc., which is now known as Oracle Corporation, introduced the Oracle V2 in June 1979, which is the first implementation of SQL language. This Oracle V2 version operates on VAX computers.

SQL (Structured Query Language) is a database computer language designed for managing data in relational database management systems (RDBMS). SQL is not case sensitive. SQL is a standardized computer language that was originally developed by IBM for querying, altering and defining relational databases, using declarative statements. SQL is a language to operate databases; it includes database creation, deletion, fetching rows, modifying rows, etc. SQL is an ANSI (American National Standards Institute) standard language, but there are many different versions of the SQL language. The standard SQL commands to interact with relational databases are CREATE, SELECT, INSERT, UPDATE, DELETE and DROP.

Purpose of SQL:

- SQL is used to Create New Databases
- SQL is used to Create New Tables in a Database
- SQL is used to Insert records in a Database
- SQL is used to Update records in a Database
- SQL is used to Delete records in a Database
- SQL is used to Retrieve data from a Database
- SQL is used to execute queries against a Database

- SQL can set permissions on tables, procedures and views
- SQL is used to Create stored procedures in a Database
- SQL is used to Create views in a Database

Who should learn SQL?

i. Database Developers:

- Design and deploy Database table structures, forms, reports and queries etc...

ii. Database Administrators (DBA):

- Keeping databases up to date and managing database access
- Writing Reports, documentation, and operating manuals

iii. Database Testers:

- Verify Data Integrity
- Verify Data Manipulations (Add, Update, and Delete)
- Verify Data comparisons

What are the subsets of SQL?

SQL Commands can be classified into groups based on their nature, they are,

- **Data Definition Language [DDL]**
- **Data Manipulation Language [DML]**
- **Data Control Language [DCL]**

Popular Database Management Systems:

1. Oracle:

Oracle database is a relational database management system (RDBMS) developed by Oracle Corporation. Important Oracle editions are,

- i) **Enterprise Edition:** Offers all features, including superior performance and security.
- ii) **Standard Edition:** Contains base functionality for users.
- iii) **Express Edition:** The lightweight, free and limited Windows and Linux edition
- iv) **Oracle Lite:** For mobile devices

2. Microsoft SQL Server:

Microsoft SQL Server is a relational database management system developed by

Microsoft. Important Microsoft SQL Server editions are,

- i) **Datacenter**: SQL Server 2008 R2 Datacenter is a full-featured edition of SQL Server.
- ii) **Enterprise**: SQL Server Enterprise Edition includes both the core database engine and add-on services.
- iii) **Standard**: SQL Server Standard edition includes the core database engine, along with the stand-alone services.
- iv) **Web**: SQL Server Web Edition is for Web hosting.
- v) **Workgroup**: SQL Server Workgroup Edition includes the core database functionality only.

3. MySQL:

MySQL is an Open-Source Relational SQL database management system used for developing web-based software applications.

Important MySQL Editions are,

- i) **Standard Edition**: Standard Edition enables us to deliver high-performance and scalable Online Transaction Processing (OLTP) applications.
- ii) **Enterprise Edition**: Enterprise Edition includes the most comprehensive set of advanced features and management tools.
- iii) **Cluster Carrier Grade Edition**: Cluster enables users to meet the database challenges of next generation web, cloud, and communications services with uncompromising scalability, uptime and agility.

4. PostgreSQL:

PostgreSQL is a powerful, open-source database management system. It runs on all major operating systems, including Linux, UNIX, Mac OS X, Solaris, and MS Windows.

5. MS Access:

Microsoft Access is bundled as part of the Microsoft Office suite. It is only available on the PC version. It is a desktop database system because its functions are intended to be run from a single computer.

What can SQL do?

- ❖ Allows users to define the data in a database and manipulate that data.
- ❖ Allows users to create and drop databases and tables.
- ❖ Allows users to create view, stored procedure, functions in a database.
- ❖ Allows users to set permissions on tables, procedures and views.
- ❖ Allows users to access data in the relational database management systems.

- ❖ Allows users to describe the data.
- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database

Structured Query Language (SQL)

- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

Advantages of SQL:

- > High speed
- > No coding needed
- > Well defined standards
- > Portability
- > Interactive language
- > Multiple data view

Stepwise Procedure for Installing Oracle 11g Express Edition:

Oracle Database (known as Oracle RDBMS) is a Database Management System produced and marketed by Oracle Corporation. The Most Fundamental and common usage of Oracle Database is to store a Pre-Defined type of Data. It supports the Structured Query language (SQL) to Manage and Manipulate the Data that it has. It is one of the most Reliable and highly used Relational Database Engines.

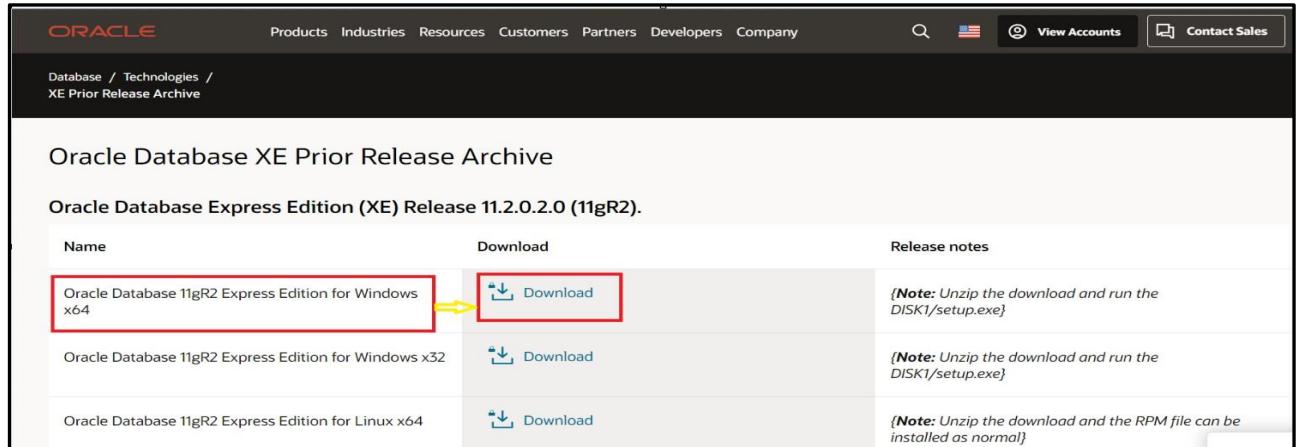
There are many versions of Oracle Database like Oracle Database 10g, Oracle Database 11g, Oracle Database 12c, Oracle Database 19c, Oracle Database 21c etc. from which Oracle 21c is the Latest Version. In this article, we will learn how to Install version 11g on Windows.

Oracle Database 11g is an enterprise database by Oracle. It delivers industry-leading performance, scalability, security, and reliability on a choice of clustered or single servers running Windows, Linux, and UNIX along with comprehensive features to easily manage the most demanding transaction processing, business intelligence, and content management applications.

Downloading the Installation Files:

Step 1: Click on "<https://www.oracle.com/database/technologies/xe-prior-release-downloads.html>" link or Go to oracle.com and Click on Options Menu and Click the Download Button.

Step 2: After you open the provided link, click on [Oracle Database 11gR2 Express Edition for Windows x64] 'Download' as shown in the image below.



The screenshot shows the Oracle Database XE Prior Release Archive page. It lists three download options:

- Oracle Database 11gR2 Express Edition for Windows x64: The 'Download' button is highlighted with a red box and a yellow arrow pointing to it.
- Oracle Database 11gR2 Express Edition for Windows x32: The 'Download' button is shown.
- Oracle Database 11gR2 Express Edition for Linux x64: The 'Download' button is shown.

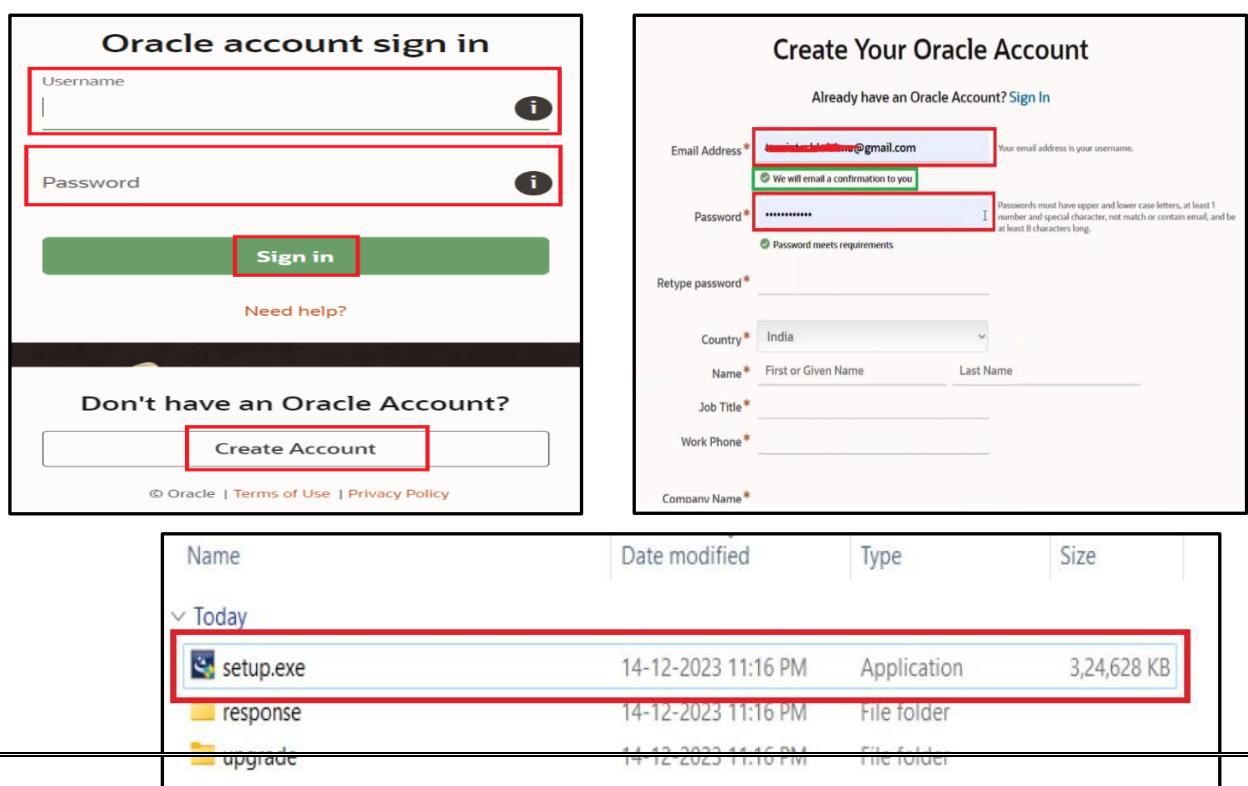
Release notes for the Windows x64 download state: '{Note: Unzip the download and run the DISK1/setup.exe}'.

Step 3: After Clicking the Download Button, the page will be directed to Login Screen where you'll need to Sign In in Oracle Account.



The screenshot shows a modal dialog. It contains the text: "You must accept the [Oracle License Agreement](#) to download this software." Below this is a checkbox labeled "I reviewed and accept the Oracle License Agreement" which is checked. The text "Required" is below the checkbox. At the bottom of the dialog is a button labeled "Download OracleXE112_Win64.zip" with a download icon.

Step 4: If you don't have one, then you must Sign Up, because without that you won't be able to download the Files.



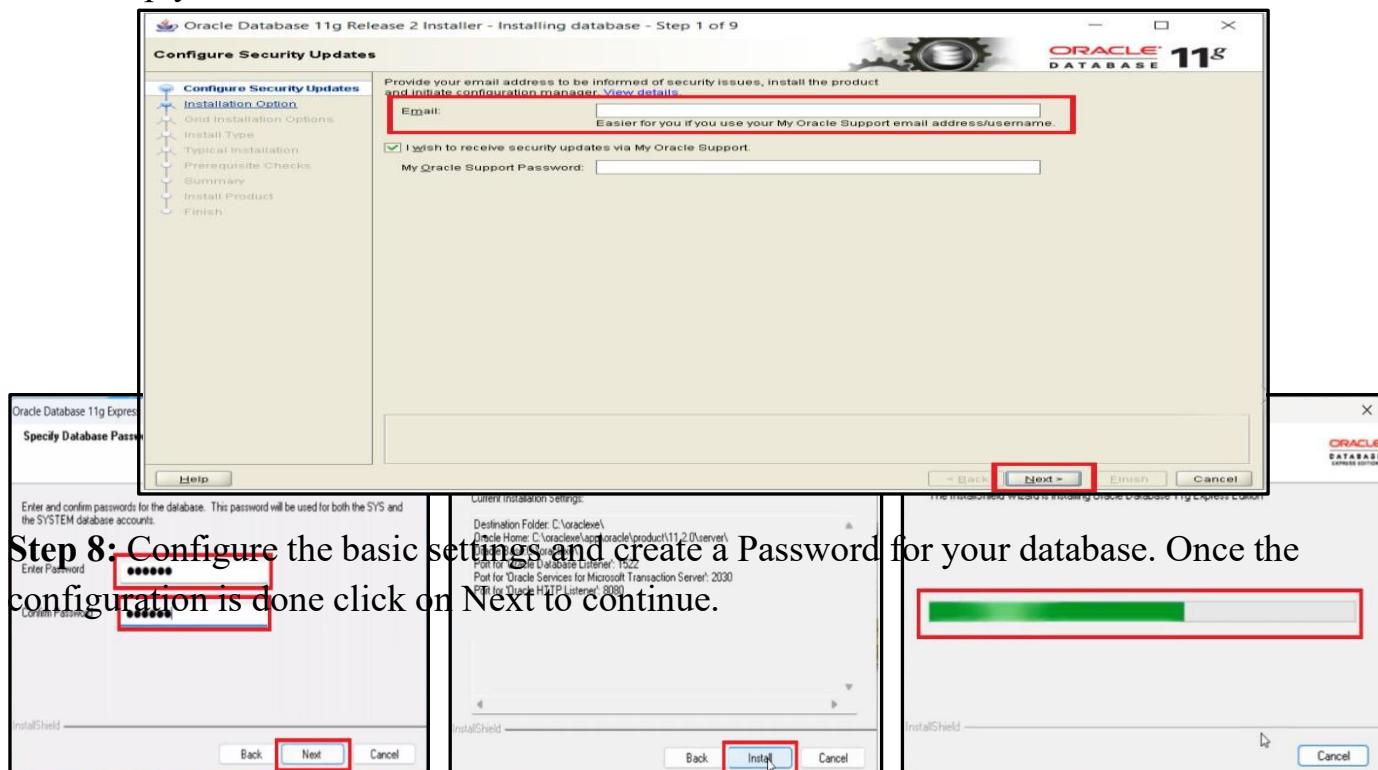
The image contains three screenshots:

- Oracle account sign in:** A form with fields for "Username" and "Password", and a "Sign in" button.
- Create Your Oracle Account:** A form for creating an account with fields for "Email Address", "Password", "Retype password", "Country", "Name", "Job Title", "Work Phone", and "Company Name". It includes a note about email being used as the username and password requirements.
- File List:** A table showing files in a folder. The table has columns: Name, Date modified, Type, and Size. It lists three files: "setup.exe", "response", and "upgrade".

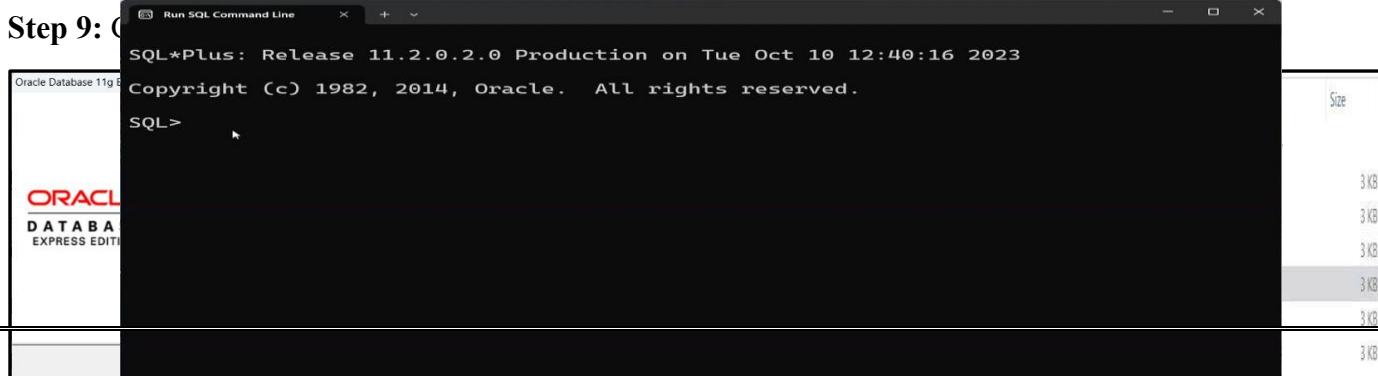
Step 5: Repeat the same steps and download them. After downloading the file Successfully, you'll find File in Downloads Folder where File will be in Compressed Form, so you'll need to Extract them.

Step 6: Now click on the setup.exe file to start the installation. Oracle quickly does the prerequisite check and launches the Universal Installer.

Step 7: Provide your Email Address to receive all the Notifications and News Alerts from Oracle. In case you don't want to receive any Alerts from Oracle, then Simply leave these fields empty and click on Next to move ahead.



Step 8: Configure the basic settings and create a Password for your database. Once the configuration is done click on Next to continue.



Create a New Oracle User and Grant Privileges:

First, you'll need login as system or sysdba. Once you're in, the basic create user command is

```
SQL> connect / as sysdba;
```

Connected.

The **CREATE USER** statement allows you to create a new database user which you can use to log in to the Oracle database. we are creating a new local user under the username. The user will be required to enter the password to log into the system. The basic syntax of the CREATE USER statement is as follows:

```
SQL> create user username identified by password;
```

So, to create the user krishdb with the password krishdb.

```
SQL> create user krishdb identified by krishdb;
```

User created.

Now you've got your user. The next step is to connect to it. But try to do so and you'll hit:

```
SQL> connect krishdb
```

Enter password: **krishdb**

Error: ORA-01045: user DATA_OWNER lacks CREATE SESSION privilege; logon denied.

The problem is you haven't given the user any permissions! By default, a database user has no privileges. Not even to connect.

Granting User Privileges:

We can now begin adding privileges to the account using the GRANT statement. GRANT is a very powerful statement with many possible options, but the core functionality is to manage the privileges of both users and roles throughout the database. You give permissions with the grant command.

For system privileges this takes the form:

```
SQL> connect / as sysdba;
```

Connected.

```
SQL> grant all privileges to krishdb;
```

Grant succeeded.

```
SQL> connect  
krishdb Enter
```

password: krishdb

Connected.

SQL> show user; [To show the User Name]

USER is "krishdb"

SQL> select user from dual; USER

krishdb

SQL> select * from all_users;

USERNAME	USER_ID	CREATED
----------	---------	---------

XS\$NULL	2147483638	29-MAY-14
----------	------------	-----------

krishdb	49	17-OCT-23
---------	----	-----------

APEX_040000	47	29-MAY-14
-------------	----	-----------

APEX_PUBLIC_USER	45	29-MAY-14
------------------	----	-----------

SQL>select * from v\$version; [Find Version]

o/p:

BANNER

Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production PL/SQL

Release 11.2.0.1.0 - Production

CORE 11.2.0.1.0 Production

TNS for 64-bit Windows: Version 11.2.0.1.0 - Production NLSRTL Version

11.2.0.1.0 – Production

SQL DATATYPES

Datatypes in Oracle:

Oracle, like many relational databases, offers a wide range of datatypes to store different kinds of data. To ensure the accuracy and efficiency of your queries, it is important to have a good understanding of the different datatypes in Oracle and to select the most suitable one for each column. Each value manipulated by Oracle Database has a data type. The data type of a value associates a fixed set of properties with the value. These properties cause Oracle to treat values of one data type differently from values of another. For example, you can add values of NUMBER data type, but not values of CHAR data type. When you create a table or cluster, you must specify a data type for each of its columns. A data type also specifies the possible values for that type, the operations that can be performed on that type and the way the values of that type are stored. SQL Datatype is used to define the values that a column can contain. Every column is required to have a name and datatype in the database table.

1. Character Datatype:

In Oracle, character data types store strings of letters, symbols, and numbers specified in the fixed-length field. It means that when you add a shorter value to the column than the column length, Oracle pads the string with spaces. When you add a longer value than the column length, the error occurs.

i. Char datatype:

It is used to store character data within the predefined length. It can be stored up to 2000 bytes or characters. Default and minimum size are 1 byte. Stores a fixed-length string value. CHAR data type is more appropriate to use where ever fixed the size of data will be handled.

ii. VARCHAR2 Datatype:

This data type stores the string, but the length of the string is not fixed. It is used to store variable string data within the predefined length. The size restriction for this data type is 1-4000 bytes for table column size and 1-32767 bytes for variables. The size is defined for each variable at the time of variable declaration. A VARIABLE length string (can contain letters, numbers, and special characters). It is always good to use VARCHAR2 instead of CHAR data type to optimize the memory usage. You must specify size for VARCHAR2.

Example: manager VARCHAR2(10) = 'oracle';

The above declaration statement declared the variable 'manager' of VARCHAR2 data type with the maximum size of 10 and assigned the value 'oracle' which is of 6 bytes. Oracle will allocate memory of only 6 bytes in this case.

iii. VARCHAR Datatype:

It is the same as VARCHAR2(size). You can also use VARCHAR(size), but it is suggested to use VARCHAR2(size). This is synonymous with the VARCHAR2 data type. It is always a good practice to use VARCHAR2 instead of VARCHAR to avoid behavioral changes.

Example: manager VARCHAR (10) = 'oracle';

The above declaration statement declared the variable 'manager' of VARCHAR data type with the maximum size of 10 and assigned the value 'oracle' which is of 6 bytes. Oracle will allocate memory of only 6 bytes in this case. (Similar to VARCHAR2).

iv. **LONG and LONGRAW Datatype:**

This data type is used to store large text or raw data up to the maximum size of 2GB.

These are mainly used in the data dictionary. LONG data type is used to store character set data, while LONG RAW is used to store data in binary format. whereas LONG works only on data that can be stored using character set.

Example: Large_text LONG;

The above declaration statement declares the variable 'Large_text' of LONG data type.

Note: Using LONG data type is not recommended by Oracle.

In Oracle, the LONG and LONG RAW data types are deprecated and should not be used for new development. These data types are used for handling large blocks of character data (LONG) and binary data (LONG RAW). Instead, you should use **CLOB** (Character Large Object) for character data and **BLOB (Binary Large Object)** for binary data. BLOBs are used to store binary information, such as images, while CLOBs are used to store character information. BLOBs and CLOBs can store up to 4 Gigabytes of data. A CLOB value can be up to 2,147,483,647 characters long.

Example: SQL> CREATE TABLE student3

(name clob); Table created.

SQL> desc student3;

Name	Null?	Type
------	-------	------

NAME	CLOB
------	------

2. **NUMBER (Numeric) Data Type:**

This data type stores fixed or floating -point numbers up to 38 digits of precision. This data type is used to work with fields which will contain only number data. The variable can be declared either with precision and decimal digit details or without this information. Values need not enclose within quotes while assigning for this data type. Oracle supports the

NUMBER data type that can store fixed or floating numeric values. A NUMBER value requires from 1 to 22 bytes.

Number having **precision** p and **scale** s. The precision represents the total number of digits that can be stored, and the scale represents the number of digits to the right of the decimal point.

Range of p: From 1 to 38.

Ranges of s: From -84 to 127.

Both precision and scale are in decimal digits.

Example:

A. NUMBER (8,2);

B. NUMBER (8);

In the above, the first declaration declares the variable 'A' is of number data type with total precision 8 and decimal digits 2. The second declaration declares the variable 'B' is of number data type with total precision 8 and no decimal digits.

3. DATE Data Type:

The DATE data type stores date and time values in a table column. It includes a year, a month, a day, hours, minutes, and seconds. This data type stores the values in date format, as date, month, and year. Whenever a variable is defined with DATE data type along with the date it can hold time information and by default time information is set to 12:00:00 if not specified. Values need to enclose within quotes while assigning for this data type. This data type contains the datetime fields YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND. It does not have fractional seconds or a time zone.

SQL> SELECT SYSDATE FROM DUAL;

SYSDATE

05-AUG-23

TIMESTAMP Data Type:

The TIMESTAMP data type allows you to store date and time data including year, month, day, hour, minute and second. In addition, it stores the fractional seconds, which is not stored by the DATE data type.

To define a TIMESTAMP column, you use the following syntax:

SQL> create table ss1(dob timestamp);

Table created.

In this example, the started_at column is a TIMESTAMP column with fractional seconds precision sets to microsecond. To specify TIMESTAMP literals, you use the following format:

TIMESTAMP 'YYYY-MM-DD HH24:MI:

SS. FF' TIMESTAMP '10-feb-2023

10:15:50 am'

SQL> insert into ss1 values ('10-feb-2023

10:15:50 am'); 1 row created.

SQL COMMANDS

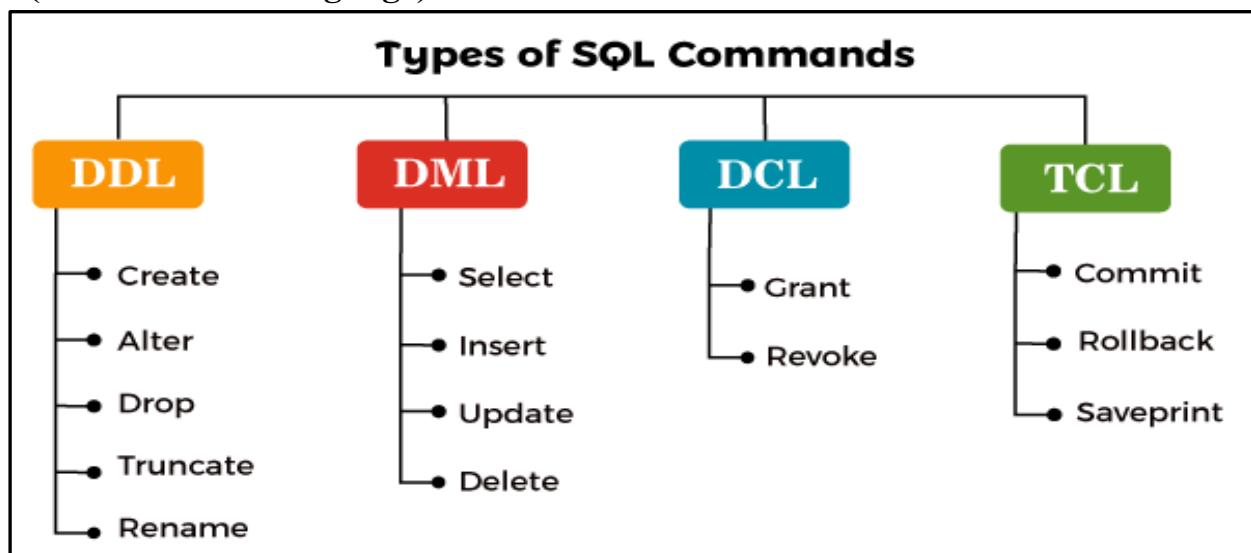
SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions, and queries of data.

SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, set permission for users.

Types of SQL Commands:

- There are five types of SQL commands: DDL, DML, DCL and TCL.

DDL (Data Definition Language). **DML (Data Manipulation Language).** **TCL (Transaction control Language).**
DCL (Data Control Language).



1. DDL [Data Definition Language]:

DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.

- All the command of DDL is auto-committed that means it permanently save all the changes in the database.
- Here are some commands that come under DDL:

CREATE

- ✚ **ALTER**
- ✚ **DROP**
- ✚ **TRUNCATE**
- ✚ **RENAME**

Command	Description
CREATE	Creates a new table, a view of a table, or other object in the database.
ALTER	Modifies an existing database object, such as a table.
DROP	Deletes an entire table, a view of a table or other objects in the
TRUNCATE	It is used to delete all the rows from the table and free the space containing the table.
Rename	It is used to change the name of the data base table.

a) Create Table Command:

It is a data definition language which is used to create any database objects. CREATE It is used to create a new table in the database.

Syntax: Create table < Table name> (col name 1 datatype (width), colname 2 datatype (width)...);

Ex: SQL > create table student (Sno number (10), Sname varchar2 (20),....);

- > The column names should be unique in a table.
- > Proper datatype should be specified with the width (size).
- > While giving the table name, we should follow the rules given below

- We can use alphabets, numbers & underscore to name a table.
- The first letter should be an alphabet, while naming a table.
- The table name can be up to 30 characters length.
- Two different tables cannot have the same name.
- Spaces, hyphens & other symbols are not allowed to name a table.

b) Alter Table Command:

This is used to change the definition of a table. Alter table command is also used for the following. It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.

- ❖ User can add a new column.
- ❖ You can modify (increase/ decrease) the width of a datatype.
- ❖ User can change the datatype.
- ❖ User can rename particular column name.
- ❖ User can drop particular column.

Alter table to add column:

Syntax: SQL> alter table < table name> add(colname datatype (width));

Ex: SQL> alter table student add (group varchar (6));

NOTE: To add a new column for the existing table.

To modify size (OR) column.

Syntax: SQL> alter table <table name> modify (column datatype (width));

Ex: (i)SQL> alter table student modify (sgroup varchar(10));

SQL> alter table student modify (sgroup char(10));

To Rename column:

Syntax: alter table tablename rename existing columnname to new columnname; **Ex:** alter table student rename column sid to StudentId;

To drop column:

Syntax: ALTER TABLE tablename DROP column columnname;

Ex: SQL> ALTER TABLE student DROP column studentid;

c) Drop Table Command:

It is a DDL Command, which is used to remove (or) delete the database table. It is used to delete both the structure and record stored in the table.

Syntax: Drop table <table name>;

SQL> Drop table student;

d) TRUNCATE:

It is used to delete all the rows from the table and free the space containing the table.

Syntax: TRUNCATE TABLE table_name;

Example: TRUNCATE TABLE EMPLOYEE

e) Rename Table Command :

It is a DDL Command, which is used to change the name of the data base table.

Syntax: rename Old tablename to new tablename

Example: SQL > rename student tostudent1;

Difference between DELETE, DROP and TRUNCATE Commands in SQL:

DELETE Command	DROP Command	TRUNCATE Command
The DELETE command is Data Manipulation Language (DML) Command.	The DROP command is Data Definition Language (DDL) Command.	The TRUNCATE command is a Data Definition Language (DDL) Command.

The DELETE command deletes one or more existing records from the table in the database.	The DROP Command drops the complete table from the database.	The TRUNCATE Command deletes all the rows from the existing table, leaving the row with the column names.
We can restore any deleted row or multiple rows from the database using the ROLLBACK command.	We cannot get the complete table deleted from the database using the ROLLBACK command.	We cannot restore all the deleted rows from the database using the ROLLBACK command.
The DELETE command does not free the allocated space of the table from memory.	The DROP command removes the space allocated for the table from memory.	The TRUNCATE command does not free the space allocated for the table from memory.
The Integrity Constraints remain the same in the DELETE command.	The Integrity Constraints get removed for the DROP command.	The Integrity Constraints will not get removed from
DELETE FROM table_name WHERE condition;	DROP TABLE table_name;	TRUNCATE TABLE table_name;
DELETE FROM Student WHERE Roll_No = 101;	DROP TABLE Student;	TRUNCATE TABLE Student;

2. DML commands:

DML commands are used to modify the database. It is responsible for all form of CHANGES in the database. The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback. They are used to

manipulate the data in a table .

It is divided into 4 types:

(a) Insert command (b) Update command (c) Select command (d) Delete command

a) Insert command:

The INSERT statement is a SQL query. It is used to insert data into the row of a table. It is used to add one (OR) more row into a table. The values must be entered in the same order as they are defined in the database table. The values are separated by commas (,). The datatypes are char, varchar2, long, date, raw-are enclosed in a single quotation marks.

□ Inserting values for all columns:

Syntax: insert into tablename values(Column1value,Column2value,Column3value,..);

Ex: insert into employee values(1001,'krishna',50000,'05-may-2020');

o/p: 1 row created.

□ Inserting values to specific columns:

Syntax: insert into tablename(columnname,columnname) values(columndata, columndata);

Ex: SQL> insert into employee (eid,ename) values(1003,'Supriya');

o/p: 1 row created.

Note: The other columns are assigned with "Null values".

□ Inserting values for all columns at a time:

We can speed up input by substitution variables.

Syntax: insert into tablename values(&column1name,&column2name,..);

Column1value:

Column2value:

Ex: SQL> insert into employee
values(&eid,'&ename',&salery,'&doj'); Enter value for eid: 1005
Enter value for ename:
archana Enter value for
salery: 61454

Enter value for DOJ: 25-jan-1999

1 row created.

Note: When the command is run values are promoted for every time. Use / slash.

b) Update command:

It is used to update rows in a table. It consists of a "set clause " and optional "where clause".

We can update a single column (OR) more than one column.

Syntax: SQL>update <table name> set colname -values . . . where <condition>;

Ex: SQL> update emp set da = sal * 10/100;

SQL> update emp set da = sal * 20/100 where ename = 'smith';

c) Select command:

It is used to fetch records from the tables and views stored in the database. It allows you to retrieve specific columns, filter rows based on conditions, and perform various operations on the data.

To select all fields from a table:

Syntax: SELECT * FROM

table_name; Example: SELECT *
from students;

To select specific fields from a table:

Syntax: SELECT field1,field2 FROM
table_name; Example: SELECT name, age
FROM students

To select specific fields from a table using conditions:

Syntax: SELECT field1,field2 FROM table_name WHERE
conditions; Example: SELECT name, age FROM students
WHERE age>10 ORDER BY name;

To select specific fields from multiple tables:

Syntax: SELECT expressions FROM table1 INNER JOIN table2 ON join_predicate;

Example: SELECT students.name, teachers.subject FROM students

INNER JOIN teachers
ON students.student_id =

student_id ;
ORDER BY name;

d) Delete command:

It is a DML command which is used to delete one (OR) more rows from a table.

Syntax: SQL>delete from <table name> where <condition>;

SQL> delete from student;

It will delete the all rows from a student table.

SQL> delete from student where Sno=101;

NOTE: Delete command removes all rows but the structure of a table remains unchanged.

3. DCL Commands:

Data control language is used to control the data among a number of users. It is divided into two types:

(a) Grant command

(b) Revoke command

a) Grant command:

It is a DCL command which gives permission to another user. **Syntax:** SQL> grant privileges on <table name> to user name; **Ex:** SQL> grant select on emp to RJC;

NOTE: Privileges are select, insert, update, delete, alter,etc.

b) Revoke command:

It is used to withdraw a privilege.

Syntax: SQL> revoke privileges on <table name> from user name;

Ex: SQL> revoke select on emp from rjc.

4. TCL commands:

This language is used to transaction process, it consists of commit & Roll back commands.

a) Commit command:

It is used to make database changes permanently.

Syntax: SQL> commit

b) Roll back command:

It is used to cancel the committed transactions.

Syntax: SQL>Roll back

Here, we don't mention table name because we use it on the present working table.

c) Savepoint Command:

The SAVEPOINT in Oracle is used for dividing (or) breaking a transaction into multiple units so that the user has a chance of roll backing the transaction up to a specified point. That

means using Save Point we can roll back a part of a transaction instead of the entire transaction. Whenever a user creates SAVEPOINT within the transaction then internally system is allocating a special memory for a SAVEPOINT and storing the transaction information which we want to roll back (cancel).

Syntax: SAVEPOINT SAVEPOINT_NAME;

Example: SAVEPOINT save1;
ROLLBACK TO
SAVEPOINT_NAME;
ROLLBACK TO save1;

```
SET TRANSACTION READ WRITE;
```

```
SAVEPOINT SavePoint1;
INSERT INTO Product VALUES(1005, 'Product-5', 5000, 500);
INSERT INTO Product VALUES(1006, 'Product-6', 6000, 600);
```

```
SAVEPOINT SavePoint2;
INSERT INTO Product VALUES(1007, 'Product-7', 7000, 700);
INSERT INTO Product VALUES(1008, 'Product-8', 8000, 800);
```

```
SAVEPOINT SavePoint3;
INSERT INTO Product VALUES(1009, 'Product-9', 9000, 900);
INSERT INTO Product VALUES(1010, 'Product-10', 10000, 1000);
```

CASE1: ROLLBACK TO SavePoint1;

CASE2: ROLLBACK TO SavePoint2;

CASE3: ROLLBACK TO SavePoint3;

```
SET TRANSACTION READ WRITE;
```

```
SAVEPOINT SavePoint1;
```

```
INSERT INTO Product VALUES(1005, 'Product-5', 5000, 500);
INSERT INTO Product VALUES(1006, 'Product-6', 6000, 600);
```

```
SAVEPOINT SavePoint2;
```

```
INSERT INTO Product VALUES(1007, 'Product-7', 7000, 700);
INSERT INTO Product VALUES(1008, 'Product-8', 8000, 800);
```

```
SAVEPOINT SavePoint3;
```

```
INSERT INTO Product VALUES(1009, 'Product-9', 9000, 900);
INSERT INTO Product VALUES(1010, 'Product-10', 10000, 1000);
```

Module-4: INTEGRITY CONSTRAINTS

An integrity constraint is a mechanism used by oracle to prevent the invalid data entry into the database table.

SQL CONSTRAINTS

Constraints are the rules enforced on data columns on table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

Constraints could be column level or table level. Column level constraints are applied only to one column, whereas table level constraints are applied to the whole table.

Following are commonly used constraints available in SQL:

NOT NULL Constraint: Ensures that a column cannot have NULL value.

DEFAULT Constraint: Provides a default value for a column when none is specified.

UNIQUE Constraint: Ensures that all values in a column are different.

PRIMARY Key Constraint: Uniquely identified each rows/records in a database table.

FOREIGN Key Constraint: Uniquely identified a rows/ records in any another database table.

CHECK Constraint: The CHECK constraint ensures that all values in a column satisfy certain conditions.

INDEX Constraint: Use to create and retrieve data from the database very quickly.

NOT NULL Constraint: By default, a column can hold NULL values. If you do not want a column to have a NULL value, then you need to define such constraint on this column specifying that NULL is now not allowed for that column. A NULL is not the same as no data, rather, it represents unknown data.

For Example, the following SQL creates a new table called CUSTOMERS and adds five columns, three of which, ID and NAME and AGE, specify not to accept NULLs:

```
CREATE TABLE CUSTOMERS (
    ID      INT          NOT NULL,
    NAME    VARCHAR (20)  NOT NULL,
    AGE     INT          NOT NULL,
    ADDRESS  CHAR (25) ,
    SALARY   DECIMAL (18, 2),
    PRIMARY KEY (ID)
);
```

If CUSTOMERS table has already been created, then to add a NOT NULL constraint to SALARY column in Oracle and MySQL, you would write a statement similar to the following:

```
ALTER TABLE CUSTOMERS
  MODIFY SALARY  DECIMAL (18, 2) NOT NULL;
```

Drop Not Null Constraint:

To drop a Not Null constraint, use the following SQL:

```
SQL> ALTER TABLE customers MODIFY age NULL;
```

DEFAULT Constraint:

The DEFAULT constraint provides a default value to a column when the INSERT INTO statement does not provide a specific value.

For example, the following SQL creates a new table called CUSTOMERS and adds five columns. Here, SALARY column is set to 5000.00 by default, so in case INSERT INTO statement does not provide a value for this column. then by default this column would be set to 5000.00.

Create:

```
SQL> create table kr1 (eid number (5) not null, ename varchar2(10), sal number (5) default 5000);
```

Table created.

If CUSTOMERS table has already been created, then to add a DEFAULT constraint to SALARY column, you would write a statement similar to the following:

Alter:

```
SQL> ALTER TABLE kr1 modify (ename varchar2(10) default 'krishna');
```

Table altered.

Drop Default Constraint:

To drop a DEFAULT constraint, use the following SQL:

```
SQL>ALTER TABLE cust1 MODIFY salary DEFAULT NULL;
```

Table altered.

UNIQUE Constraint:

The UNIQUE Constraint prevents two records from having identical values in a particular column. In the CUSTOMERS table, for example, you might want to prevent two or more people from having identical age.

For example, the following SQL creates a new table called CUSTOMERS and adds five columns. Here, AGE column is set to UNIQUE, so that you cannot have two records with same age:

```
CREATE TABLE CUSTOMERS (
    ID      INT          NOT NULL,
    NAME    VARCHAR (20)  NOT NULL,
    AGE     INT          NOT NULL UNIQUE,
    ADDRESS CHAR (25),
    SALARY   DECIMAL (18, 2),
    PRIMARY KEY (ID)
);
```

If CUSTOMERS table has already been created, then to add a UNIQUE constraint to AGE column, you would write a statement similar to the following:

```
ALTER TABLE CUSTOMERS
MODIFY AGE INT NOT NULL UNIQUE;
```

DROP a UNIQUE Constraint:

To drop a UNIQUE constraint, use the following SQL:

```
SQL> alter table customers drop constraint age;
```

CHECK Constraint:

The CHECK Constraint enables a condition to check the value being entered into a record. If the condition evaluates to false, the record violates the constraint and isn't entered into the table.

For example, the following SQL creates a new table called CUSTOMERS and adds five columns. Here, we add a CHECK with AGE column, so that you cannot have any CUSTOMER below 18 years:

Create a CHECK Constraint:

```
CREATE TABLE CUSTOMERS (
    ID      INT          NOT NULL,
    NAME    VARCHAR (20)  NOT NULL,
    AGE     INT          NOT NULL CHECK (AGE >= 18),
    ADDRESS CHAR (25) ,
    SALARY   DECIMAL (18, 2),
    PRIMARY KEY (ID)
);
```

If CUSTOMERS table has already been created, then to add a CHECK constraint to AGE column, you would write a statement similar to the following:

Alter a CHECK Constraint:

```
SQL> alter table kr1 modify (sal number (5) check (sal >=10000));
```

Table altered.

You can also use following syntax, which supports naming the constraint in multiple columns as well:

```
SQL> ALTER TABLE kr1 ADD CONSTRAINT salconstraint CHECK (sal >= 500);
```

Table altered.

```
ALTER TABLE CUSTOMERS
  ADD CONSTRAINT myCheckConstraint CHECK(AGE >= 18) ;
```

Check Created Constraints:

```
SQL> SELECT CONSTRAINT_NAME, CONSTRAINT_TYPE FROM
  USER_CONSTRAINTS WHERE TABLE_NAME = 'KR1';
```

```
CONSTRAINT_NAME  C
```

```
-----
```

```
SYS_C007042      C
```

```
SALCONSTRAINT
```

```
      C
```

DROP a CHECK Constraint:

To drop a CHECK constraint, use the following SQL.

```
SQL> ALTER TABLE kr1 DROP CONSTRAINT "SYS_C007044";
```

Table altered.

```
ALTER TABLE CUSTOMERS
  DROP CONSTRAINT myCheckConstraint;
```

PRIMARY KEY CONSTRAINTS:

- A primary key constrains of a combination of NOT NULL and UNIQUE.
- A primary key constrains will not accept null values as well as duplicate values.
- Primary key column is used to uniquely every row in a table.
- A table can have only one primary key.
- Primary key constraints can be created at column level or table level.

Create primary key constraint at column level:

Ex: Create table student6(Sno number(3) PRIMARY KEY, Sname varchar2(10), Marks number(3)); Insert into student6 values(101,'Arun',50);
Insert into student6 values(102,'Arun',50);

Insert into student6 values(101,Arun,50); ><
Insert into student6 values(NULL,'Arun',50);

Insert into student6 values(103,'Arun',50);

Create Primary key constraint at table level:

Ex:Create table student7(Sno number(3), Sname varchar2(10), Marks number(3) PRIMARY KEY(Sno));

DATA INTEGRITY

The overall precision, completeness, and continuity of data is known as data integrity. Data integrity also applies to the data's protection and security in terms of regulatory enforcement, such as GDPR [General Data Protection Regulation] compliance. It is kept up to date by a set of procedures, guidelines, and specifications that were put in place during the design phase.

Database integrity defines the validity and consistency of stored information. Integrity is generally defined in terms of constraints, which are consistency rules that the database is not allowed to violate. Constraints can apply to each attribute or they can apply to relationships between tables. Integrity constraints provide that changes (update deletion, insertion) made to the database by authorized users do not result in a loss of data consistency. Therefore, integrity constraints guard against accidental damage to the database.

Example:

Imagine this: A pharmaceutical company touts the safety of its new wonder drug. But when the FDA inspects the offshore production facility, work is halted immediately; important quality-control data is missing. Unfortunately, this real-life example of compromised data integrity is all too common. Problems with the accuracy and consistency of data can cause everything from minor hassles to significant business problems.



Data Integrity Risks: An assortment of factors can affect the integrity of the data stored in a database. A few examples include the following:

Human error: When individuals enter information incorrectly, duplicate or delete data, don't follow the appropriate protocols, or make mistakes during the implementation of procedures meant to safeguard information, data integrity is jeopardized.

Transfer errors: When data can't successfully transfer from one location in a database to

another, a transfer error has occurred.

Bugs and viruses: Spyware, malware, and viruses are pieces of software that can invade a computer and alter, delete, or steal data.

Compromised Hardware: Sudden computer or server crashes, and problems with how a computer or other device functions, are examples of significant failures and may be indications that your hardware is compromised. Compromised hardware may render data incorrect or incomplete, limit or eliminate access to data, or make information hard to use.

RISKS OF NOT HAVING DATA INTEGRITY

Loss in E-commerce



Not having data integrity leads to the wrong product being sent to the customer, which leads to a loss in e-commerce.

Financial Loss



It leads to financial loss, for example, a company overpays for a product because the data about the cost of the product is inaccurate.

Wrong Diagnosis



It can lead to severe consequences in the healthcare industry. For example, if a patient's medical records are inaccurate, it could lead to the wrong diagnosis and treatment being given to the patient.

Overpaying Insurance



It can lead to problems in the insurance industry. For example, if a company's data about the cost of repairs to a car is inaccurate, it could lead to the company overpaying for the repairs.

www.erp-information.com

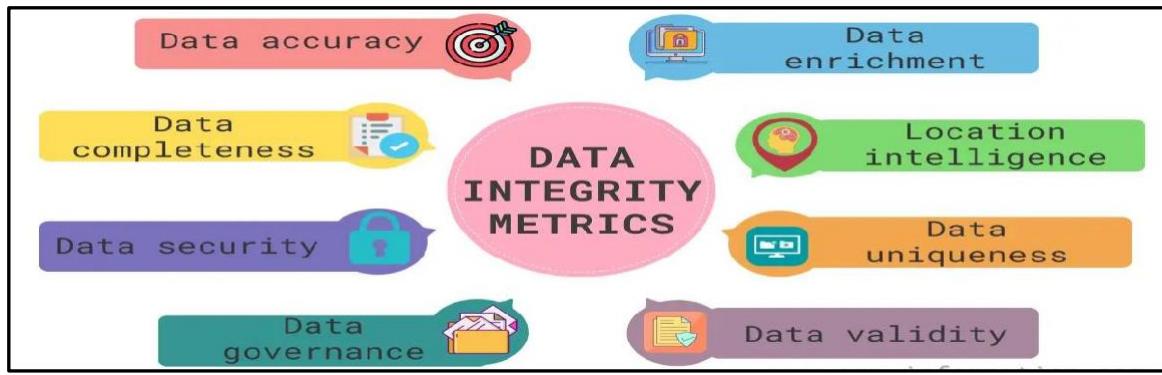
Benefits of Data Integrity:

- It provides the searchability and traceability of data to its source.
- With data integrity, we can easily search and track data.
- It helps to Get rid of redundant, inaccurate, or outdated data.
- It avoids the misclassification or improper storage of essential data.
- Data integrity helps ensure data security by ensuring data is safe from unauthorized access.
- Data integrity enables data governance, which is critical for managing data.
- It also helps to ensure data validity, which is vital for ensuring data is not corrupted.

Data Integrity Key Metrics [Characteristics]: Data integrity is comprised of common core characteristics:

- **Data accuracy:** How close the data is to the real-world data set. for example, a data accuracy of 95% means that the data is very close to the actual data set.
- **Data completeness:** How much data is included in a data set.
- **Data security:** Ensuring that data is safe from unauthorized access.
- **Data governance:** Ensuring that data is managed to meet the organization's needs.
- **Data validity:** Ensuring data is valid (i.e., not corrupted) by checking for errors.
- **Data uniqueness:** Make sure data is unique (i.e., no duplicates).
- **Location intelligence:** With location insight and analytics, make data more actionable by providing a layer of richness and complexity.

- **Data enrichment:** By adding data from external sources to internal data, you can give it more context, nuance, and significance. Adding business, consumer, or location details enhances your data's completeness and context by giving you a more comprehensive and contextualized perspective.



Different Kinds of Data Integrity (or) Types of Data Integrity:

Physical and logical data integrity are the two forms of data integrity. Both are a collection of procedures and methods for maintaining data integrity in hierarchical and relational databases. Two types of data integrity need to be understood

1. Physical data integrity.

2. Logical data integrity.

1. Physical integrity:

Physical integrity is the overall protection of the wholeness of a data set as it is stored and retrieved. Protecting data against external factors, such as natural calamities, power outages, or hackers, falls under the domain of physical integrity. Moreover, human faults, storage attrition, and several other problems can make data operators unable to obtain information from a database. Physical integrity refers to the safeguarding of data's completeness and precision during storage and retrieval. Physical integrity is jeopardized when natural disasters occur, electricity goes out, or hackers interrupt database functions. Data processing administrators, device programmers, applications programmers, and internal auditors may be unable to obtain reliable data due to human error, storage degradation, and a variety of other issues.

2. Logical integrity:

Logical integrity allows data to remain unchanged as it is utilized in a relational database. Maintaining logical integrity helps protect from human error and malicious intervention. In a relational database, logical consistency ensures the data remains intact as it is used in various ways. Logical integrity, like physical integrity, defends data from human error and hackers, but in a different way. There are four different forms of logical consistency. Logical integrity keeps data unchanged as it's used in different ways in a relational database. Logical integrity protects data from human error and hackers as well, but in a much different way than physical integrity does.

Logical integrity constraints can be categorized into **Four Types**:

(a) Entity integrity:

It's a characteristic of relational databases, which store information in tables that can be connected and used in a number of ways. Entity integrity involves the creation of primary keys to identify data as distinct entities and ensure that no data is listed more than once or is null. This allows data to be linked to and enables its usage in a variety of ways. It depends on the making of primary keys or exclusive values that classify data items. The purpose is to ensure that data is not recorded multiple times (i.e., each data item is unique), and the table has no null fields. Entity integrity is a critical feature of a relational database that stores data in a tabular format, which can be interconnected and used in various ways.

(b) Referential integrity:

Referential integrity refers to the series of processes that make sure data is stored and used uniformly. Rules embedded into the database's structure about how foreign keys are used ensure that only appropriate changes, additions, or deletions of data occur. This allows for a consistent and meaningful combination of data sets across the database. Critically, referential integrity allows the ability to combine various tables within a relational database, facilitating uniform insertion and deletion practices. Rules may include constraints that eliminate the entry of duplicate data, guarantee that data entry is accurate, and/or disallow the entry of data that doesn't apply.

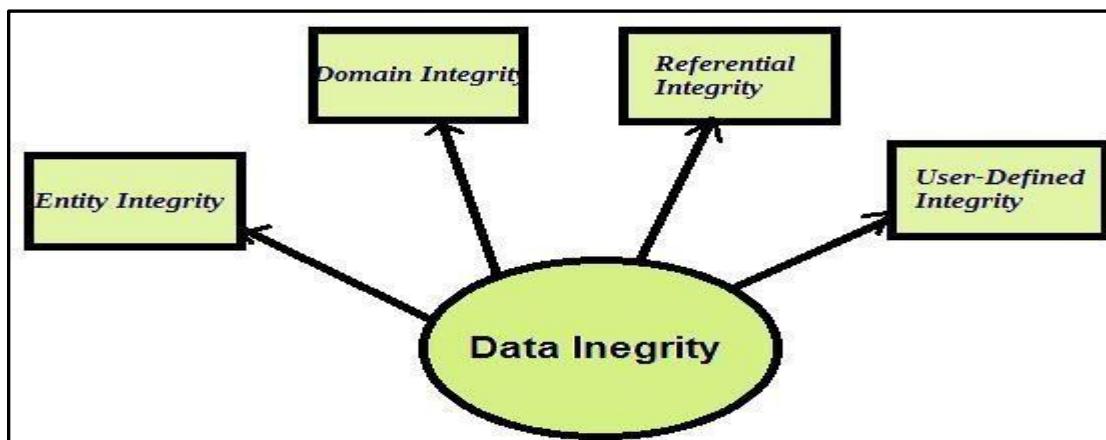
(c) Domain integrity:

Domain integrity is the collection of processes that ensure the accuracy of each piece of data in a domain. Domain integrity is a sequence of rules and procedures that provide all data items pertain to the correct

domains. For instance, if a user types a birth date in a street address area, the system will display an error message that will avoid the user from filling that field with wrong information. In this context, a domain is a set of acceptable values that a column is allowed to contain. It can include constraints and other measures that limit the format, type, and amount of data entered. **For example**, you might specify that a particular column can only contain integer values between 1 and 10.

(d) User-defined integrity:

User-defined integrity involves the rules and constraints created by the user to fit their particular needs. Sometimes entity, referential, and domain integrity aren't enough to safeguard data. Often, specific business rules must be taken into account and incorporated into data integrity measures. It comprises the rules defined by the operator to fulfil their specific requirements. Entity, referential, and domain integrity are not enough to refine and secure data. User-defined integrity provides rules and constraints that are created by the user in order to use data for their specific purpose. **For example**, this could be something like a “unique” constraint that ensures no two rows in a table have the same value for a particular column.



SQL DUAL TABLE

DUAL is a table automatically created by Oracle Database along with the data dictionary. DUAL is in the schema of the user SYS but is accessible by the name DUAL to all users. It has one column, DUMMY, defined to be VARCHAR2(1), and contains one row with a value X. Selecting from the DUAL table is useful for computing a constant expression with the SELECT statement. Because DUAL has only one row, the constant is returned only once. It's used for selecting data from system functions and calculations when you don't need any data from the database.

The DUAL table is a special one-row, one-column table present by default in Oracle and other database installations. In Oracle, the table has a single VARCHAR2(1) column called DUMMY that has a value of 'X'. It is suitable for use in selecting a pseudo column such as SYSDATE or USER. Oracle's SQL syntax requires the FROM clause but some queries don't require any tables - DUAL can be used in these cases.

SQL> SELECT * FROM

dual; D

--

-

X

SQL> SELECT 1+1 FROM

dual; 1+1

2

SQL> select 2+3 as Addition from

dual; ADDITION

5

SQL> select 2-3 as Sub from

dual; SUB

-1

SQL> SELECT 2 + 2 AS result FROM

dual; RESULT

4

SQL> SELECT 'Hello' || ',' || 'World!' AS message
FROM dual; MESSAGE

Hello, World!

SQL> SELECT 1 FROM

dual; 1

1

SQL> SELECT USER FROM

dual; USER

KRISHNA

SQL> SELECT SYSDATE FROM

dual; SYSDATE

16-JUN-23

SQL OPERATORS

An operator is a reserved word or a character used primarily in an SQL statement's WHERE clause to perform operation(s), such as comparisons and arithmetic operations.

Operators are used to specify conditions in an SQL statement and to serve as conjunctions for multiple conditions in a statement.

Different types of SQL Operators:

1. Arithmetic operators
2. Comparison operators
3. Logical operators
4. Operators used to negate conditions

1. SQL Arithmetic Operators:

Assume variable a holds 10 and variable b holds 20, then:

Operator	Description	Example
+ Addition	Adds values on either side of the operator	a + b will give 30
- Subtraction	Subtracts right hand operand from left hand operand	a - b will give -10
*	Multiplies values on either side of the operator	a * b will give 200
/ Division	Divides left hand operand by right hand operand	b / a will give 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a will

Here are simple examples showing usage of SQL Arithmetic Operators:

SQL> select 10+ 20 from dual;

```
+----+
| 10+ 20 |
+----+
| 30 |
+----+
```

SQL> select 10+ 20 as addition from dual;

ADDITION

30

SQL> select 12 - 5 from dual;

```
+-----+
```

```
| 12 - 5 |
```

```
+-----+
```

```
| 10 |
```

```
+-----+
```

SQL> select 10 - 20 as subtraction from dual; SUBTRACTION

```
-----
```

-10

SQL> select 10 * 20 from dual;

```
+-----+
```

```
| 10 * 20 |
```

```
+-----+
```

```
| 200 |
```

```
+-----+
```

SQL> select 10 / 5 from dual;

```
+-----+
```

```
| 10 / 5 |
```

```
+-----+
```

```
| 2.0000 |
```

```
+-----+
```

2. SQL Comparison Operators:

Assume variable a holds **10** and variable b holds **20**, then:

Operator	Description	Example

=	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(a = b) is not true. (10=20)
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a != b) is true. (10!=20)
\diamond	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a \diamond b) is true. (10 \diamond 20)
>	Checks if the value of left operand is greater than the value of right operand, if yes then	(a > b) is not true. (10>20)
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is true. (10<20)
\geq	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(a \geq b) is not true. (10 \geq 20)
\leq	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(a \leq b) is true. (10 \leq 20)
!<	Checks if the value of left operand is not less than the value of right operand, if yes then condition becomes true.	(a !< b) is false. (10!<20)
!>	Checks if the value of left operand is not greater than the value of right operand, if yes then condition becomes true.	(a !> b) is true. (10!>20)

Consider the **CUSTOMERS** table having the following records:

SQL> SELECT * FROM CUSTOMERS;

+-----+ +

| ID | NAME | AGE | ADDRESS | SALARY |

+-----+ +

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

```
| 3 | kaushik | 23 | Kota | 2000.00 |
| 4 | Chaitali | 25 | Mumbai | 6500.00 |
| 5 | Hardik | 27 | Bhopal | 8500.00 |
| 6 | Komal | 22 | MP | 4500.00 |
| 7 | Muffy | 24 | Indore | 10000.00 |
+---+-----+---+      +      +

```

7 rows in set (0.00 sec)

Here are simple examples showing usage of SQL Comparison Operators:

SQL> SELECT * FROM CUSTOMERS WHERE SALARY > 5000;

```
+---+-----+---+-----+
| ID | NAME | AGE | ADDRESS | SALARY |
+---+-----+---+-----+
| 4 | Chaitali | 25 | Mumbai | 6500.00 |
| 5 | Hardik | 27 | Bhopal | 8500.00 |
| 7 | Muffy | 24 | Indore | 10000.00 |

```

```
+-----+-----+-----+
```

3 rows in set (0.00 sec)

SQL> SELECT * FROM CUSTOMERS WHERE SALARY = 2000;

```
+-----+-----+-----+
```

ID	NAME	AGE	ADDRESS	SALARY
----	------	-----	---------	--------

```
+-----+-----+-----+
```

1	Ramesh	32	Ahmedabad	2000.00
---	--------	----	-----------	---------

3	kaushik	23	Kota	2000.00
---	---------	----	------	---------

```
+-----+-----+-----+
```

2 rows in set (0.00 sec)

SQL> SELECT * FROM CUSTOMERS WHERE SALARY != 2000;

```
+-----+-----+-----+
```

ID	NAME	AGE	ADDRESS	SALARY
----	------	-----	---------	--------

```
+-----+-----+-----+
```

2	Khilan	25	Delhi	1500.00
---	--------	----	-------	---------

4	Chaitali	25	Mumbai	6500.00
---	----------	----	--------	---------

5	Hardik	27	Bhopal	8500.00
---	--------	----	--------	---------

6	Komal	22	MP	4500.00
---	-------	----	----	---------

7	Muffy	24	Indore	10000.00
---	-------	----	--------	----------

```
+-----+-----+-----+
```

5 rows in set (0.00 sec)

SQL> SELECT * FROM CUSTOMERS WHERE SALARY <> 2000;

```
+-----+-----+-----+
```

ID	NAME	AGE	ADDRESS	SALARY
----	------	-----	---------	--------

```
+-----+-----+-----+
```

2	Khilan	25	Delhi	1500.00
---	--------	----	-------	---------

4	Chaitali	25	Mumbai	6500.00
---	----------	----	--------	---------

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+-----+-----+-----+

5 rows in set (0.00 sec)

```
SQL> SELECT * FROM CUSTOMERS WHERE SALARY >= 6500;
```

```
+-----+-----+-----+
| ID | NAME | AGE | ADDRESS | SALARY |
+-----+-----+-----+
| 4 | Chaitali | 25 | Mumbai | 6500.00 |
| 5 | Hardik | 27 | Bhopal | 8500.00 |
| 7 | Muffy | 24 | Indore | 10000.00 |
+-----+
3 rows in set (0.00 sec)
```

3. SQL Logical Operators:

Here is a list of all the **Logical Operators** available in SQL.

OPERATOR	DESCRIPTION
ALL	The ALL operator is used to compare a value to all values in another value set.
AND	The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
ANY	The ANY operator is used to compare a value to any applicable value in the list according to the condition.
BETWEEN	The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.
IN	The EXISTS operator is used to search for the presence of a row in a specified table that meets certain criteria.
LIKE	The IN operator is used to compare a value to a list of literal values that have been specified.
NOT	The LIKE operator is used to compare a value to similar values using wildcard operators.
OR	The NOT operator reverses the meaning of the logical operator with which it is used. [Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. This is a negate operator.]
	The OR operator is used to combine multiple conditions in an SQL

	statement's WHERE clause.
IS NULL	The NULL operator is used to compare a value with a NULL value.
UNIQUE	The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).

Consider the **CUSTOMERS** table having the following records:

SQL> SELECT * FROM CUSTOMERS;

```
+-----+-----+-----+
| ID | NAME | AGE | ADDRESS | SALARY |
+-----+-----+-----+
| 1  | Ramesh | 32 | Ahmedabad | 2000.00 |
```

2 Khilan 25 Delhi 1500.00
3 kaushik 23 Kota 2000.00
4 Chaitali 25 Mumbai 6500.00
5 Hardik 27 Bhopal 8500.00
6 Komal 22 MP 4500.00
7 Muffy 24 Indore 10000.00

-----+-----+-----+

7 rows in set (0.00 sec)

Here are simple examples showing usage of SQL **Logical Operators**:

SQL> SELECT * FROM CUSTOMERS WHERE AGE >= 25 AND SALARY >= 6500;

-----+-----+-----+-----+-----+
ID NAME AGE ADDRESS SALARY
-----+-----+-----+-----+-----+
4 Chaitali 25 Mumbai 6500.00
5 Hardik 27 Bhopal 8500.00
-----+-----+-----+-----+-----+

2 rows in set (0.00 sec)

SQL> SELECT * FROM CUSTOMERS WHERE AGE >= 25 OR SALARY >= 6500;

-----+-----+-----+-----+-----+
ID NAME AGE ADDRESS SALARY
-----+-----+-----+-----+-----+
1 Ramesh 32 Ahmedabad 2000.00
2 Khilan 25 Delhi 1500.00
4 Chaitali 25 Mumbai 6500.00
5 Hardik 27 Bhopal 8500.00
7 Muffy 24 Indore 10000.00
-----+-----+-----+-----+-----+

5 rows in set (0.00 sec)

SQL> SELECT * FROM CUSTOMERS WHERE AGE IS NOT NULL;

+-----+-----+-----+

| ID | NAME | AGE | ADDRESS | SALARY |

+-----+-----+-----+

```
| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |
| 2 | Khilan | 25 | Delhi | 1500.00 |
| 3 | kaushik | 23 | Kota | 2000.00 |
| 4 | Chaitali | 25 | Mumbai | 6500.00 |
| 5 | Hardik | 27 | Bhopal | 8500.00 |
| 6 | Komal | 22 | MP | 4500.00 |
| 7 | Muffy | 24 | Indore | 10000.00 |
```

7 rows in set (0.00 sec)

SQL> SELECT * FROM CUSTOMERS WHERE NAME **LIKE** 'Ko%';

```
+-----+ + +-----+
| ID | NAME | AGE | ADDRESS | SALARY |
+-----+ + +-----+
| 6 | Komal | 22 | MP | 4500.00 |
```

1 row in set (0.00 sec)

SQL> SELECT * FROM CUSTOMERS WHERE AGE **IN** (25, 27);

```
+-----+ + +-----+
| ID | NAME | AGE | ADDRESS | SALARY |
+-----+ + +-----+
| 2 | Khilan | 25 | Delhi | 1500.00 |
| 4 | Chaitali | 25 | Mumbai | 6500.00 |
| 5 | Hardik | 27 | Bhopal | 8500.00 |
```

3 rows in set (0.00 sec)

SQL> SELECT * FROM CUSTOMERS WHERE AGE **BETWEEN** 25 AND 27;

```
+-----+ + +-----+
```

ID	NAME	AGE	ADDRESS	SALARY
----	------	-----	---------	--------

ID	NAME	AGE	ADDRESS	SALARY
----	------	-----	---------	--------

2	Khilan	25	Delhi	1500.00
---	--------	----	-------	---------

4	Chaitali	25	Mumbai	6500.00
---	----------	----	--------	---------

5	Hardik	27	Bhopal	8500.00
---	--------	----	--------	---------

```
+-----+-----+-----+
```

3 rows in set (0.00 sec)

SQL> SELECT AGE FROM CUSTOMERS WHERE **EXISTS** (SELECT AGE FROM CUSTOMERS WHERE SALARY > 6500);

```
+---+
```

AGE

```
+---+
```

32

25

23

25

27

22

24

```
+---+
```

7 rows in set (0.02 sec)

SQL> SELECT * FROM CUSTOMERS WHERE AGE > **ALL** (SELECT AGE FROM CUSTOMERS WHERE SALARY > 6500);

```
+-----+---+ + +
```

ID	NAME	AGE	ADDRESS	SALARY
----	------	-----	---------	--------

```
+-----+---+ + +
```

1	Ramesh	32	Ahmedabad	2000.00
---	--------	----	-----------	---------

```
+-----+---+ + +
```

1 row in set (0.02 sec)

SQL> SELECT * FROM CUSTOMERS WHERE AGE > **ANY** (SELECT AGE FROM CUSTOMERS WHERE SALARY > 6500);

```
+-----+-----+-----+
```

ID	NAME	AGE	ADDRESS	SALARY
----	------	-----	---------	--------

+ + + + + +

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

```
+-----+-----+-----+
```

4 rows in set (0.00 sec)

SQL AND and OR Operators:

The SQL AND and OR operators are used to combine multiple conditions to narrow data in an SQL statement. These two operators are called conjunctive operators.

These operators provide a means to make multiple comparisons with different operators in the same SQL statement.

The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.

Syntax:

The basic syntax of AND operator with WHERE clause is as follows:

```
SELECT column1, column2, column FROM table_name WHERE [condition1] AND  
[condition2]...AND [conditionN];
```

You can combine N number of conditions using AND operator. For an action to be taken by the SQL statement, whether it be a transaction or query, all conditions separated by the AND must be TRUE.

Example: Consider the **CUSTOMERS** table having the following records:

```
+-----+-----+-----+
```

ID	NAME	AGE	ADDRESS	SALARY
----	------	-----	---------	--------

```
+-----+-----+-----+
```

1	Ramesh	32	Ahmedabad	2000.00
---	--------	----	-----------	---------

2	Khilan	25	Delhi	1500.00
---	--------	----	-------	---------

3	kaushik	23	Kota	2000.00
---	---------	----	------	---------

4	Chaitali	25	Mumbai	6500.00
---	----------	----	--------	---------

5	Hardik	27	Bhopal	8500.00
---	--------	----	--------	---------

6	Komal	22	MP	4500.00
---	-------	----	----	---------

7	Muffy	24	Indore	10000.00
---	-------	----	--------	----------

```
+-----+-----+-----+
```

Following is an example, which would fetch ID, Name and Salary fields from the

CUSTOMERS table where salary is greater than 2000 AND age is less than 25 years:

**SQL> SELECT ID, NAME, SALARY FROM CUSTOMERS WHERE SALARY > 2000
AND age < 25;**

This would produce the following result:

ID	NAME	SALARY
6	Komal	4500.00

7	Muffy	10000.00
---	-------	----------

-----+-----+

The OR Operator:

The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause. The basic **syntax** of OR operator with WHERE clause is as follows:

```
SELECT column1, column2, columnN FROM table_name WHERE [condition1] OR [condition2]...OR [conditionN];
```

You can combine N number of conditions using OR operator. For an action to be taken by the SQL statement, whether it be a transaction or query, only any ONE of the conditions separated by the OR must be TRUE.

Example: Consider the CUSTOMERS table having the following records:

ID	NAME	AGE	ADDRESS	SALARY
----	------	-----	---------	--------

-----+-----+-----+-----+

1	Ramesh	32	Ahmedabad	2000.00
---	--------	----	-----------	---------

2	Khilan	25	Delhi	1500.00
---	--------	----	-------	---------

3	kaushik	23	Kota	2000.00
---	---------	----	------	---------

4	Chaitali	25	Mumbai	6500.00
---	----------	----	--------	---------

5	Hardik	27	Bhopal	8500.00
---	--------	----	--------	---------

6	Komal	22	MP	4500.00
---	-------	----	----	---------

7	Muffy	24	Indore	10000.00
---	-------	----	--------	----------

-----+-----+-----+-----+

Following is an example, which would fetch ID, Name and Salary fields from the CUSTOMERS table where salary is greater than 2000 OR age is less than 25 years:

```
SQL> SELECT ID, NAME, SALARY FROM CUSTOMERS WHERE SALARY > 2000  
OR age < 25;
```

This would produce the following result:

-----+-----+

ID	NAME	SALARY
----	------	--------

+ + + +

| 3 | kaushik | 2000.00 |

| 4 | Chaitali | 6500.00 |

| 5 | Hardik | 8500.00 |

| 6 | Komal | 4500.00 |

| 7 | Muffy | 10000.00 |

SQL CLAUSES

Clauses in SQL are similar to conditionals in high-level languages. SQL being a query language requires a method to apply constraints on the data and for this we use Clauses. We have a large variety in the SQL clauses like the Where clause, Union Clause, Order By clause etc.

Clauses help us to restrict and manage the data using valid constraints on the data in our database.

In SQL (Structured Query Language), a clause is a component of a SQL statement that performs a specific function in a query. It is a part of a SQL statement that specifies certain conditions and operations to be performed on data. A SQL statement usually consists of one or more clauses combined together to create a complete statement that retrieves, manipulates or modifies data stored in a database

Use of SQL Clause:

Some of the uses of SQL Clauses are as follows:

1. Allow us to apply constraints on data.
2. Help us to reduce the complexity of the query.
3. With the help of clauses, we can filter the data according to our requirements.
4. Allow us to restrict the number of outputs we need.
5. Help in writing user-friendly queries, which are easy to read and understand.

Some of the most common clauses in SQL include:

Clause	Description
SELECT	retrieves data from one or more tables in a database.
FROM	specifies the table or tables from which to retrieve data.
WHERE	filters data based on a specified condition.
GROUP BY	groups data based on a specified column or expression.
HAVING	filters data based on a specified condition, after grouping.
ORDER BY	sorts data based on a specified column or expression.
LIMIT	limits the number of rows returned by a query.
JOIN	combines data from two or more tables based on a specified relationship.

Each clause in SQL has its own syntax and usage, and combining multiple clauses together in a SQL statement allows you to perform complex queries and operations on the data stored in a database.

SQL SELECT Clause:

The SELECT clause is a fundamental clause in the SQL (Structured Query Language) used to retrieve data from one or more tables in a database. It specifies the columns or expressions to be returned in the result set of a query.

In its simplest form, the SELECT clause is used to retrieve all columns from a table:

SQL> SELECT * FROM customers;

Alternatively, you can **specify** the columns you want to retrieve explicitly:

SQL> SELECT customer_name, email **FROM** customers;

This query retrieves only the "customer_name" and "email" columns from the "customers" table.

Rename columns in the result set:

SQL> SELECT customer_name **AS** name, email **AS** contact **FROM** customers;

SQL FROM Clause:

In SQL, the **FROM** clause is used to specify the table or tables from which data should be retrieved. It is a mandatory clause that appears immediately after the **SELECT** clause in a SQL query.

The basic syntax of the **FROM** clause is as follows:

SQL> SELECT column1, column2, ... **FROM** table1

Where "column1", "column2", and so on represent the columns to be retrieved, and "table1" represents the table from which the data should be retrieved.

Here is an **example** of a simple SQL query that uses the **FROM** clause to retrieve data from a single table:

SQL>SELECT * FROM customers;

In this example, the asterisk (*) represents all columns in the "customers" table. The **FROM** clause specifies that the data should be retrieved from the "customers" table.

SQL WHERE Clause:

- > The SQL **WHERE** clause is used to specify a condition while fetching the data from single table or joining with multiple tables.
- > If the given condition is satisfied, then only it returns specific value from the table. You would use **WHERE** clause to filter the records and fetching only necessary records.
- > The **WHERE** clause is not only used in **SELECT** statement, but it is also used in **UPDATE**, **DELETE** statement etc...

Syntax:

The basic syntax of **SELECT** statement with **WHERE** clause is as follows:

SELECT column1, column2, columnN **FROM** table_name **WHERE** [condition]

You can specify a condition using comparison or logical operators like **>**, **<**, **=**, **LIKE**, **NOT** etc. Below examples would make this concept clear.

Example:

Consider the **CUSTOMERS** table having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00

4 Chaitali 25 Mumbai 6500.00
5 Hardik 27 Bhopal 8500.00
6 Komal 22 MP 4500.00
7 Muffy 24 Indore 10000.00
-----+-----+-----+

Following is an example, which would fetch ID, Name and Salary fields from the **CUSTOMERS** table where salary is greater than 2000:

SQL> SELECT ID, NAME, SALARY FROM CUSTOMERS WHERE SALARY > 2000;

This would produce the following result:

ID NAME SALARY
-----+-----+-----+
4 Chaitali 6500.00
5 Hardik 8500.00
6 Komal 4500.00
7 Muffy 10000.00
-----+-----+-----+

Following is an **example**, which would fetch ID, Name and Salary fields from the **CUSTOMERS** table for a customer with name Hardik. Here, it is important to note that all the strings should be given inside single quotes ('') where as numeric values should be given without any quote as in above example:

SQL> SELECT ID, NAME, SALARY FROM CUSTOMERS WHERE NAME = 'Hardik';

This would produce the following result:

-----+-----+-----+
ID NAME SALARY
-----+-----+-----+
5 Hardik 8500.00
-----+-----+-----+

SQL LIKE Clause:

The SQL LIKE clause is used to compare a value to similar values using wildcard operators. There are two wildcards used in conjunction with the **LIKE** operator:

The percent sign

(%) The
underscore (_)

The **percent sign** represents zero, one, or multiple characters. The underscore represents a single number or character. The symbols can be used in combinations.

Syntax:

The basic syntax of % and _ is as follows:

SELECT FROM table_name WHERE column **LIKE** 'XXXX%'

or

SELECT FROM table_name WHERE column **LIKE** '%XXXX'

or

SELECT FROM table_name WHERE column **LIKE**
'XXXX_'

or

SELECT FROM table_name WHERE column **LIKE**
'_XXXX'

or

SELECT FROM table_name WHERE column **LIKE**
'_XXXX_'

You can combine N number of conditions using AND or OR operators. Here, XXXX could be any numeric or string value.

Example:

Here are number of examples showing WHERE part having different LIKE clause with '%' and '_' operators:

Statement	Description
WHERE SALARY LIKE '200%'	Finds any values that start with 200
WHERE SALARY LIKE '%200%'	Finds any values that have 200 in any position
WHERE SALARY LIKE '_00%'	Finds any values that have 00 in the second and third positions
WHERE SALARY LIKE '2_%_%'	Finds any values that start with 2 and are at least 3 characters in length
WHERE SALARY LIKE "%2"	Finds any values that end with 2
WHERE SALARY LIKE '_2%3'	Finds any values that have a 2 in the second position and end with a 3
WHERE SALARY LIKE '2 3'	Finds any values in a five-digit number that start with 2 and end with 3

Let us take a real example, consider the **CUSTOMERS** table having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00

5 Hardik 27 Bhopal 8500.00
6 Komal 22 MP 4500.00
7 Muffy 24 Indore 10000.00
-----+-----+-----+-----+

Following is an example, which would display all the records from **CUSTOMERS** table where SALARY starts with 200:

SQL> SELECT * FROM CUSTOMERS WHERE SALARY LIKE '200%';

-----+-----+-----+-----+
ID NAME AGE ADDRESS SALARY
-----+-----+-----+-----+
1 Ramesh 32 Ahmedabad 2000.00
3 kaushik 23 Kota 2000.00

SQL ORDER BY Clause:

The SQL ORDER BY clause is used to sort the data in ascending or descending order, based on one or more columns. Some database sorts query results in ascending order by default.

Syntax:

SELECT column-list FROM table_name [WHERE condition] [ORDER BY column1, column2, .. columnN] [ASC | DESC];

You can use more than one column in the ORDER BY clause. Make sure whatever column you are using to sort, that column should be in column-list.

Example:

Consider the **CUSTOMERS** table having the following records:

-----+-----+-----+-----+
ID NAME AGE ADDRESS SALARY
-----+-----+-----+-----+
1 Ramesh 32 Ahmedabad 2000.00
2 Khilan 25 Delhi 1500.00

3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Following is an **example**, which would sort the result in ascending order by NAME and SALARY:

SQL> SELECT * FROM CUSTOMERS ORDER BY NAME, SALARY;

This would produce the following result:

ID	NAME	AGE	ADDRESS	SALARY
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
3	kaushik	23	Kota	2000.00
2	Khilan	25	Delhi	1500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00
1	Ramesh	32	Ahmedabad	2000.00

Following is an example, which would sort the result in descending order by NAME:

SQL> SELECT * FROM CUSTOMERS ORDER BY NAME DESC;

This would produce the following result:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
7	Muffy	24	Indore	10000.00
6	Komal	22	MP	4500.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
5	Hardik	27	Bhopal	8500.00

4 Chaitali 25 Mumbai 6500.00

+-----+-----+-----+

Following is an example, which would sort the result in ascending order by NAME:

SQL> SELECT * FROM CUSTOMERS ORDER BY NAME ASC;

This would produce the following result:

+-----+-----+-----+

ID	NAME	AGE	ADDRESS	SALARY
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
3	kaushik	23	Kota	2000.00
2	Khilan	25	Delhi	1500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00
1	Ramesh	32	Ahmedabad	2000.00

SQL Group By:

- The SQL GROUP BY clause is used in collaboration with the SELECT statement to arrange identical data into groups.
- The GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

Syntax:

The basic syntax of GROUP BY clause is given below. The GROUP BY clause must follow the conditions in the **WHERE** clause and must precede the **ORDER BY** clause if one is used.

SELECT column1, column2 FROM table_name WHERE [conditions] GROUP BY column1, column2 ORDER BY column1, column2

Example1:

```
SQL> SELECT iteam_supplied, SUM(iteam_price) as totalamount FROM supplier
GROUP BY
iteam_supplied;
```

ITEAM_SUPPLIED TOTALAMOUNT

Processor 25500

Printer 6000

Mouse 350

Keyboard 400

Consider the **CUSTOMERS** table having the following records:

Example2:

ID	NAME	AGE	ADDRESS	SALARY

1 Ramesh 32 Ahmedabad 2000.00
2 Khilan 25 Delhi 1500.00
3 kaushik 23 Kota 2000.00
4 Chaitali 25 Mumbai 6500.00
5 Hardik 27 Bhopal 8500.00
6 Komal 22 MP 4500.00
7 Muffy 24 Indore 10000.00

-----+-----+

If you want to know the total amount of salary on each customer, then **GROUP BY** query would be as follows:

SQL> SELECT NAME, SUM(SALARY) FROM CUSTOMERS **GROUP BY** NAME;

This would produce the following result:

-----+-----+

NAME SUM(SALARY)

-----+-----+

Chaitali 6500.00

Hardik 8500.00

kaushik 2000.00

Khilan 1500.00

Komal 4500.00

Muffy 10000.00

Ramesh 2000.00

-----+-----+

Now, let us have following table where **CUSTOMERS** table has the following records with duplicate names:

-----+-----+

ID NAME AGE ADDRESS SALARY

+ + + + + +

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Ramesh | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | kaushik | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

```
| 7 | Muffy | 24 | Indore | 10000.00 |
```

```
+-----+-----+-----+
```

Now again, if you want to know the total amount of salary on each customer, then GROUP BY query would be as follows:

```
SQL> SELECT NAME, SUM(SALARY) FROM CUSTOMERS GROUP BY NAME;
```

This would produce the following result:

```
| NAME | SUM(SALARY) |
```

```
+-----+-----+
```

```
| Hardik | 8500.00 |
```

```
| kaushik | 8500.00 |
```

```
| Komal | 4500.00 |
```

```
| Muffy | 10000.00 |
```

```
| Ramesh | 3500.00 |
```

```
+-----+
```

SQL Distinct Keyword:

The SQL **DISTINCT** keyword is used in conjunction with SELECT statement to eliminate all the duplicate records and fetching only unique records.

There may be a situation when you have multiple duplicate records in a table. While fetching such records, it makes more sense to fetch only unique records instead of fetching duplicate records.

Syntax:

The basic syntax of DISTINCT keyword to eliminate duplicate records is as follows:

```
SELECT DISTINCT column1, column2, ..., columnN FROM table_name WHERE  
[condition]
```

Example1:

```
SQL> SELECT DISTINCT iteam_supplied FROM supplier ORDER BY iteam_supplied;  
ITEAM_SUPPLIED
```

Keyboard

Mouse

Printer

Processor

Example

2:

Consider the CUSTOMERS table having the following records:

+-----+-----+-----+

ID	NAME	AGE	ADDRESS	SALARY
----	------	-----	---------	--------

ID	NAME	AGE	ADDRESS	SALARY
----	------	-----	---------	--------

1	Ramesh	32	Ahmedabad	2000.00
---	--------	----	-----------	---------

2	Khilan	25	Delhi	1500.00
---	--------	----	-------	---------

3	kaushik	23	Kota	2000.00
---	---------	----	------	---------

4	Chaitali	25	Mumbai	6500.00
---	----------	----	--------	---------

5	Hardik	27	Bhopal	8500.00
---	--------	----	--------	---------

6	Komal	22	MP	4500.00
---	-------	----	----	---------

7	Muffy	24	Indore	10000.00
---	-------	----	--------	----------

ID	NAME	AGE	ADDRESS	SALARY
----	------	-----	---------	--------

First, let us see how the following SELECT query returns duplicate salary records:

SQL> SELECT SALARY FROM CUSTOMERS ORDER BY SALARY;

This would produce the following result where salary 2000 is coming twice which is a duplicate record from the original table.

SALARY

SALARY

1500.00

2000.00

2000.00

4500.00

6500.00

8500.00

10000.00

SALARY

Now, let us use DISTINCT keyword with the above SELECT query and see the result:

SQL> SELECT DISTINCT SALARY FROM CUSTOMERS ORDER BY SALARY;

This would produce the following result where we do not have any duplicate entry:

+-----+

| SALARY |

+-----+

| 1500.00 |

2000.00
4500.00
6500.00
8500.00
10000.00

SQL SORTING

The SQL ORDER BY clause is used to sort the data in ascending or descending order, based on one or more columns. Some databases sort query results in ascending order by default.

Syntax: The basic syntax of ORDER BY clause which would be used to sort result in ascending or descending order is as follows:

SELECT column-list FROM table_name [WHERE condition] [ORDER BY column1, column2, .. columnN] [ASC | DESC];

You can use more than one column in the ORDER BY clause. Make sure whatever column you are using to sort, that column should be in column-list.

Example:

Consider the CUSTOMERS table having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Following is an example, which would sort the result in ascending order by NAME and SALARY:

SQL> SELECT * FROM CUSTOMERS ORDER BY NAME, SALARY;

This would produce the following result:

+-----+-----+-----+

| ID | NAME | AGE | ADDRESS | SALARY |

4 Chaitali 25 Mumbai 6500.00				
5 Hardik 27 Bhopal 8500.00				
3 kaushik 23 Kota 2000.00				
2 Khilan 25 Delhi 1500.00				
6 Komal 22 MP 4500.00				
7 Muffy 24 Indore 10000.00				
1 Ramesh 32 Ahmedabad 2000.00				

Following is an example, which would sort the result in descending order by NAME:

SQL> SELECT * FROM CUSTOMERS ORDER BY NAME DESC;

This would produce the following result:

ID NAME AGE ADDRESS SALARY				
1 Ramesh 32 Ahmedabad 2000.00				
7 Muffy 24 Indore 10000.00				
6 Komal 22 MP 4500.00				
2 Khilan 25 Delhi 1500.00				
3 kaushik 23 Kota 2000.00				
5 Hardik 27 Bhopal 8500.00				
4 Chaitali 25 Mumbai 6500.00				

To fetch the rows with own preferred order, the SELECT query would be as follows:

**SQL> SELECT * FROM CUSTOMERS ORDER BY (CASE
ADDRESS WHEN 'DELHI' THEN 1
WHEN 'BHOPAL' THEN
2 WHEN 'KOTA' THEN 3**

```
WHEN 'AHMADABAD' THEN 4
WHEN 'MP' THEN 5
ELSE 100 END) ASC, ADDRESS DESC;
```

This would produce the following result:

ID	NAME	AGE	ADDRESS	SALARY
2	Khilan	25	Delhi	1500.00
5	Hardik	27	Bhopal	8500.00
3	kaushik	23	Kota	2000.00
6	Komal	22	MP	4500.00
4	Chaitali	25	Mumbai	6500.00
7	Muffy	24	Indore	10000.00
1	Ramesh	32	Ahmedabad	2000.00

This will sort customers by ADDRESS in your own Order of preference first and in a natural order for the remaining addresses. Also remaining Addresses will be sorted in the reverse alpha order.

SQL Queries Syntax:

SQL is followed by unique set of rules and guidelines called Syntax. This tutorial gives you a quick start with

SQL by listing all the basic SQL Syntax:

All the SQL statements start with any of the keywords like CREATE, ALTER, INSERT, UPDATE, SELECT, DELETE, DROP, USE, SHOW and all the statements end with a semicolon (;).

Query Name	Syntax
SQL SELECT Statement:	SELECT column1, column2, columnN FROM table_name;
SQL DISTINCT Clause:	SELECT DISTINCT column1, column2....columnN FROM table_name;
SQL WHERE Clause:	SELECT column1, column2, columnN FROM table_name WHERE CONDITION;
SQL AND/OR Clause:	SELECT column1, column2, columnN FROM table_name WHERE

	CONDITION-1 {AND OR} CONDITION-2;
SQL IN Clause:	SELECT column1, column2, columnN FROM table_name WHERE column_name IN (val-1, val-2, ..., val-N);
SQL BETWEEN Clause:	SELECT column1, column2, columnN FROM table_name WHERE column_name BETWEEN val-1 AND val-2;
SQL LIKE Clause:	SELECT column1, column2, columnN FROM table_name WHERE column_name LIKE { PATTERN };
SQL ORDER BY Clause:	SELECT column1, column2, columnN FROM table_name WHERE CONDITION ORDER BY column_name {ASC DESC};
SQL GROUP BY Clause:	SELECT SUM(column_name) FROM table_name WHERE CONDITION GROUP BY column_name;
SQL COUNT Clause:	SELECT COUNT(column_name) FROM table_name WHERE CONDITION;
SQL HAVING Clause:	SELECT SUM(column_name) FROM table_name WHERE CONDITION GROUP BY column_name HAVING (arithematic function condition);
SQL CREATE TABLE Statement:	CREATE TABLE table_name(column1 datatype, column2 datatype, column3 datatype, ..., columnN datatype, PRIMARY KEY(one or more columns));
SQL DROP TABLE Statement:	DROP TABLE table_name;
SQL CREATE INDEX Statement:	CREATE UNIQUE INDEX index_name ON table_name (column1, column2, ... columnN);
SQL DROP INDEX Statement:	ALTER TABLE table_name DROP INDEX index_name;
SQL DESC Statement:	DESC table_name;
SQL TRUNCATE TABLE Statement:	TRUNCATE TABLE table_name;
SQL ALTER TABLE Statement:	ALTER TABLE table_name {ADD DROP MODIFY} column_name

	{data_type};
SQL INSERT INTO Statement:	INSERT INTO table_name(column1, column2....columnN)VALUES (value1, value2. valueN);
SQL UPDATE Statement:	UPDATE table_name SET column1 = value1, column2 = value2. columnN=valueN [WHERE CONDITION];
SQL DELETE Statement:	DELETE FROM table_name WHERE {CONDITION};
SQL CREATE DATABASE Statement:	CREATE DATABASE database_name;
SQL DROP DATABASE Statement:	DROP DATABASE database_name;
SQL USE Statement:	USE DATABASE database_name;
SQL COMMIT Statement:	COMMIT;
SQL ROLLBACK Statement:	ROLLBACK;

Module-6: SQL FUNCTIONS

SQL FUNCTIONS

Functions will manipulate the data items and gives the result. They are two types of functions.

1. Group functions or Multiple row functions or Aggregate functions.

2. Scalar functions or Single row function.

1. Group functions or multiple row functions or Aggregate functions:

This functions act on group of rows.

- **AVG:** In Oracle Database, the AVG function is an aggregate function that calculates the average value of a set of values. It is commonly used to find the mean value of a column in a table or a set of values specified in the query.

Syntax: AVG(column_name);

Ex: SQL> select AVG(subject1) from stu1;
AVG(SUBJECT1)

72.8571429

- **SUM:** In Oracle Database, the SUM function is an aggregate function that is used to calculate the sum of a set of values. It is often used in conjunction with the GROUP BY clause to perform summation within groups.

Syntax: SUM(column_name) FROM
table_name;; Ex: SQL> select sum(subject1)
from stu1; SUM(SUBJECT1)

510

SQL> select sum(subject1) ,htno from stu1 group
by htno; SUM(SUBJECT1)HTNO

95 102
90 101
0 107

98	104
54	105
99	103
74	106

7 rows selected.

- **MAX:** In Oracle Database, the MAX function is an aggregate function that is used to find the maximum value within a set of values. It is often used in conjunction with the GROUP BY clause to find the maximum value for each group.

Syntax: MAX(column_name) FROM
table_name;; Ex: SQL> select max(subject1)
from stu1; MAX(SUBJECT1)

99

- **MIN:** In Oracle Database, the MIN function is an aggregate function that is used to find the minimum value of a set of values. It is commonly used with the SELECT statement to retrieve the smallest value from a specified column or expression.

Syntax: MIN(column_name) FROM
table_name;; Ex: SQL> select min(subject1)
from stu1; MIN(SUBJECT1)

0

- **COUNT 0:**

Count All Rows:

If you use COUNT(*), it counts all rows in a table, including rows with NULL values in any of the columns. In Oracle Database, the COUNT(*) function is an aggregate function that returns the number of rows in a specified table, view, or result set that meets the specified conditions. It counts all the rows, including those with NULL values in the specified column (or all columns if * is used).

Syntax: select count(*) from stu1; FROM table_name;;

Ex: SQL> select count(*) from stu1; //Return total no.of rows in
the table COUNT(*)

8

Count Rows Based on a Condition:

If you use COUNT with a specific column or expression, it counts the number of rows where the specified column or expression is not NULL.

Syntax: `SELECT COUNT(column_name) FROM your_table;`

`SELECT COUNT(column_name) FROM your_table WHERE condition;`

Ex: `SQL> select count(htno) from`

`stu1; COUNT(HTNO)`

8

SQL> select count(subject1) from stu1 where subject1>=90;
COUNT(SUBJECT1)

4

Count Distinct Values:

If you use COUNT(DISTINCT column_name), it counts the number of distinct (unique) non-null values in the specified column.

Syntax: SELECT COUNT(DISTINCT column_name) FROM your_table;

Ex: SQL> SELECT COUNT(DISTINCT
htno) FROM stu1;
COUNT(DISTINCTHTNO)

7

Dual table: It is a dummy table which is generally used to perform some calculation is seeing to the system date and etc. Dual table is collection of one row and one column with 'X' in it.

Ex: **Select SYSDATE from dual;**

Select 10+20 from dual;

Select 20+40, 50+60 from dual;

2. Scalar functions or single row functions:

Scalar function are decided into four types. They are given that.

- A. Character functions**
- B. Number functions**
- C. Data functions**
- D. Conversion functions**

A. Character functions:

Upper: converts into lower case to upper case. Ex: Select upper('oracle') from dual;
//ORACLE

Lower: This function is convert to the upper to lower case. Ex: Select lower('ORACLE') from dual;
//oracle

SQL> select sid,upper(sname) from student; SID UPPER(SNAM

1 KRISHNA

2 SANDEEP

3 KEERTHI

4 RENUKA

SQL> Select ENO, lower('ENAME'), SAL from emp;

INITCAP: First letter is capital and reaming letters are small letters.

Ex: Select INITCAP('oracle training') from dual; //Oracle

SQL> Select INITCAP('ORACLE TRAINING') from dual;

//Oracle **SQL>** select initcap(sup_name) as SUP_NAME
from supplier; SUP_NAME

Murali

Kiran

Mohan

Ramesh

Manish

Srikanth

Suresh

LENGTH: Returns length of the string.

Ex: Select LENGTH('oracle') from dual; //length 6

SQL> Select LENGTH('MALLIKHARJUNA') from dual; //length 13

SQL> Select * from emp where length(ename) = 4;

SQL> select length(sup_name) as SUP_NAME from supplier; SUP_NAME

5

5

6

6
8
6

7 rows selected.

REPLACE: The REPLACE function is typically used within a SELECT statement, not as a standalone command. If you want to use the REPLACE function in a query and retrieve the result, you should do something like this:

Ex: SQL> SELECT REPLACE('Hello World', 'Hello', 'Hi') AS Result
FROM dual; RESULT

Hi World

SQL> SELECT htno,sname, REPLACE(sname, 'Krishna', 'Srinu') AS
updated_name FROM stu1; HTNO SNAME UPDATED_NAME

101 Renuka Renuka

102 Hari Hari

103 Vijith Vijith

104 Nithin Nithin

109

106 Krishna Srinu

105 fvbjhn fvbjhn

107 Srinath Srinath

108 Srinath Srinath

LPAD: In Oracle, the LPAD function is used to pad a string with a specified character or a specific set of characters on the left side (prefix) until the string reaches a specified length. This is helpful when you want to align values in a column or create formatted output.

Syntax: LPAD (string, length, [pad_character])

- > String is the input string that you want to pad.
- > length is the total desired length of the resulting padded string.

- > pad_character is an optional parameter specifying the character(s) to use for padding. By default, it uses a single space.pads the character towards the left side.

Ex: select LPAD('oracle',10,'z') from dual;
//zzzzoracle LPAD('ORAC

zzzzoracle

SQL> SELECT LPAD('Hello', 10, '*') AS padded_string FROM dual; PADDED_STR

*****Hello

SQL> SELECT LPAD('Hello', 10) AS padded_string FROM dual; PADDED_STR

Hello

RPAD: In Oracle, the RPAD function is used to right-pad a string with a specified character or a space to a specified length. This function is particularly useful when you want to align or format string values in a specific way. Rpad the character towards the right side.

Syntax: RPAD(string, length [, pad_string])

Ex: SQL> Select RPAD('ORACLE',10,'X') from dual;
RPAD('ORAC

ORACLEXXXX

SQL> SELECT RPAD('Hello', 10) AS padded_string FROM dual; PADDED_STR

Hello

SQL> SELECT RPAD('Hello', 10, '*') AS padded_string FROM dual; PADDED_STR

Hello*****

Nesting Function:[Both LPad and RPad]

SQL> SELECT LPAD(RPAD('ORACLE', 8, 'Z'), 10, 'Z') AS padded_string FROM dual; PADDED_STR

ZZORACLEZZ

LTRIM: In Oracle, the LTRIM function is used to remove leading spaces or a specific character from a string. It trims or removes characters from the left (start) of a string.

Syntax: LTRIM(string [, trim_string])

```
SQL> Select LTRIM('zzoracle','z') from
dual; LTRIM(
```

```
----
```

oracle

```
SQL> SELECT LTRIM(' Hello World') AS trimmed_string
FROM dual; TRIMMED_STR
```

```
----
```

Hello World

```
SQL> SELECT LTRIM('$$OpenAI$$', '$') AS trimmed_string FROM
dual; TRIMMED_S
```

```
----
```

OpenAI\$\$\$

RTRIM: In Oracle, the RTRIM function is used to remove specified characters or spaces from the right side of a string. It is commonly used to clean up trailing spaces or specific characters from string values.

Syntax: RTRIM(string [, trim_characters])

```
SQL> SELECT RTRIM('Hello   ') AS trimmed_string
FROM dual; TRIMM
```

```
----
```

Hello

```
SQL> Select RTRIM('ZORACLEZZZ','Z')
from dual; RTRIM(
```

```
----
```

ZORACLE

TRIM: In Oracle, the TRIM function is used to remove specified characters from the beginning or end of a string. It is commonly used to remove leading or trailing spaces or other specified characters. Removes the specified characters from both sides.

```
SQL> Select TRIM('z' from 'zzoraclezz' )
```

```
from dual; TRIM('
```

```
----
```

Oracle

```
SQL> Select TRIM('ORACLE ') from
```

```
dual; TRIM('
```

```
----
```

ORACLE

INSTR: In Oracle, the INSTR function is used to find the position of a substring within a string. It returns the starting position of the substring within the string. Returns the position of the string.

Syntax: INSTR(string, substring [, start_position [, nth_occurrence]])

```
SQL> SELECT INSTR('Hello World', 'Wo') AS position FROM dual; POSITION
```

7

```
SQL> Select INSTR('ORACLE','A') from dual; INSTR('ORACLE','A')
```

3

```
SQL> SELECT INSTR('The quick brown fox jumps over the lazy cat.', 'cat') FROM dual; INSTR('THEQUICKBROWNFOXJUMPSOVERTHELAZycat. ','CAT')
```

41

SUBSTR: Returns the part of the string. The SQL query SELECT SUBSTR('ORACLE', 2, 3) FROM dual; returns the substring of the string 'ORACLE' starting from the second character, and including the next 3 characters.

```
SQL> Select SUBSTR('ORACLE',2,3) from dual; SUB
```

RA

C

```
SQL> Select SUBSTR('ORACLE',2,4) from dual; SUB
```

RACL

E

```
SQL> Select SUBSTR('ORACLE',2,1) from dual; SUB
```

--

-

R

SQL> Select SUBSTR('ORACLE',3,2)
from dual; SUB

```
--  
A  
C  
SQL> Select SUBSTR('ORACLE',3,10) from  
dual; SUB
```

```
--  
ACLE  
SQL> Select SUBSTR('ORACLE',3) from  
dual; SUB
```

```
--  
ACLE
```

CONCAT: In Oracle, the CONCAT function is used to concatenate two or more strings together. It takes two string values as input and returns a single string that is the result of concatenating the input strings. To join the two words. It will accept only two characters.

Syntax: CONCAT(string1, string2)

```
SQL> SELECT CONCAT('Hello', 'World') AS concatenated_string FROM  
dual; CONCATENAT
```

```
-----  
  
HelloWorld
```

```
SQL> SELECT 'Hello' || ' ' || 'World' AS concatenated_string  
FROM dual; CONCATENATE
```

```
-----  
  
Hello World
```

```
SQL> SELECT 'Hello' || ' ' || 'beautiful' || ' ' || 'World' AS concatenated_string FROM dual;  
CONCATENATED_STRING
```

```
-----  
  
Hello beautiful World
```

B. Number functions:

ABS: In Oracle, the ABS function is used to return the absolute value of a number. It takes a numeric expression as an argument and returns the absolute value of that expression. Returns absolute values.

Syntax: ABS(numeric_expression)

```
SQL> SELECT ABS(-10) AS absolute_value FROM dual;  
ABSOLUTE_VALUE
```

10

```
SQL> SELECT ABS(-10.5) AS absolute_value FROM dual;  
ABSOLUTE_VALUE
```

10.5

SQRT: To calculate the square root (SQRT) of a number using an Oracle database, you can use the SQRT function. Returns the squawroot values.

Syntax: SELECT SQRT(number) FROM your_table;

```
SQL> Select SQRT(25) from dual;  
SQRT(25)
```

5

```
SQL> Select SQRT(97) from dual;  
SQRT(97)
```

9.8488578

MOD(A,B): Returns the MOD values. SQL> select MOD(10,3) from dual; // 1

POWER(A,B):
SQL> Select POWER(2,5) from dual; POWER(2,5)

32

CEIL: In Oracle SQL, the CEIL function is used to return the smallest integer greater than or equal to a specified numeric value. The syntax for the CEIL function is as follows:

CEIL(numeric_value);

```
SQL> Select CEIL(40.9) from dual; //41
```

This statement will return the value 41, which is the smallest integer greater than or equal to the input value of 40.9. If the input value is already an integer, the CEIL function returns the

same integer value without rounding. For example, CEIL(4) returns 4.

SQL> Select CEIL(40.2) from dual; //41

SQL> Select CEIL(40.5) from dual; //41

FLOOR: The FLOOR function in Oracle SQL is used to round a numeric value down to the nearest integer or specified decimal place.

SQL> Select FLOOR(40.9) from dual; //40 **SQL>** Select FLOOR(40.2) from dual; //40 **SQL>** Select FLOOR(40.5) from dual; //40
TRUNC:(TRUNCATE) Remove the decimal points.

SQL> Select TRUNC(40.9) from dual; // 40 **SQL>** Select TRUNC(40.2) from dual; // 40 **SQL>** Select TRUNC(40.5) from dual; // 40
SQL> Select TRUNC(40.1234,2) from dual; // 40.12 **SQL>** Select TRUNC(685.195364,3) from dual; // 685.195 **SQL>** Select TRUNC(6854,-1) from dual; // 6850
SQL> Select TRUNC(6854,-1) from dual; TRUNC(6854,-1)

6850

SQL> Select TRUNC(6854,-2) from dual; TRUNC(6854,-2)

6800

SQL> Select TRUNC(6854,-3) from dual; TRUNC(6854,-3)

6000

SQL> Select TRUNC(6854,-4) from dual; TRUNC(6854,-4)

0

SQL> Select TRUNC(6854,-2) from dual; // 6800 **SQL>** Select TRUNC(7777,-3)

from dual; // 7000 **ROUND:** Rounds of the nearest value.

SQL> Select ROUND(40.9) from dual; //41

SQL> Select ROUND(40.2) from dual; //40

SQL> Select ROUND(40.5) from dual; //41

```
SQL> Select ROUND(123.863,2) from dual;  
//123.86 SQL> Select ROUND(123.868,2) from  
dual; //123.87 SQL> Select  
ROUND(856.81766,3) from dual; //856.818  
SQL> Select ROUND(123,-1) from dual; //120  
SQL> Select ROUND(140,-2) from dual;  
//100 SQL> Select ROUND(127,-3) from  
dual; //130 SQL> Select ROUND(17763,-3)  
from dual; //18000 GREATEST:  
SQL> Select GREATEST (100,200,300) from dual; // 300
```

LEAST:

```
SQL> Select LEAST (100,200,300) from dual; // 100
```

SIGN: In Oracle Database, the SIGN() function returns the sign of a numeric expression, indicating whether it is positive, negative, or zero.

Syntax: SIGN(numeric_expression) FROM dual;

Ex: SQL> SELECT SIGN(7) AS sign_value FROM
dual; SIGN_VALUE

```
-----  
1
```

```
SQL> SELECT SIGN(-7) AS sign_value FROM dual;  
SIGN_VALUE
```

```
-----  
-1
```

```
SQL> SELECT SIGN(0) AS sign_value FROM dual;
```

```
SIGN_VALUE
```

```
-----  
0
```

```
SQL> SELECT SIGN(10) AS result1, SIGN(0) AS result2, SIGN(-5) AS result3
```

FROM dual; RESULT1 RESULT2 RESULT3

1 0 -1

C. Date functions:

They are four date functions.

- **ADD_MONTHS**
- **MONTHS_BETWEEN**
- **NEXT_DAY**
- **LAST_DAY**

ADD_MONTHS

HS:

ADD_MONTHS of months to the given date.

SQL> Select ADD_MONTHS(SYSDATE,12) from dual; // 16-JUN-13
SQL> Select ADD_MONTHS('11-APR-05',3) from dual; // 11-APR-05

MONTHS_BETWEEN:

Returns number of months b/w given the two months.

SQL> Select MONTHS_BETWEEN('11-JAN-05','11-JAN-04') from dual; // 12

SQL> Select MONTHS_BETWEEN('11-JAN-04','11-JAN-05') from dual; // -12

SQL> Select EMPNO, ENAME, SAL, HIREDATE from emp; //display emp table

SQL> Select EMPNO, ENAME, SAL, MONTHS_BETWEEN(SYSDATE,HIREDATE) from emp;

emp;

SQL> Select EMPNO, ENAME, SAL, MONTHS_BETWEEN(SYSDATE,HIREDATE)/12 from emp;

SQL> Select EMPNO, ENAME, SAL, ROUND(MONTHS_BETWEEN(SYSDATE,HIREDATE)/12) from emp;

SQL> Select EMPNO, ENAME, SAL, ROUND(MONTHS_BETWEEN(SYSDATE,HIREDATE)/12) EXP from emp;

CURRENT_DATE:

Return the current date and time in the session time zone. **SQL>** SELECT CURRENT_DATE FROM dual; CURRENT_D

17-AUG-23

CURRENT_TIMESTAMP:

Return the current date and time with time zone in the session

time zone **SQL> SELECT CURRENT_TIMESTAMP**
FROM dual; CURRENT_TIMESTAMP

17-AUG-23 02.47.10.399000 PM +05:30

NEXT_DAY: Returns date of the specified date.

SQL> Select NEXT_DAY(SYSDATE,'MONDAY') from dual; // 18-JUN-12

SQL> Select NEXT_DAY('11-JAN-05','MONDAY') from dual; // 17-JAN-05

LAST_DAY: Returns the last day of the month.

SQL> Select LAST_DAY(SYSDATE) from dual; // 30-JUN-12

SQL> Select LAST_DAY('11-FEB-05') from dual; // 28-FEB-05

D. Conversion functions:

Conversion functions are one data type to another data type conversion. They are three conversion functions.

1. **TO_CHAR**
2. **TO_NUMBER**
3. **TO_DATE**

1. **TO_CHAR:** This function is having two functionalities.

a. Number to_char:

This function is used only \$ or u-rows and number is used 9.

SQL> SELECT TO_CHAR(SYSDATE, 'YYYY-MM-DD')
FROM dual; TO_CHAR(SY

2023-04-17

SQL> SELECT TO_CHAR(SYSDATE, 'YYYY-DD-MM')
FROM dual; TO_CHAR(SY

2023-17-08

SQL> SELECT TO_CHAR(SYSDATE, 'DD')
FROM dual; TO

--

```
1
7
SQL> SELECT TO_CHAR(SYSDATE, 'MM')
FROM dual; TO
--
0
4
SQL> SELECT TO_CHAR(SYSDATE, 'YY') FROM dual;
```

TO

--
2
3

SQL> SELECT TO_CHAR(SYSDATE, 'Day: MONTH DD, YYYY')
FROM dual; TO_CHAR (SYSDATE,'DAY: MONTHDD, YYYY')

Thursday: AUGUST 17, 2023

SQL> select TO_CHAR(SYSDATE, 'Day: MONTH DD, YYYY',
'NLS_DATE_LANGUAGE = Spanish')
from dual;
TO_CHAR(SYSDATE,'DAY:MONTHDD,YYYY','NLS_DATE_LANGUAGE=
SPANISH')

Jueves : AGOSTO17, 2023

SQL> select TO_CHAR(SYSDATE, 'Day: MONTH DD, YYYY',
'NLS_DATE_LANGUAGE = English')
from dual;
TO_CHAR(SYSDATE,'DAY:MONTHDD,YYYY','NLS_DATE_LANGUAGE=
ENGLISH')

Thursday: AUGUST 17, 2023

b. Date to _char:

SQL> Select SYSDATE from
dual; SYSDATE

17-AUG-23

The output will display the current date in the default format of your Oracle database, which usually looks something like "17-AUG-23".

SQL> Select TO_CHAR(SYSDATE,'DD-MONTH-YY')
from dual; TO_CHAR(SYSDATE,'DD-MONTH-YY')

17-AUGUST -23

This query should give you the current date with the month's full name in uppercase, for example: "17- AUGUST-23".

SQL> Select TO_CHAR(SYSDATE,'DAY') from dual;

TO_CHAR(SYSDATE,'DAY')

THURSDAY

Returns the name of the day of the week for the current date. In your example, the query result is 'THURSDAY', which is correct for August 17, 2023.

```
SQL> Select TO_CHAR(SYSDATE,'YYYY')  
from dual; TO_C
```

--

2023

Returns the current year in four-digit format. In this case, it returns 2023, which represents the current year.

```
SQL> Select TO_CHAR(SYSDATE,'MM')  
from dual; TO
```

--

0

8

Returns the current month in two-digit format. In this case, it returns 08, which corresponds to August.

```
SQL> Select TO_CHAR(SYSDATE,'DDD')  
from dual; TO_
```

--

22

9

Will return the day of the year (1-366) for the current date. For example, if today's date is August 17, 2023, the query would return the value 229, which corresponds to the 229th day of the year.

```
SQL> Select TO_CHAR(SYSDATE,'DD') from  
dual; TO
```

--

1

7

```
SQL> Select TO_CHAR(SYSDATE,'MON') from  
dual; TO_CHAR(SYSD
```

AUG

```
SQL> Select TO_CHAR(SYSDATE,'DY') from dual;  
TO_CHAR(SYSD
```

THU

SQL> Select TO_CHAR(SYSDATE,'DD-MM-YY HH:MI:SS')
from dual; TO_CHAR(SYSDATE,'

17-08-23 10:53:32

This query will give you the current date and time in the format "17-08-23 12:34:56".

SQL> Select * from emp where TO_CHAR(HIREDATE,'YYYY') = '1981';

SQL> Select * from ss2 where TO_CHAR(dob,'YY') = '22';

DOB

10-JAN-22

15-JAN-22

SQL> Select * from emp where TO_CHAR(HIREDATE,'YYYY') = '1980';

SQL> Select * from emp where TO_CHAR(HIREDATE,'MON') = 'DEC';

SQL> Select eno,ename,hiredate from emp;

SQL> Select eno,ename, TO_CHAR(HIREDATE,'DD-MM-YY') from emp;

SQL> Select eno,ename, TO_CHAR(HIREDATE,'DD-MM-YYYY') from emp;

SQL> Select * from ss2 where TO_CHAR(dob,'mon') = 'jan';

DOB

10-JAN-22

15-JAN-22

2. TO NUMBER:

SQL> Select TO_NUMBER(LTRIM('\$1400','\$')) + 10 from dual;

TO_NUMBER(LTRIM('\$1400','\$'))+10

1410

The query you provided is attempting to remove the dollar sign ('\$') from the string "\$1400", convert the remaining number to a numeric format, and then add 10 to it.

SQL> SELECT TO_NUMBER('\$12345.678', '\$999999.999') FROM dual;

TO_NUMBER('\$12345.678','\$999999.999')

12345.678

If the number is within the scope of the format, it is correct, otherwise it is wrong.

SQL> Select To_number (' 000012134 ') from dual;
TO_NUMBER('000012134')

12134

The query, to convert the string ' 000012134 ' to a numeric value using the TO_NUMBER function. The trimmed string '000012134' is then converted to the numeric value 12134.

```
SQL> SELECT TO_NUMBER('97.13') + 25.5 FROM dual;  
TO_NUMBER('97.13')+25.5
```

122.63

Convert the string 97.13 to a number using TO_NUMBER () and then adds 25.5 to that number.

```
SQL> SELECT TO_NUMBER ('-$12,345.67', '$99,999.99')  
FROM dual; TO_NUMBER ('-$12,345.67','$99,999.99')
```

-12345.67

Converts the string -\$12,345.67 to a number, passing the format string \$99,999.99 to TO_NUMBER (): TO_NUMBER.

3. TO_DATE:

This function is used to convert character values to data value. **SQL>** select TO_DATE('2003/07/09', 'yyyy/mm/dd') from dual; TO_DATE('

09-JUL-03

The query SELECT TO_DATE('2003/07/09', 'yyyy/mm/dd') FROM dual; is used to convert the date string '2003/07/09' into an actual date using the TO_DATE function. The result of this query will be the date '2003- 07-09' in Oracle's standard date format.

```
SQL> select TO_DATE('070903', 'MMDDYY') from  
dual; TO_DATE('
```

09-JUL-03

```
SQL> select TO_DATE('20020315', 'yyyymmdd') from
```

dual; TO_DATE('

15-MAR-02

SQL> SELECT TO_DATE('2015/05/15 8:30:25', 'YYYY/MM/DD HH:MI:SS') FROM dual;

TO_DATE('

15-MAY-15

This would convert the string value of 2015/05/15 8:30:25 to Oracle's standard date format. Ex: ADD_MONTHS

SQL> SELECT ADD_MONTHS('11-JAN-05', 2) FROM dual;

ADD_MONTH

11-MAR-05

In this query, '11-JAN-05' is the input date, and 2 is the number of months you want to add to it. The ADD_MONTHS function will correctly calculate the new date by adding 2 months to '11-JAN-05'.

SQL> SELECT ADD_MONTHS(TO_DATE('11-JANUARY-2005 01:45 p.M.', 'DD-MONTH-YYYY HH:MI A.M.'), 4) FROM dual;

ADD_MONTH

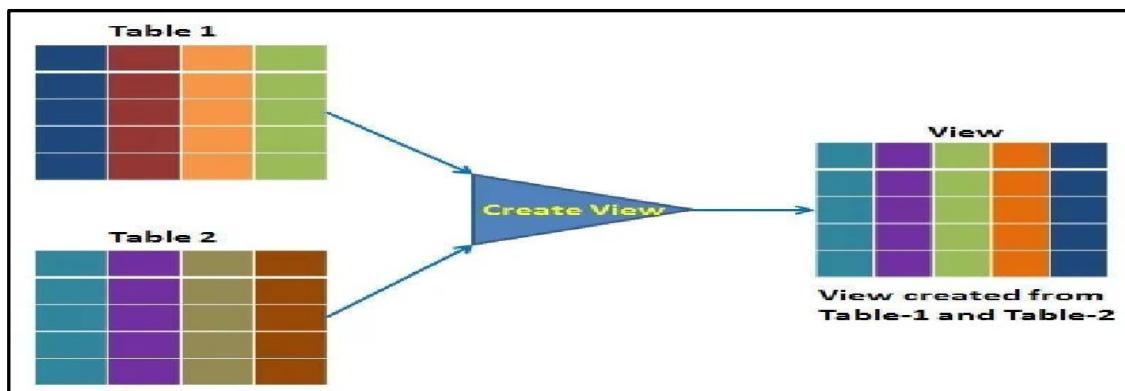
11-MAY-05

Module-7: SQL VIEWS and SQL INDEXES

Views (Virtual Tables) in SQL:

Views are a special version of tables in SQL. They provide a virtual table environment for various complex operations. You can select data from multiple tables, or you can select specific data based on certain criteria in views. It does not hold the actual data; it holds only the definition of the view in the data dictionary. A view is a logical representation of data from one or more than one table. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database. Views in SQL are kind of virtual tables. A view also has rows and columns as they are in a real table in the database. We can create a view by selecting fields from one or more tables present in the database. A View can either have all the rows of a table or specific rows based on certain condition. We can create View using CREATE VIEW statement. A View can be created from a single table or multiple tables.

Note: A view always shows up-to-date data! The database engine recreates the view, every time a user query it.



1. Create View:

We can create View using CREATE VIEW statement. A View can be created from a single table or multiple tables.

Syntax:

```
CREATE VIEW view_name AS SELECT column1, column2, ... FROM table_name  
WHERE condition;
```

view_name: Name for the View

table_name: Name of the table

condition: Condition to select rows

SQL> create view v1 as select sup_name,city from supplier; View created.

SQL> desc v1;

Name	Null?	Type
SUP_NAME		VARCHAR2(20)
CITY		VARCHAR2(15)

2. Insert View:

You can insert rows into a view only if the view is modifiable and contains no derived columns. The reason for the second restriction is that an inserted row must provide values for all columns, but the database server cannot tell how to distribute an inserted value through an expression.

```
SQL> insert into v1  
values('S0','Mohan'); 1 row created.
```

```
SQL> select * from  
v1;
```

```
SU SUP_NAME  
-----
```

```
S1 murali
```

```
S2 Kiran
```

```
S3
```

```
Mohan
```

```
S4
```

```
Ramesh
```

```
S5
```

```
Manish
```

```
S6
```

```
Srikanth
```

```
S7 Suresh
```

```
S8 Anil
```

```
S0
```

```
Mohan
```

```
S9
```

```
Charan
```

Ex: Insert into InsertDept(DeptID, DeptName, Place) Values(50, 'ADMINISTRATION', 'DELHI');

3. Update VIEW:

You can modify the definition of an Oracle VIEW without dropping it by using the Oracle CREATE OR REPLACE VIEW Statement.

Syntax: CREATE OR REPLACE VIEW view_name AS SELECT columns FROM table WHERE conditions;

SQL> CREATE or REPLACE VIEW v1 AS SELECT sup_no,sup_name FROM supplier; View created.

SQL> desc v1;

Name	Null?	Type
------	-------	------

SUP_NO	NOT NULL
--------	----------

VARCHAR2(2)	SUP_NAME
	VARCHAR2(20)

4. Drop VIEW:

Once an Oracle VIEW has been created, you can drop it with the Oracle DROP VIEW Statement.

Syntax: DROP VIEW view_name;

SQL> DROP VIEW v1;

View dropped.

Types of Views: They are different types

- Simple (Single) views
- Complex views
- Read only views
- With check option views
- Materialized views

1. Simple (Single) Views:

A view based on only a single table, which doesn't contain GROUP BY clause and any functions. These views can only contain a single base table or can be created only from one table. Group functions such as MAX(), COUNT(), etc., cannot be used here, and it does not contain groups of data. By using Simple View, DML operations can be performed. Insert, delete, and update are directly possible, but Simple View does not contain group by, like rownum, distinct, columns defined by expressions. Simple view also does not include NOT NULL columns from the base tables. when view is created using one base table it is called as Single view.

Syntax: Create view<view_name> as <select STMT>;

SQL> Create view v1 as select sup_no,sup_name from supplier; View created.

SQL> Select * from v1;

SU SUP_NAME

S1 murali

S2 Kiran

S3

Mohan

S4

Ramesh

S5

Manish

S6

Srikanth

S7 Suresh

S8 Anil

S9 Charan

9 rows selected.

- View does not contain any data.
- View does not consume memory location.
- We can perform DML operations on simple views.
- Any DML operation performed on simple view will be reflected on base table.

Select * from TAB: Will gives list of all the tables and view which are the data base tables.

Ex: SQL> Select * from TAB;

TNAME	TABTYPE	CLUSTERID
-------	---------	-----------

CV1	VIEW
STUDENT	TABLE
SUPPLIER	TABLE
V1	VIEW
V3	VIEW
V9	VIEW

To see the structure of the table.

SQL> desc v1;

Name	Null?	Type
SUP_NO	NOT NULL	VARCHAR2(2)
SUP_NAME		VARCHAR2(20)Name Null Type

Query to see only views:

SQL> Select view_name from user_views;

VIEW_NAME

CV

1

V1

V3

V9

In user_view from the database we get list of all the view and corresponding select statement used for the view.

SQL> Select view_name, Text from
user_views; VIEW_NAME TEXT

```
V1      SELECT sup_no,sup_name FROM supplier
V2      select sup_no,sup_name,city from supplier
V9      SELECT sup_name, city, SUM(iteam_price) AS iteam FROM supplier GROUP BY
sup_name
```

2. Complex View:

These views can contain more than one base table or can be constructed on more than one base table, and they contain a group by clause, join conditions, an order by clause. Group functions can be used here, and it contains groups of data. Complex views cannot always be used to perform DML operations. Insert, delete, and update cannot be applied directly on complex views. But unlike Simple Views, Complex Views can contain group by, pseudocolumn like rownum, distinct, columns defined by expressions. When a view is created using multiple base tables it is called Complex view.

- **Create View:** The CREATE VIEW statement is used to define and create a view in a database.

Syntax:

```
SQL> Create view viewname As select s.table1column1, s.table1column2,
t.table2column1 t.table2column2 from table1name s, table2name t;
```

```
SQL> Create view CV2 As select s.sup_no, s.sup_name, t.sid, t.sname from supplier s,
student t;
```

View created.

```
SQL> select * from cv2;
```

In the above query we will select SUP_NO and SUP_NAME from SUPPLIER table and SID and SNMAE from STUDENT table. In the output you can see that each row of the table Supplier is joined with every row of the table Student. The total rows in the result-set = 8* 7 = 56.

SU SUP_NAME SID SNAME

```
-----
S1 Suresh      1 Supraja
S2 Kiran       1 Supraja
S3 Mohan       1 Supraja
S4 Ramesh      1 Supraja
```

S5 Manish	1 Supraja
S6 Srikanth	1 Supraja
S0 Mohan	1 Supraja
S7 Krishna	1 Supraja
S1 Suresh	2 Avinash
S2 Kiran	2 Avinash
S3 Mohan	2 Avinash
S4 Ramesh	2 Avinash
S5 Manish	2 Avinash

S6 Srikanth	2 Avinash
S0 Mohan	2 Avinash
S7 Krishna	2 Avinash
S1 Suresh	3 Venkatesh
S2 Kiran	3 Venkatesh
S3 Mohan	3 Venkatesh
S4 Ramesh	3 Venkatesh
S5 Manish	3 Venkatesh
S6 Srikanth	3 Venkatesh
S0 Mohan	3 Venkatesh
S7 Krishna	3 Venkatesh
S1 Suresh	4 Eswar
S2 Kiran	4 Eswar
S3 Mohan	4 Eswar
S4 Ramesh	4 Eswar
S5 Manish	4 Eswar
S6 Srikanth	4 Eswar
S0 Mohan	4 Eswar
S7 Krishna	4 Eswar
S1 Suresh	5 Divya
S2 Kiran	5 Divya
S3 Mohan	5 Divya
S4 Ramesh	5 Divya
S5 Manish	5 Divya
S6 Srikanth	5 Divya
S0 Mohan	5 Divya
S7 Krishna	5 Divya
S1 Suresh	12 John
S2 Kiran	12 John
S3 Mohan	12 John

S4 Ramesh 12 John

S5 Manish 12 John

S6 Srikanth	12 John
S0 Mohan	12 John
S7 Krishna	12 John
S1 Suresh	6 krishna
S2 Kiran	6 krishna
S3 Mohan	6 krishna
S4 Ramesh	6 krishna
S5 Manish	6 krishna
S6 Srikanth	6 krishna
S0 Mohan	6 krishna
S7 Krishna	6 krishna

56 rows selected.

SQL> Create view cv1 As select s.sup_no, s.sup_name, s.city, t.sid, t.sname from supplier s, student t; View created.

SQL> select * from cv1;

OUTPUT:

SU	SUP_NAME	CITY	SID	SNAME
S1	murali	Hyderabad	1	Krishna
S2	Kiran	Delhi	1	Krishna
S3	Mohan	Delhi	1	Krishna
S4	Ramesh	Bangalore	1	Krishna
S5	Manish	Mumbai	1	Krishna
S6	Srikanth	Chennai	1	Krishna
S7	Suresh	Hyderabad	1	Krishna
S8	Anil	HYD	1	Krishna
S9	Charan	SEC	1	Krishna

9 rows selected.

- **Insert View:** An "insert" operation in a database is used to add new rows of data to a table. The INSERT statement is typically used for this purpose.

```
insert into Test_v7 values(7878,'ravi',9000,40,'HR','HYD'); // Error
```

Different between simple and complex Views:

Simple view	Complex view
Contains only one single base table or is created from only one table	Contains more than one base table or is created from more than one table.
We cannot use group functions like MAX(), COUNT(), etc.	We can use group functions.
DML operations could be performed through a simple view.	DML operations could not always be performed through a complex view.
INSERT, DELETE and UPDATE are directly possible on a simple view.	We cannot apply INSERT, DELETE and UPDATE on complex view directly.
Simple view does not contain group by, distinct, column like rownum, columns defined by expressions.	It can contain group by, distinct, like rownum, columns defined by expressions.
Example: CREATE VIEW Employee AS SELECT Empid, Empname FROM Employee WHERE Empid = '030314';	Example: CREATE VIEW EmployeeByDepartment AS SELECT e.emp_id, d.dept_id, e.emp_name FROM Employee e, Department d WHERE e.dept_id=d.dept_id;

3. Read Only View:

If view is created with “WITH READ ONLY” no data manipulation is allowed in any condition. Only selects are allowed against the views. A view is read-only if it is not deletable, updatable, or insertable.

SQL> Create view v5 As select sup_no,sup_name,city from supplier with read only; View created.

SQL> select * from v5;

SU SUP_NAME CITY

S1 murali Hyderabad

S2 Kiran Delhi

S3 Mohan Delhi

S4 Ramesh Bangalore

S5 Manish Mumbai

S6 Srikanth Chennai

S7 Suresh Hyderabad

S8 Anil

HY

D S0 Mohan

S9 Charan SEC

10 rows selected.

```
SQL> insert into v5
values('S0','Mohan','KMM'); insert into v5
values('S0','Mohan','KMM')
* ERROR at line 1:
```

ORA-42399: cannot perform a DML operation on a read-only view

4. With Check Option View:

The WITH CHECK OPTION clause is used for an updatable view to prohibits the changes to the view that would produce rows which are not included in the defining query. In Oracle, the WITH CHECK OPTION clause is used when creating a view to ensure that any data modification (insert, update, or delete) performed through the view adheres to the view's filter conditions.

Syntax:

```
CREATE VIEW view_name AS SELECT * FROM table_name WHERE condition;
```

```
SQL> CREATE VIEW V6 AS SELECT sup_no, sup_name, iteam_supplied, iteam_price
FROM supplier WHERE iteam_price > 5000 WITH CHECK OPTION;
```

View created.

```
SQL> select * from v6;
```

SU	SUP_NAME	ITEAM_SUPPLIED	ITEAM_PRICE
S2	Kiran	Processor	8000
S4	Ramesh	Processor	9000
S5	Manish	Printer	6000
S6	Srikanth	Processor	8500
S8	Anil	LAPTO	5500
S9	Charan	System	10000

S2	Kiran	Processor	8000
S4	Ramesh	Processor	9000
S5	Manish	Printer	6000
S6	Srikanth	Processor	8500
S8	Anil	LAPTO	5500
S9	Charan	System	10000

6 rows selected.

These views will allow DML operation only when where condition is satisfied

```
SQL> Insert into v6
values('SS','AAA','KKK',4000); Insert into v6
values('SS','AAA','KKK',4000)
```

***ERROR at line 1:**

ORA-01402: view WITH CHECK OPTION where-clause violation

5. Materialized view:

Materialized views are the views whose contents are computed and stored. Materialized views are also a logical virtual table, but in this case the result of the query is stored in the table or the disk. The performance of the materialized views is better than normal views. This is because the data is stored on the disk. A materialized view is a database object that contains the results of a query. The FROM clause of the query can name tables, views, and other materialized views. Use the CREATE MATERIALIZED VIEW statement to

create a materialized view. A materialized view is a particular type of database object that contains any results derived from a query. In computing, a materialized view is a database object that contains the results of a query. For example, it may be a local copy of data located remotely, or may be a subset of the rows and/or columns of a table or join result, or may be a summary using an aggregate function.

For example, let's say you have a database with two tables: one contains the number of employees in your business, and the other contains the number of departments in your business. Using a materialized view, you could query the database to retrieve all the employees who are associated with a particular department.

Syntax: Create MATERIALIZED view <VIEW_NAME> AS <select STMT>; **SQL>** Create materialized view MV1 As select sup_no,sup_name,city from supplier; Materialized view created.
SQL> select * from mv1;

SU SUP_NAME CITY

S1 murali Hyderabad
S2 Kiran Delhi
S3 Mohan Delhi
S4 Ramesh Bangalore
S5 Manish Mumbai
S6 Srikanth Chennai
S7 Suresh Hyderabad
S8 Anil HYD
S9 Mohan
S9 Charan SEC

10 rows selected.

INDEX

An index is a database structure that provides quick lookup of data in a column or columns of a table. Indexes provide faster access to data for operations that return a small portion of a table's rows. Although Oracle allows an unlimited number of indexes on a table, the indexes only help if they are used to speed up queries. In Oracle, an index is a database object that creates and stores records for all values in specific columns or clusters. With the help of indexes, users can access the necessary data portion much faster and easier. Indexes are among the most popular means of Oracle database performance tuning. They are applicable in SELECT statements, where the usage of indexes can improve the overall performance significantly. Indexes are independent of the tables – you can create or drop them whenever you need – it won't affect the tables and other indexes. When you alter the table by inserting or deleting rows, all indexes related to that table must be updated. And any change in the indexed column also requires updating the index.

Create a Normal Index in Oracle:

To create a new index, we need to apply the **CREATE INDEX** command with the below syntax: **Syntax:** CREATE INDEX index_name ON table_name (column1, column2, ...columnN); **Example:** SQL> create index i1 on supplier(sup_no,sup_name);

Index created.

The parameters are as follows:

index_name – the name of the index you'll create.

table_name – the name of the table for which you are creating that index.

column1, column2, ... columnN – the table columns to include in that index.

If you want to make Oracle create table unique index, you should modify that basic command syntax in the following way:

CREATE [UNIQUE] INDEX index_name ON table_name (column1, column2, ...columnN);

In this statement, the keyword UNIQUE defines that the indexed columns must contain unique combinations of values.

Rename an index in Oracle:

Quite often, we need to change the name of the index. This operation is common in Oracle. If the user has the ALTER right for an index (if not, check it with the administrators), the task is straightforward.

The Oracle alter index rename command is as follows:

Syntax: ALTER INDEX current_index_name RENAME TO
new_index_name; In this statement,
current_index_name specifies the name of an existing index we want to
rename new_index_name specifies a new name of an existing index
SQL> ALTER INDEX i1 RENAME TO index1;

Index altered.

Drop an index in Oracle:

The DROP INDEX command in Oracle allows the users to delete any existing index from the current database schema. It won't affect the table physically because indexes are independent objects and are stored separately. Still, check if you might still use that index in some queries. In that case, the queries using the dropped index will take longer.

Syntax: `DROP INDEX index_name;`

`index_name` specifies the name of the index you want to delete

`schema_name` specifies an optional parameter. If it is absent, Oracle considers that you want to delete an index from your current schema.

SQL> `drop index i2;`

Index dropped.

Index is an object which is used to improve the performance of select statements.

Types of Indexes:

1. Simple Index:

A simple index is an index on a single column. The key values are sorted in ascending order in the index, which allows for efficient searching and retrieval of data. When a query is executed that involves a column included in a simple index, Oracle can use the index to locate the desired data more quickly, reducing the amount of disk I/O and improving the query performance. The index structure essentially acts as a roadmap to find the relevant rows in the table. Note that an index can only cover one table. We cannot build an index that covers multiple tables. A simple index in Oracle is a B-Tree index, which is the most common and basic type of index. It organizes data in a balanced tree structure, allowing efficient lookup and range scans. Let's see an example of creating a simple index in Oracle. When index is created on one column it is called as simple index.

Syntax: `CREATE INDEX <INDEX_NAME> ON <TABLE_NAME> (COL_NAME);`

SQL> `Create index IDX1 on supplier(sup_name);`

Index created.

This statement creates a simple index named idx1 on the `sup_name` column of the `supplier` table. Once the index is created, Oracle automatically maintains it as data in the table is inserted, updated, or deleted. Index should be created on columns which we regularly use in the `where` clause. When an index is created a separate structure is created with first column is `ROWID` and second column as indexed column. The Rows in the index will be arranged in the ascending order of indexed column.

2. Composite Index:

In Oracle, a composite index refers to an index that is created on multiple columns of a database table. It is also known as a multi-column index or concatenated index. A composite index allows you to speed up the retrieval of data when you have queries that involve multiple columns in the WHERE clause. By creating a composite index on multiple columns, Oracle creates a logical ordering of the data based on those columns. This logical ordering can significantly improve the performance of queries that involve the indexed columns. When a query matches the leading columns of the composite index, Oracle can quickly locate the desired rows.

To create a composite index in Oracle, you specify the names of multiple columns in the CREATE INDEX statement, separated by commas. Here's an example of creating a composite index on two columns, "column1" and "column2," of a table named "my_table":

To create a composite index on the "sup_no" and "iteam_price" columns of the "supplier" table, you can use the following syntax:

Syntax: CREATE INDEX idx_sup ON supplier(sup_no, iteam_price);

Index created.

This command creates a composite index named " idx_sup " on the " sup_no " and " iteam_price " columns of the "supplier" table.

Disadvantages of index:

Index will consume memory. The performance of DML command will be decreased.

3. Unique index:

In Oracle, a unique index is an index that enforces the uniqueness of values in one or more columns of a table. **It ensures that no two rows in the table can have the same values in the indexed columns.** If a unique index is defined on a column or set of columns, Oracle automatically checks for duplicate values when inserting or updating data in the table and throws an error if a duplicate is detected. A unique index ensures that no two rows of a table have duplicate values in the indexed column (or columns).

Here's an example of creating a unique index in Oracle:

Let's say we have a table called "Employees" with columns "EmployeeID" and "Email":

SQL> CREATE TABLE Employees (EmployeeID NUMBER, Email VARCHAR2(100));

Table created.

To create a unique index on the "Email" column, you can use the following syntax:

SQL> CREATE UNIQUE INDEX idx_employees_email ON Employees(Email);

Index created.

This statement creates a unique index named "idx_employees_email" on the "Email" column of the "Employees" table.

Once the unique index is created, it ensures that the values in the "Email" column are unique. If an attempt is made to insert or update a row with a duplicate value in the "Email" column, Oracle will raise an error.

For example, let's insert some data into the table:

SQL> INSERT INTO Employees (EmployeeID, Email)VALUES
(1,'krishna@example.com'); 1 row created.

SQL> INSERT INTO Employees (EmployeeID, Email) VALUES (2,
'pavan@example.com'); These two statements will execute successfully since the
values in the "Email" column are unique. Now, if we try to insert a row with a
duplicate email value:

SQL> INSERT INTO Employees (EmployeeID, Email) VALUES (3,
'krishna@example.com');

Oracle will raise an error because the unique index constraint is violated. Unique indexes are useful when you want to enforce uniqueness on columns that do not qualify as primary keys but still need to have unique values within the table.

4. Non-unique index:

In Oracle, a non-unique index is an index that allows duplicate values to exist. It means that multiple rows in the indexed column(s) can have the same value. Non-unique indexes are commonly used when there is a need to improve the performance of queries that involve searching or filtering data. If an index column contains duplicated values they are called as non-unique index. Now, let's create a non-unique index on the "Name" column:

```
SQL> CREATE INDEX idx_name ON student(sname);
```

Index created.

With this index in place, you can have multiple rows with the same name value.

```
SQL> insert into student values(2,'krishna',200,50);
```

1 row created.

```
SQL> insert into student values(3,'krishna',200,50);
```

1 row created.

Query to see list of all the indexes:

```
SQL> Select index_name, table_name from user_indexes;
```

INDEX_NAME	TABLE_NAME
SYS_C007001	SUPPLIER
IDX_SUP_NAME	SUPPLIER
IDX_SUP	SUPPLIER
INDEX1	SUPPLIER
IDX_NAME	STUDENT
SYS_C007035	STUDENT
SYS_C007069	MV2
SYS_C007066	MV1
IDX_EMPLOYEES_EMAIL	EMPLOYEES

To drop on index:

DROP INDEX statement is used to remove an existing index from a table.

SQL> Drop INDEX IDX_SUP;

Index dropped.

Module-8: SQL SUBQUERIES and SQL Joins SUBQUERIES

Subqueries are queries that are nested within another SQL query. A subquery can contain more than one query inside it, one after another. They allow us to select specific rows that satisfy certain conditions at the run time. They are also known as the inner query or inner select, and the query that contains them is known as the outer query or outer select. A Subquery or Inner query or Nested query is a query within another SQL query and embedded within the WHERE clause. A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

- ✓ Subqueries are nested inside statements like SELECT, INSERT, UPDATE, DELETE, or any other subquery.
- ✓ Subqueries are present in the WHERE clause, FROM clause, or HAVING clause of the PARENT SQL query.
- ✓ They are used with comparison operators and logical operators like $>$, $<$, \geq , \leq , \neq , SOME, ANY, ALL, IN and BETWEEN etc..
- ✓ They execute before the outer query at the run time and pass the result to complete the statement.
- ✓ Subqueries are used to compare an expression to the output and check if any row gets selected.

There are a few rules that subqueries must follow:

- Subqueries must be enclosed within parentheses.
- A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
- An ORDER BY cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY can be used to perform the same function as the ORDER BY in a subquery.
- Subqueries that return more than one row can only be used with multiple value operators, such as the IN operator.
- A subquery cannot be immediately enclosed in a set function.

Note:

→ Subqueries should always place in the inside.

→ Subqueries are executed first and then parent query is executed by using the result of subquery.

a. Single Row Subquery:

Subqueries that return a single row as an output to their parent query are called single-row subqueries. Single- row subqueries are used in a SQL SELECT statement with HAVING

clause, WHERE clause, or a FROM clause and a comparison operator. Single-row subqueries are used in the SELECT statement. When subquery returns one row (1 value). It is called Single Row Subquery.

Let's see it with an example.

Ex: `SELECT * FROM student WHERE SID IN (SELECT SID FROM student WHERE sub1 > 50);`

OUTPUT:

SID	SNAME	SUB1	SUB2	SUB3	TOTAL	AVERAGE
1	krishna	70	80	90		

2	Sandeep	90	80	100		
3	keerthi	70	50	80		
5	Ramu	55	65	75	200	30

b. Level Two query:

Subqueries that return multiple rows as an output to their parent query are called multiple-row subqueries. Multiple row subqueries can be used in a SQL SELECT statement with a HAVING clause, WHERE clause, a FROM clause, and a logical operator (ALL, IN, NOT IN, and ANY). Let's explore it using an example.

SQL> SELECT sid,sname FROM student WHERE SID IN (SELECT SID FROM student WHERE sub1 > 50) and sname IN (SELECT sname FROM student WHERE sub2 < 60);

OUTPUT:

SID SNAME

3 keerthi

c. Level Three query:

SQL> SELECT sid,sname FROM student WHERE SID IN (SELECT SID FROM student WHERE sub1 > 50) and sid IN (SELECT sid FROM student WHERE sub2 > 60) and sid IN (SELECT sid FROM student WHERE sub3 > 90);

OUTPUT:

SID SNAME

2 Sandeep

Note: The above query is three level query.

- ✓ A subquery can also be nested inside INSERT, UPDATE, and DELETE statements. Subqueries must be enclosed within parentheses. Apart from the above type of subqueries, you can use a subquery inside INSERT, UPDATE and DELETE statement.
- ✓ Subqueries with the SELECT Statement:
- ✓ Subqueries are most frequently used with the SELECT statement.

- ✓ Sub query can be nested upto 32 levels.

The basic syntax is as follows:

SELECT column_name [, column_name] FROM table1 [, table2] WHERE column_name
OPERATOR

Example:

Consider the **STUDENT** table having the following records:

Now, let us check the following subquery with SELECT statement: This would produce the following result:

SQL> SELECT * FROM student WHERE SID IN (SELECT SID FROM student WHERE sub1 > 50);

SID	SNAME	SUB1	SUB2	SUB3	TOTAL	AVERAGE
------------	--------------	-------------	-------------	-------------	--------------	----------------

1	krishna	70	80	90		
2	Sandeep	90	80	100		
5	keerthi	70	50	80		
5	Ramu					

1) Subqueries with the INSERT Statement:

Subqueries also can be used with INSERT statements. The INSERT statement uses the data returned from the subquery to insert into another table. The selected data in the subquery can be modified with any of the character, date or number functions.

The basic syntax is as follows:

```
INSERT INTO table_name [ (column1 [, column2 ]) ] SELECT [ *|column1 [, column2 ] FROM table1 [, table2 ] [ WHERE VALUE OPERATOR ]
```

Example:

Consider a table STUDENT2 with similar structure as STUDENT table. Now to copy complete STUDENT table into STUDENT2:

SQL> INSERT INTO STUDENT2 SELECT * FROM STUDENT WHERE ID IN (SELECT ID FROM STUDENT);

5 rows created.

SQL> SELECT * FROM STUDENT2;

SID	SNAME	SUB1	SUB2	SUB3
------------	--------------	-------------	-------------	-------------

1	Krishna	50	60	70
2	Hemima	80	90	70
3	Jaya	40	50	70
4	Malleswari	90	60	70
5	Supriya	20	30	50

2) Subqueries with the UPDATE Statement:

The subquery can be used in conjunction with the UPDATE statement. Either single or multiple columns in a table can be updated when using a subquery with the UPDATE statement.

The basic syntax is as follows:

UPDATE table SET column_name = new_value [WHERE OPERATOR [VALUE]
(SELECT COLUMN_NAME FROM TABLE_NAME) [WHERE]] Example:

Assuming, we have STUDENT5 table available which is backup of STUDENT table.

SQL> UPDATE STUDENT SET SUB1 = SUB1 + 10 WHERE SUB2 IN (SELECT SUB2 FROM STUDENT5 WHERE SUB2 <= 40);

1 row updated.

SQL> SELECT * FROM STUDENT;

SID	SNAME	SUB1	SUB2	SUB
1	Krishna	50	60	70
2	Hemima	80	90	70
3	Jaya	40	50	70
4	Malleswari	90	60	70
5	Supriya	40	30	50

3) Subqueries with the DELETE Statement:

The subquery can be used in conjunction with the DELETE statement like with any other statements mentioned above.

Syntax:

DELETE FROM TABLE_NAME [WHERE OPERATOR [VALUE] (SELECT COLUMN_NAME FROM TABLE_NAME) [WHERE]]

Example:

Assuming, we have CUSTOMERS_BKP table available which is backup of CUSTOMERS table.

Following example deletes records from CUSTOMERS table for all the customers whose AGE is greater than or equal to 27:

SQL> DELETE FROM STUDENT WHERE SUB1 IN (SELECT SUB1 FROM STUDENT5 WHERE SUB1

= 55);

1 row deleted.

SQL> SELECT * FROM STUDENT;

SID	SNAME	SUB1	SUB2	SUB3
1	Krishna	50	60	70

2	Hemima	80	90	70
3	Jaya		40	50
4	Malleswari	90	60	70
5	Supriya	40	30	50

JOINS

Joins are used to retrieve the data from multiple tables. The SQL Joins clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each. Here, it is noticeable that the join is performed in the WHERE clause. Several operators can be used to join tables, such as =, <, >, <>, <=, >=, !=, BETWEEN, LIKE, and NOT; they can all be used to join tables. However, the most common operator is the equal symbol.

SQL Join Types:

There are different types of joins available in SQL:

1. **INNER JOIN (EQUI JOIN and NON-EQUIJOIN)**
2. **Left Join**
3. **Right Join**
4. **Full Join**
5. **Cross Join**
6. **Self-Join**

1. INNER JOIN:

a. EQUIJOIN:

The most frequently used and important of the joins is the INNER JOIN. They are also referred to as an EQUIJOIN. The INNER JOIN creates a new result table by combining column values of two tables (table1 and table2) based upon the join-predicate. The query compares each row of table1 with each row of table2 to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row.

Syntax:

The basic syntax of INNER JOIN is as follows:

```
SELECT table1.column1,table2.column2...FROM table1 INNER JOIN table2  
ON table1.common_filed = table2.common_field;
```

Example:

Consider the following two tables,

- i. **CUSTOMERS** table is as follows:

SQL> select * from customers;

ID	NAME	AGE
5	Sandeep	30
1	murali	30
2	kiran	30

3 kiran 30

4 25

ii. Another table is **ORDERS** as follows:

SQL> select * from orders;

OID	PNAME	ID	AMOUNT
-----	-------	----	--------

1	PC	1	50000
2	Mobile	2	70000
3	tab	5	30000
4	TV	9	20000

Now, let us join these two tables using INNER JOIN as follows:

Example:

SQL> SELECT CUSTOMERS.ID, CUSTOMERS.NAME, CUSTOMERS.AGE
FROM CUSTOMERS INNER JOIN ORDERS ON CUSTOMERS.ID =

ORDERS.ID	NAME	AGE
-----------	------	-----

5	Sandeep	30
1	Murali	30
2	Kiran	30

SQL> select empno, ename, dname from emp,

OUTPUT:

EMPNO	ENAME	DNAME	dept where emp.deptno = dept.deptno;
7369	SMITH	RESEARCH	
7499	ALLEN	SALES	
7521	WARD	SALES	

NON-EQUIJOIN:

b.

Non-Equi Join is also a type of INNER Join in which we need to retrieve data from multiple tables. Non-Equi Join matches the column values from different tables based on an inequality based on the operators like

<,>,<=,>=,! = , BETWEEN, etc. Non-equi joins are joins whose join conditions use conditional operators other than equals. An example would be where we are matching first name and then last name, but we are checking where one field from a table does not equal field from another table. Non-Equi Join in SQL retrieves data using any operator or condition except the equality condition. The value of the column in each row from the source table is compared with the corresponding value of the target table. If the data matches in the source and target table, the comparison returns true, and therefore that data is retrieved from the table. NON EQUI JOIN performs a JOIN using comparison operator other than equal(=) sign like >,<,>=,<= with conditions.

Syntax: **SELECT * FROM TableName1, TableName2 WHERE**
TableName1.columnName [> | < | >=
| <= | != | BETWEEN] table_name2.column;

SQL> SELECT CUSTOMERS.ID, CUSTOMERS.NAME, CUSTOMERS.AGE,
ORDERS.OID, ORDERS.PNAME, ORDERS.ID, ORDERS.AMOUNT FROM
CUSTOMERS INNER JOIN ORDERS ON CUSTOMERS.ID != ORDERS.ID;

ID	NAME	AGE	OID	PNAME	ID	AMOUNT
2	Hemima	25	1	Computer	1	50000
3	pavan	30	1	Computer	1	50000
4	Supriya	25	1	Computer	1	50000
5	jaya	28	1	Computer	1	50000
1	krishna	30	2	LapTop	2	65000
3	pavan	30	2	LapTop	2	65000
4	Supriya	25	2	LapTop	2	65000
5	jaya	28	2	LapTop	2	65000
5	jaya	30	3	Mobile	3	15000
1	krishna	25	3	Mobile	3	15000
2	Hemima	25	3	Mobile	3	15000
4	Supriya	28	3	Mobile	3	15000
5	jaya	30	4	TV	9	25000
5	jaya	25	4	TV	9	25000
1	krishna	30	4	TV	9	25000
2	Hemima	25	4	TV	9	25000
3	pavan	28	4	TV	9	25000
4	Supriya					
5	jaya					
1	krishna	30	5	tab	8	10000
2	Hemima	25	5	tab	8	10000

3 pavan	30	5 tab	8	10000
4 Supriya	25	5 tab	8	10000
5 jaya	28	5 tab	8	10000

22 rows selected.

**SQL> SELECT CUSTOMERS.ID, CUSTOMERS.NAME, CUSTOMERS.AGE,
ORDERS.OID, ORDERS.PNAME, ORDERS.ID,ORDERS.AMOUNT FROM
CUSTOMERS INNER JOIN ORDERS ON CUSTOMERS.ID > ORDERS.ID;**

ID	NAME	AGE	OID	PNAME	ID	AMOUNT
2	Hemima	25	1	Computer	1	5000
					0	
3	pavan	30	1	Computer	1	50000
4	Supriya	25	1	Computer	1	50000
5	jaya	28	1	Computer	1	50000
3	pavan	30	2	LapTop	2	65000
4	Supriya	25	2	LapTop	2	65000
5	jaya	28	2	LapTop	2	65000
4	Supriya	25	3	Mobile	3	15000
5	jaya	28	3	Mobile	3	15000

9 rows selected.

SQL> SELECT CUSTOMERS.ID, CUSTOMERS.NAME, CUSTOMERS.AGE, ORDERS.OID, ORDERS.PNAME, ORDERS.ID,ORDERS.AMOUNT FROM CUSTOMERS INNER JOIN ORDERS ON CUSTOMERS.ID < ORDERS.ID;

ID	NAME	AGE	OID	PNAME	ID	AMOUNT
1	krishna	30	2	LapTop	2	65000
1	krishna	30	3	Mobile	3	15000
2	Hemima	25	3	Mobile	3	1500
					0	
1	krishna	30	4	TV	9	25000
2	Hemima	25	4	TV	9	25000
3	pavan	30	4	TV	9	25000
4	Supriya	25	4	TV	9	25000
5	jaya	28	4	TV	9	25000
1	krishna	30	5	tab	8	10000
2	Hemima	25	5	tab	8	10000

3 pavan	30	5 tab	8	10000
4 Supriya	25	5 tab	8	10000
5 jaya	28	5 tab	8	10000

**13 rows
selected.**

2. LEFT JOIN:

The SQL LEFT JOIN returns all rows from the left table, even if there are no matches in the right table. This means that if the ON clause matches 0 (zero) records in right table, the join will still return a row in the result,

but with NULL in each column from right table.

This means that a left join returns all the values from the left table, plus matched values from the right table or NULL in case of no matching join predicate.

Syntax: The basic syntax of LEFT JOIN is as follows:

```
SELECT table1.column1, table2.column2... FROM table1 LEFT JOIN table2 ON  
table1.common_filed = table2.common_field;
```

Example1:

Now, let us join these two tables using LEFT JOIN as follows:

```
SQL> SELECT CUSTOMERS.ID, CUSTOMERS.NAME, CUSTOMERS.AGE,  
ORDERS.OID, ORDERS.PNAME, ORDERS.ID FROM CUSTOMERS LEFT JOIN  
ORDERS ON ORDERS.ID = CUSTOMERS.ID;
```

ID	NAME	AGE	OID	PNAME	ID
1	murali	30	1	PC	1
2	kiran	30	2	Mobil e	2
5	Sandeep	30	3	tab	5
4		25	3	kiran	30

Example2:

```
SQL> SELECT ORDERS.OID, ORDERS.PNAME, ORDERS.ID, CUSTOMERS.ID,  
CUSTOMERS.NAME, CUSTOMERS.AGE FROM  
ORDERS LEFT JOIN CUSTOMERS  
ON CUSTOMERS.ID = ORDERS.ID;
```

OID	PNAME	ID	ID	NAME	AGE

3	tab	5	5	Sandeep	30
1	PC	1	1	murali	30
2	Mobile	2	2	kiran	30
4	TV				

3. RIGHT JOIN:

The SQL RIGHT JOIN returns all rows from the right table, even if there are no matches in the left table. This means that if the ON clause matches 0 (zero) records in left table, the join will still return a row in the result, but with NULL in each column from left table.

This means that a right join returns all the values from the right table, plus matched values from the left table or NULL in case of no matching join predicate.

Syntax: The basic syntax of RIGHT JOIN is as follows:

```
SELECT table1.column1, table2.column2... FROM table1 RIGHT JOIN table2 ON  
table1.common_field = table2.common_field;
```

Example1:

Now, let us join these two tables using RIGHT JOIN as follows:

```
SQL> SELECT CUSTOMERS.ID, CUSTOMERS.NAME, CUSTOMERS.AGE,  
ORDERS.OID, ORDERS.PNAME, ORDERS.ID FROM CUSTOMERS RIGHT JOIN  
ORDERS ON CUSTOMERS.ID = ORDERS.ID;
```

ID	NAME	AGE	OID	PNAME	ID
5	Sandeep	30	3	tab	5
1	murali	30	1	PC	1
2	kiran	30	2	Mobile	2
4				TV	9

Example2:

```
SQL> SELECT ORDERS.OID, ORDERS.PNAME, ORDERS.ID, CUSTOMERS.ID,  
CUSTOMERS.NAME, CUSTOMERS.AGE FROM ORDERS RIGHT JOIN  
CUSTOMERS ON ORDERS.ID = CUSTOMERS.ID; OUTPUT:
```

OID	PNAME	ID	ID	NAME	AGE
1	PC	1	1	Murali	30
2	Mobile	2	2	Kiran	30
3	Tab	5	5	Sandeep	30
4		25	3	Kiran	30

4. FULL JOIN:

The SQL FULL JOIN combines the results of both left and right outer joins.

The joined table will contain all records from both tables, and fill in NULLs for missing matches on either side.

Syntax: The basic syntax of FULL JOIN is as follows:

```
SQL> SELECT table1.column1,table2.column2...FROM table1 FULL JOIN table2  
ON table1.common_filed = table2.common_field;
```

Example1: Now, let us join these two tables using FULL JOIN as follows:

```
SQL> SELECT ORDERS.OID, ORDERS.PNAME, ORDERS.ID, CUSTOMERS.ID,  
CUSTOMERS.NAME, CUSTOMERS.AGE FROM ORDERS FULL JOIN CUSTOMERS  
ON ORDERS.ID  
= CUSTOMERS.ID;
```

OID	PNAME	ID	ID	NAME	AGE
3	tab	5	5	Sandeep	30
1	PC	1	1	murali	30
2	Mobile	2	2	kiran	30
3	kiran	30			
4		25			
4	TV	9			

6 rows selected.

Example2:

```
SQL> SELECT CUSTOMERS.ID, CUSTOMERS.NAME, CUSTOMERS.AGE,  
ORDERS.OID, ORDERS.PNAME, ORDERS.ID FROM CUSTOMERS FULL JOIN  
ORDERS ON CUSTOMERS.ID = ORDERS.ID;
```

ID	NAME	AGE	OID	PNAME	ID
5	Sandeep	30	3	tab	5
1	murali	30	1	PC	1
2	kiran	30	2		
3	kiran	30			
4		25		Mobile	2
4	TV	9			

6 rows selected.

5. SELF JOIN:

Self-Join is a specific type of Join. A self-join is a join that joins a table with itself. A self-join simply specifies that each row of a table is combined with itself and every other row of the

table. A self-join is useful for comparing rows within a table or querying hierarchical data. A self-join uses other joins such as inner join and left join. In addition, it uses the table alias to assign the table different names in the same query. The SQL SELF JOIN is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement. To join a table itself means that each row of the table is combined with itself and with every other row of the table. The self-join can be viewed as a join of two copies of the same table. The

table is not actually copied, but SQL performs the command as though it were. To perform a self-join, Oracle Database combines and returns rows of the table that satisfy the join condition.

Syntax: The basic syntax of SELF JOIN is as follows:

```
SQL> SELECT a.column_name, b.column_name... FROM table1 a, table1 b WHERE a.common_field = b.common_field;
```

Here, WHERE clause could be any given expression based on your requirement.

Example: Now, let us join this table using SELF JOIN as follows:

```
SQL> SELECT C.ID, C.NAME, C.AGE FROM CUSTOMERS C JOIN CUSTOMERS M ON C.ID = M.ID;
```

OUTPUT:

ID	NAME	AGE
----	------	-----

5	Sandeep	30
1	murali	30
2	kiran	30
3	kiran	30
4		25

```
SQL> SELECT O.OID, O.PNAME, O.ID, C.ID, C.NAME, C.AGE FROM ORDERS O JOIN CUSTOMERS C ON O.ID = C.ID;
```

OID	PNAME	ID	ID	NAME	AGE
-----	-------	----	----	------	-----

1	LapTop	1	1	krishna	25
5	Pen	3	3	Prasanna	21
3	Mouse	5	5	Vinitha	22

```
SQL> SELECT O.OID, O.PNAME, O.ID, C.ID, C.NAME, C.AGE FROM ORDERS O JOIN CUSTOMERS C ON O.ID != C.ID;
```

OID	PNAME	ID	ID	NAME	AGE
-----	-------	----	----	------	-----

1 LapTop	1	2 Shirisha	35
1 LapTop	1	3 Prasanna	21
1 LapTop	1	4 Srinu	25
1 LapTop	1	5 Vinitha	22
2 Mobile	6	1 krishna	25
2 Mobile	6	2 Shirisha	35
2 Mobile	6	3 Prasanna	21
2 Mobile	6	4 Srinu	25
2 Mobile	6	5 Vinitha	22
3 Mouse	5	1 krishna	25
3 Mouse	5	2 Shirisha	35
3 Mouse	5	3 Prasanna	21
3 Mouse	5	4 Srinu	25
4 CPU	8	1 krishna	25
4 CPU	8	2 Shirisha	35
4 CPU	8	3 Prasanna	21
4 CPU	8	4 Srinu	25
4 CPU	8	5 Vinitha	22
5 Pen	3	1 krishna	25
5 Pen	3	2 Shirisha	35
5 Pen	3	4 Srinu	25
5 Pen	3	5 Vinitha	22

22 rows selected.

6. CROSS JOIN (OR) CARTESIAN JOIN:

A CROSS JOIN, also known as a Cartesian join, is a type of join in SQL that combines every row from one table with every row from another table. It produces a Cartesian product, which is the result of multiplying the number of rows in the first table by the number of rows in the second table. The result set of a CROSS JOIN contains all possible combinations of rows from both tables. CROSS JOIN can be useful in certain scenarios, such as when you need to generate all possible combinations or when you want to create a

temporary table for further analysis. However, it's important to exercise caution with CROSS JOIN, as it can lead to large result sets and potentially impact performance if used without proper consideration.

Syntax: The basic syntax of INNER JOIN is as follows:

SQL> SELECT table1.column1, table2.column2... FROM table1, table2 [, table3] Example:

Example: Now, let us join these two tables using INNER JOIN as follows:

SQL> SELECT CUSTOMERS.ID, CUSTOMERS.NAME, ORDERS.oid, ORDERS.pname FROM CUSTOMERS CROSS JOIN ORDERS;

In the above query we will select ID and NAME from Customers table and OID and PNMAE from Orders table. In the output you can see that each row of the table Student is joined with every row of the table Orders. The total rows in the result-set = $5 * 4 = 20$.

OUTPUT:

**SQL> SELECT CUSTOMERS.ID, CUSTOMERS.NAME, ORDERS.oid,
ORDERS.pname FROM CUSTOMERS CROSS JOIN ORDERS;**

ID	NAME	OID	PNAME
	krishna	1	Compute r
	krishna	2	LapTop
	krishna	3	Mobile
	krishna	4	TV
	krishna	5	tab
2	Hemima	1	Compute r
2	Hemima	2	LapTop
2	Hemima	3	Mobile
2	Hemima	4	TV
2	Hemima	5	tab
3	pavan	1	Compute r
3	pavan	2	LapTop
3	pavan	3	Mobile
3	pavan	4	TV
3	pavan	5	tab
4	Supriya	1	Compute r
4	Supriya	2	LapTop
4	Supriya	3	Mobile
4	Supriya	4	TV
4	Supriya	5	tab
5	jaya	1	Compute r
5	jaya	2	LapTop
5	jaya	3	Mobile
5	jaya	4	TV
5	jaya	5	tab

25 rows selected.

Module-9: SQL SEQUENCES and SQL SYNONYM SEQUENCES

In Oracle, a sequence is an object used to generate unique numeric values automatically. Sequences are often used to generate primary key values for tables or to create unique identifiers. Sequence is an object which is used to generate numbers. A sequence is a set of integers 1, 2, 3, ... that are generated in order on demand. Sequences are frequently used in databases because many applications require each row in a table to contain a unique value, and sequences provide an easy way to generate them. Sequences provide a reliable and efficient way to generate unique numeric values in Oracle. They are commonly used for generating primary keys, unique identifiers, and other scenarios where unique sequential numbers are required.

Different features of sequences:

1. Sequence is database object that generate produce integer values in sequential order.
2. It automatically generates primary key and unique key values.
3. It may be ascending or descending order.
4. It can be used for multiple tables.
5. Sequence numbers are stored and generated independently of tables.
6. It saves a time by reducing application code.
7. It is used to generate unique integers.
8. It is used to create an auto number field.
9. Useful when you need to create a unique number to act as a primary key.
10. Oracle provides an object called as a Sequence that can generate numeric values. The value generated can have maximum of 38 digits.

Syntax: Create sequence <SEQUENCE_NAME> start with <value> increment by <value>; Here's an example of creating and using a sequence in Oracle:

Creating a Sequence:

SQL> CREATE SEQUENCE stu_id START WITH 5 INCREMENT BY 1;

Sequence created.

In this example, we create a sequence called "stu_id" that starts with 5 and increments by 1 for each new value.

Using the Sequence [Insert Values]:

To use the sequence and generate unique values, you can reference it in your SQL statements. Here's an example of inserting data into an "Employees" table using the sequence:

SQL> INSERT INTO student (sid, sname,marks) VALUES (stu_id.NEXTVAL, 'John',80);

1 row created.

In this example, the stu_id.NEXTVAL function is used to retrieve the next value from the sequence and assign it to the "sid" column in the "Student" table.

SQL> select * from student;

SID	SNAME	TOTALMARKS	MARKS

2 krishna	200	50
3 krishna	200	50
5 John	100	80
1 Krishna	100	90

NEXTVAL:

Nextval is used to generate a number. You can use the NEXTVAL function of the sequence to generate unique values whenever needed in your SQL statements. Each time you call NEXTVAL, Oracle will return the next value from the sequence.

CURRVAL:

Currval is used to know the latest number generated. To view the current value of a sequence without incrementing it, you can use the CURRVAL function. Viewing the Current Value of a Sequence.

Example:

```
SQL> SELECT stu_id.CURRVAL FROM  
DUAL; CURRVAL
```

```
-----  
5
```

To know the latest value generated. The DUAL table is a system table in Oracle that can be used for single- row queries. Calling CURRVAL on the sequence retrieves its current value without advancing it.

□ Alter Sequence:

In Oracle, you can use the ALTER SEQUENCE statement to modify the attributes of a sequence. This allows you to adjust properties such as the sequence increment, minimum value, maximum value, and more. Now, let's say we want to modify this sequence to change the increment to 15 and the maximum value to 50. Here's how the ALTER SEQUENCE statement would look:

Syntax: ALTER SEQUENCE sequence_name [INCREMENT BY increment_value] [MAXVALUE max_value];

```
SQL> ALTER SEQUENCE stu_id INCREMENT BY 15 MAXVALUE 50;
```

This statement will modify the "stu_id" sequence to increment by 15 and set the maximum value to 50. Other attributes of the sequence will remain unchanged.

Query to see all sequences in oracle:

To see all sequences in an Oracle database, you can query the USER_SEQUENCES view, depending on your permissions and the scope of sequences you want to retrieve. Here's how you can query these views:

Only Sequence Name:

```
SQL> SELECT sequence_name FROM
user_sequences; SEQUENCE_NAME
-----
```

SSS_SI

D

STU_ID

STU_SI

D

Sequence Name with min_value, max_value, increment_by, last_number:

SQL> SELECT sequence_name, min_value, max_value, increment_by, last_number FROM user_sequences;

SEQUENCE_NAME	MIN_VALUE	MAX_VALUE	INCREMENT_BY	
				LAST_NUMBER

SSS_SID	1	50	10	53
STU_ID	1	1.0000E+28	1	27
STU_SID	1	1.0000E+28	1	1

Drop Sequence:

Once you have created your sequence in Oracle, you might find that you need to remove it from the database.

Syntax: `DROP SEQUENCE sequence_name;`

sequence_name: The name of the sequence that you wish to drop. **Example:**

SQL> DROP SEQUENCE stu_id;

Sequence dropped.

SYNONYM

In Oracle, a synonym is an alternative name given to an object, such as a table, view, sequence, procedure, or function. Synonyms can provide a convenient and concise way to reference database objects by using a different name. A synonym is an alias or friendly name for the database objects (such as tables, views, stored procedures, functions, and packages). It is an alternate name given to an object. A SYNONYM provides another name for database object, referred to as original object, that may exist on a local or another server. A synonym belongs to schema, name of synonym should be unique. Synonyms can be useful in scenarios where you have long or complex object names, or when you want to provide an alias or abstraction layer for the underlying objects. They can also be used to simplify the migration or maintenance of database objects by redirecting references to the synonyms instead of modifying the actual object names in dependent code.

Here's an example to illustrate the concept of synonyms in Oracle:

Let's say we have a table called "**Employees**" with the following columns:

EmployeeID	Name	Department
1	John	Sales
2	Jane	HR
3	Mary	Sales
4	Michael	Finance
5	Alice	HR

Create Synonym: Now, let's create a synonym called "Emp" for the "Employees" table:

SQL> CREATE SYNONYM Emp FOR

Employees; Synonym created.

With this synonym in place, you can refer to the "Employees" table as "Emp". For example, you can run queries like:

SQL> SELECT * FROM Emp;

EmployeeID	Name	Department
1	John	Sales

2	Jane	HR
3	Mary	Sales
4	Michael	Financ e
5	Alice	HR

This query will retrieve all the rows from the "Employees" table, but you are referencing it using the synonym "Emp".

Query to see all synonyms:

```
SQL> select synonym_name from
user_synonyms; SYNONYM_NAME
```

STU

□ Drop a
synonym: SQL>
drop synonym stu;
Synonym dropped.

LOCKS

As oracle is a multiuser environment there is always a chance that multiple users will perform DML operators on some table parallel in such case the data becomes inconsistent. To maintain the consistency of data base a user can lock the table. Select for update command is used for locking a table.

BEGIN

```
UPDATE employees  
SET first_name =  
'Jane'  
WHERE employee_id =  
1; COMMIT;  
END;
```

Ex: Ajit> select * from student for update; Table is locked by Ajit

ASHWIN> update Ajit.student set marks = 95 where sno = 101; Client will be in waiting state. Locks are released when a user executes commit command.

Levels of locks: There are three levels of locks.

1. **Row level**
2. **Page level**
3. **Table level**

***Row level:** when where clause in select for update command is evaluating to one row, row level lock is applied.

Ex: select * from student where sno =101 for update;

***Page level:** when where clause in select for update command is evaluating to multiple rows, page level lock is applied.

Ex: Select * from emp where deptno = 20 for update;

***Table level:** When where clause is not used in select for update, table level lock is applied. Ex: select * from emp for update;

Module-10: TABLE ALIAS and TRIGGERS and FLASHBACK and PURGE COMMANDS

TABLE ALIAS:

Table aliases in Oracle allow you to assign temporary names or shorthand references to tables in your SQL queries. Using table aliases can make your queries more concise and readable, especially when dealing with complex queries involving multiple tables. To assign a table alias, you simply provide the alias immediately after the table name in the FROM or JOIN clause. The alias can be any valid identifier and is followed by a space. Table alias is an alternate name given to a table. By using a table alias length of the table reduces and at the same time performance is maintains. Table alias is temporary once the query is executed the table alias are loosed.

In Oracle, aliasing can also be done in column name as well as in table name. Aliasing is done to give a temporary to a column or table.

Syntax for column:

Column_name AS alias_name

Syntax for table:

Table_name alias_name

Parameters:

column_name: original name of the

column table_name: original name of

the table alias_name: temporary name

SQL> SQL> SELECT C.ID AS CUSTID, C.NAME AS CUSTNAME, O.OID AS ORDERID FROM CUSTOMERS C JOIN ORDERS O ON C.ID = O.ID;

In the above query, the table aliases "c" and "o" are assigned to the "Customers" and "Orders" tables, respectively. These aliases are used throughout the query to refer to the tables instead of their full names. The result set will include columns from both tables, and the join condition is specified using the aliases.

OUTPUT:

CUSTID	CUSTNAME	ORDERID
--------	----------	---------

1	krishna	1
2	Hemima	2
3	pavan	3

Create a duplicate table in Oracle:

When working on oracle sometime for testing purposes we are required to create a copy of a table.

A duplicate database is useful for a variety of purposes:

- Test backup and recovery procedures

- Test an upgrade to a new release of Oracle Database
- Test the effect of applications on database performance

Copy a table with data within the same Oracle database:

- Oracle provides convenient syntax that helps implement the above-mentioned scenarios. By means of the “CREATE TABLE ... AS SELECT ... ” command, you can create a duplicate table within the same database schema. To create an exact copy of the table preserving the table structure and all the data, execute the query as follows:

Syntax:

```
CREATE TABLE new_table_name AS SELECT * FROM existing_table_name;
```

Example:

```
SQL> CREATE TABLE oo1 AS SELECT * FROM orders;
```

Table created.

```
SQL> select * from oo1;
```

OID	PNAME	AMOUNT	ID
1	LapTop	500	1
2	Mobile	250	6
3	Mouse	999	5
5	CPU	800	8
5	Pen	10	3

- If you want to limit your data copying to specific columns, indicate the column names after SELECT in the following way:

Syntax:

```
CREATE TABLE new_table_name AS SELECT column_name1,column_name2
                                FROM existing_table_name;
```

```
SQL> CREATE TABLE oo2 AS SELECT oid, pname FROM orders;
```

Table created.

```
SQL> select * from
      oo2;
```

OID PNAME

1 LapTop

2 Mobile

3 Mouse

4 CPU

5 Pen

SQL> CREATE TABLE oo3 AS SELECT oid,pname FROM orders where pname='Mobile';

Table created.

SQL> select * from oo3;

OID PNAME

2 Mobile

SQL> CREATE TABLE oo4 AS SELECT oid,pname FROM orders where pname='Mobile123';

Table created.

SQL> select * from oo4;

no rows selected

SQL> desc oo4;

Name	Null?	Type
------	-------	------

OID	NUMBER(5)
-----	-----------

PNAME	VARCHAR2(20)
-------	--------------

TRIGGERS

A Trigger is a PL/SQL block which is executed automatically basing on an event. Triggering events are inserted, update, delete. Triggers in PL/SQL. Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events –

- A database manipulation (DML) statement (DELETE, INSERT, or UPDATE) A database definition (DDL) statement (CREATE, ALTER, or DROP).
- A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN). Triggers can be defined on the table, view, schema, or database with which the event is associated. **Syntax:**

Create or replace trigger <TRIGGER_NAME><TIMMING><EVENT> ON
<OBJECT_NAME>

Begin

END;

/

Example:

```
create or replace trigger trg1 after insert on dept begin
DBMS_OUTPUT.PUT_LINE('hello');
END;
```

/

Trigger created

Triggers are divided into two types

1. **Statement level trigger**
 2. **Row level trigger**
- **Statement level trigger:**

These triggers are executed only once irrespective of number of rows affected by the event. By default, every trigger is a statement level.

- **Row level trigger:**

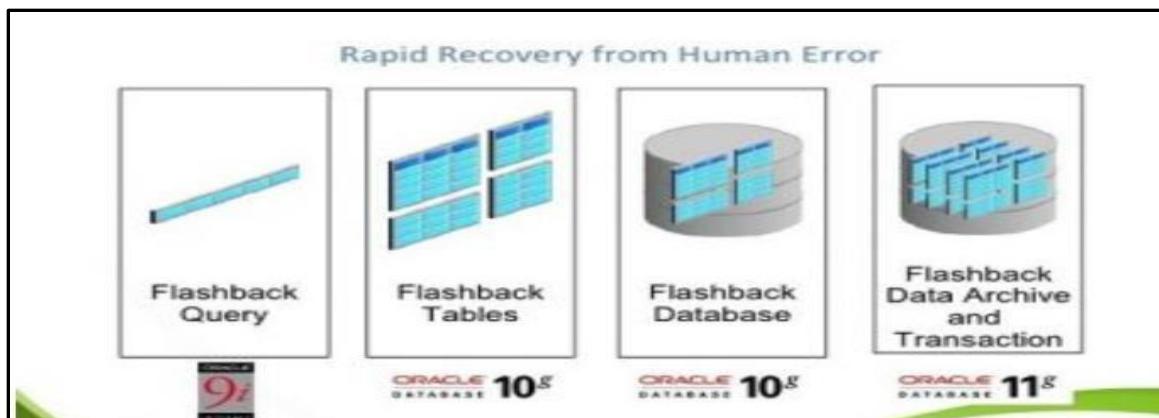
These triggers are executed for every row which is affected by the event (multiple numbers of times). We can create a row level trigger by using “FOR EACH ROW” clause.

FLASHBACK and PURGE COMMANDS

What is Recycle bin?

Oracle has introduced “Recycle bin” feature oracle log to store all the dropped objects. A user drops a very important table accidentally. Oracle log Recycle bin feature the user can easily restore the dropped object. Oracle database has the recycle bin which is a data dictionary table and contains information about dropped objects. Until you don’t use purge option, all dropped tables and its objects such as indexes, constraints and etc are not removed and still occupy space.

Actually, the recycle bin is any life buy for the database administrators, because sometime we can restore or flash back the dropped tables from recycle bin, if these objects are still in the recycle bin.



You can check if the recycle bin is on or off as

follows. SQL> show parameter recyclebin;

NAME	TYPE	VALUE
------	------	-------

recyclebin	string	on
------------	--------	----

To disable the recycle bin, you can disable the recycle bin both for session and system level as follows. SQL> ALTER SESSION SET recyclebin = OFF; (or)

SQL> ALTER SYSTEM SET recyclebin = OFF;

Session altered.

You can enable the recycle bin both for session and system level

as follows. SQL> Alter system set recycle bin = on;

or

SQL> Alter session set recycle bin = on;

Session altered.

Now, lets go to make an example and recover a dropped table from recycle bin as follows. SQL> show parameter recyclebin;

NAME TYPE VALUE

recyclebin string on

To view the recycle bin use the follows command:

SQL> show recycle bin;

Restore Dropped Tables from Recycle Bin:

SQL> select * from sss;

SID	SNAME	MARKS
1	John	80
2	John	80
3	John	80

SQL> select count(*) from sss; Following table has 3 rows.

COUNT(*)

3

Lets drop it.

SQL> drop table sss; Table dropped.

Check if it exists.

SQL> select * from sss; select * from sss
*

ERROR at line

1:

ORA-00942: table or view does not exist

Now lets go to flashback table from recycle bin as follows.

FLASHBACK:

- **FLASHBACK** is a command used in Oracle Database to recover a database or table to a previous state, such as a specific point in time or before a particular transaction.
- It allows you to view and potentially revert changes made to the database without the need for a complete database restore.
- The **FLASHBACK** command can be used to query historical data using features like Flashback Query and Flashback Version Query.
- It's a useful feature for data recovery and auditing purposes.

Restore the objects back to data base:

FLASHBACK table <<table_name>> to
before drop; SQL> flashback table sss to
before drop;
Flashback complete.

Check if table exists or not.

SQL> select * from sss;

SID	SNAME	MARKS
1	John	80
2	John	80
3	John	80

SQL> select count(*) from sss;

Following table has 3 rows.

COUNT(*)

3

You can list the objects and droptime from recycle bin as follows.

SQL> select object_name, droptime from dba_recyclebin;

OBJECT_NAME	DROPTIME
BIN\$4YkbXtBoAdngQwEAAH8n+g==	0 2020-03-30:10:49:57
BIN\$4YkbXtBpAdngQwEAAH8n+g==	0 2020-03-30:10:49:57
BIN\$4YkbXtBqAdngQwEAAH8n+g==	0 2020-03-30:10:49:57

BIN\$4YkbXtBoAdngQwEAAH8n+g== 0 2020-03-30:10:49:57

BIN\$4YkbXtBpAdngQwEAAH8n+g== 0 2020-03-30:10:49:57

BIN\$4YkbXtBqAdngQwEAAH8n+g== 0 2020-03-30:10:49:57

BIN\$4YkbXtBrAdngQwEAAH8n+g== 0 2020-03-30:10:49:57

BIN\$4YkbXtBsAdngQwEAAH8n+g==\$0 2020-03-30:10:49:57

BIN\$4YkcLLzGAd3gQwEAAH/TdQ==\$0 2020-03-30:10:50:11

PURGE command:

Clearing the recycle bin (RB):

You can purge recycle bin as follows. `purge recyclebin` will remove all objects from the user's recycle bin and release all space associated with objects in the recycle bin

To clear the recycle bin the following statement can be used.

SQL> `purge recyclebin;`

Recyclebin purged.

Purge DBA_RECYCLEBIN:

Purge dba_recyclebin needs sysdba privilege and it removes all objects from the system-wide recycle bin, and is equivalent to purging the recycle bin of every user

SQL> purge dba_recyclebin;

DBA Recyclebin purged.

I would like to thank you for taking the time to read SQL TUTORIAL. I hope that you enjoyed reading this book as much as I enjoyed it while I was writing it. It is my biggest pleasure if we by any means manage to help you build a Strong SQL Queries for yourself.

Thank you.

N. Krishna

[Software Professional Trainer].

For Queries: 9493217167.

SQL Interview Questions

[Top 200 SQL Interview Questions with Answers]

Here is a list of possible interview questions of **SQL**, along with answers to help you prepare for your interview:

1. What is SQL?

- SQL – Structured Query Language, an ANSI (American National Standards Institute) Standard language for accessing databases and It was initially developed at IBM in the 1970s.
- SQL is the standard language to communicate with relational database management systems like Oracle, MS Access, MS SQL Server, MySQL, DB2, Sybase Etc...
- SQL is used to Create New Databases, New Tables in a Database, Insert records in a Database, Update records in a Database, and Delete records in a Database.
- SQL is used to retrieve data from a Database, Execute queries against a Database, Create stored procedures in a Database, Create views in a Database, Set permissions on tables, procedures, and views.

2. When did SQL launch?

Structured Query Language was first launched by IBM in 1974 and it is Free Software (anybody can use with free of cost).

3. Who should learn SQL?

- Database Developers
- Database Testers
- Database Administrators

4. What are the Uses of SQL?

- Creating new databases
- Creating new tables in a database
- Inserting records in a database
- Updating records in a database
- Deleting records from a database
- Retrieving data from a database

- Executing queries against a database
- Creating stored procedures in a database
- Creating views in a database
- Setting permissions on tables, procedures, and views etc...

5. What is SQL Process?

When we are executing an SQL command for any RDBMS, the system determines the best way to carry out our request, and the SQL engine figures out how to interpret the task.

There are various components included in the process. These components are Query Dispatcher, Optimization engines, Classic Query Engine and SQL query engine, etc. Classic query engine handles all non-SQL queries but SQL query engine won't handle logical files.

6. Is SQL support programming?

Basically, SQL is a programming language (Structured Query Language) but no control flow statements in SQL, it is a command-based programming language.

7. What is the difference between SQL, SQL Server, and MySQL?

- SQL – Structured Query Language used to manage the relational databases like Oracle, MS SQL Server, MySQL, Sybase, etc,
- SQL Server and MySQL are relational database management systems used to store, retrieve, modify and administer a database using SQL.
- MS SQL Server is a commercial Database Management System whereas MySQL is an Open-Source software.

8. What are the different subsets of SQL?

The important subsets of SQL are:

- DDL – Data Definition Language – It allows you to perform various operations on the database such as CREATE, ALTER, and DELETE objects.
- DML – Data Manipulation Language – It allows you to access and manipulate data. It helps you to insert, update, delete and retrieve data from the database.
- DCL – Data Control Language – It allows you to control access to the database. Example – Grant, Revoke access permissions.

9. What do you mean by DBMS? What are its different types?

A Database Management System (DBMS) is a software application that interacts with the user, applications, and the database itself to capture and analyze data.

A DBMS allows a user to interact with the database. The data stored in the database can be modified, retrieved and deleted and can be of any type like strings, numbers, images, etc.

There are two types of DBMS:

- Relational Database Management System/SQL Database: The data is stored in relations (tables). Example – Oracle.
- Non-Relational Database Management System/NoSQL Database: There is no concept of relations. Example – MongoDB.

10. What is Database Engine?

Software that stores and retrieves data in a database. It may be a self-contained program or the part of DBMS that performs the storage and retrieval operations.

11. What is the difference between DELETE and TRUNCATE statements?

Delete – Delete is a DML (Data Manipulation Language) command, it is used to delete a row in a table, we can roll back data after using delete statement, and It is slower than truncate statement.

Truncate – Truncate is a DDL (Data Definition Language) command, it is used to delete all the rows from a table, we cannot roll back data, and It is faster.

12. What do you mean by the table, field, and record in SQL?

A table refers to a collection of data in an organized manner in form of rows and columns where the rows are known as records and the columns are known as fields.

13. What are the important SQL language elements?

The important SQL Language elements are:

- Identifiers:

Names of Database Objects such as Tables, Views, Columns, etc,

- Data Types:

Define the type of data that is contained by a Column

- Constants:

Symbols that represent specific data types

- Operators:

Perform Arithmetic, Comparison, and Logical Operations

- Functions:
Built-in Functions to perform specific operations
- Clauses:
Constituent components of statements and queries.
- Expressions:
Produce scalar values, or tables containing columns and rows of data.
- Queries:
Retrieve the data based on specific criteria, this an important element of SQL.
- Statements Etc...

14. What is Database Server?

Database Management Systems provide Database server functionality, database server provides database services to other computer programs or computers.

15. What is the difference between CHAR and VARCHAR data types in SQL?

Both Char and Varchar are used for characters datatype but varchar is used for character strings of variable length whereas Char is used for strings of fixed length.

CHAR Datatype:

It is a datatype in SQL which is used to store character strings of the fixed length specified. If the length of the string is less than set or fixed-length then it is padded with extra blank spaces so that its length became equal to the set length.

VARCHAR Datatype:

It is a datatype in SQL which is used to store character strings of variable length but a maximum of the set length specified. If the length of the string is less than set or fixed-length then it will store as it is without padded with extra blank spaces.

16. What is an Index in SQL? What are the different types of indexes in SQL?

In most SQL projects the data is in millions and billions. Because of the huge volume of the data performance of SQL statements degrades. So there are a lot of different ways to improve the performance of the application, the performance of reports, or the performance of SQL queries.

17. Definition Of Index:

An index refers to a performance tuning method of allowing faster retrieval of records from the table. An index creates an entry for each value and hence it will be faster to retrieve data.

There are different types of Indexes in SQL:

- Clustered Index
- Non-Clustered Index
- Unique Index
- Filtered Index
- Columnstore Index
- Hash Index etc,

18.What do you mean by data integrity?

Data Integrity defines the accuracy as well as the consistency of the data stored in a database. It also defines integrity constraints to enforce business rules on the data when it is entered into an application or a database.

19.What is a Query in SQL? What are the parameters of SQL Queries?

A query is a question or inquiry about a set of data. We use Structured Query Language (SQL) to retrieve meaningful and relevant information from databases. When building a structure, we pull data from tables and fields.

An SQL query typically includes Query parameter, Column list, Sort list, Options, and Format.

Example: SQL > Select * from employee where department_id=10 and salary>3500

20.What is key in SQL? What are the different types of Keys in SQL?

A Key in DBMS is an attribute or set of attributes that helps you to identify a row(tuple) in a relation(table). They allow you to find the relation between two tables.

21.Different types of Keys in SQL:

- **Super Key** – A super key is a group of single or multiple keys which identifies rows in a table.
- **Primary Key** – is a column or group of columns in a table that uniquely identifies every row in that table.
- **Candidate Key** – is a set of attributes that uniquely identify tuples in a table. Candidate Key is a super key with no repeated attributes.

- **Alternate Key** – is a column or group of columns in a table that uniquely identifies every row in that table.
- **Foreign Key** – is a column that creates a relationship between two tables. The purpose of foreign key is to maintain data integrity and allow navigation between two different instances of an entity.
- **Compound Key** – has two or more attributes that allow you to uniquely recognize a specific record. It is possible that each column may not be unique by itself within the database.
- **Composite Key** – is a combination of two or more columns that uniquely identify rows in a table. The combination of columns guarantees uniqueness, though individual uniqueness is not guaranteed.
- **Surrogate Key** – An artificial key that aims to uniquely identify each record is called a surrogate key. These kinds of keys are unique because they are created when you don't have any natural primary keys.

22. What is the difference between DROP and TRUNCATE commands?

‘DROP’ command removes a table and it cannot be rolled back from the database whereas the ‘TRUNCATE’ command removes all the rows from the table.

23. What is Normalization and what are the advantages of it?

Normalization in SQL is the process of organizing data to avoid duplication and redundancy. Some of the advantages of Normalization are:

- Better Database organization
- More Tables with smaller rows
- Efficient data access
- Greater Flexibility for Queries
- Quickly find the information
- Easier to implement Security
- Allows easy modification
- Reduction of redundant and duplicate data
- More Compact Database
- Ensure Consistent data after modification

24.What is the ACID property in a database?

ACID stands for Atomicity, Consistency, Isolation, Durability. It is used to ensure that the data transactions are processed reliably in a database system.

Atomicity: Atomicity refers to the transactions that are completely done or failed where transaction refers to a single logical operation of data. It means if one part of any transaction fails, the entire transaction fails and the database state is left unchanged.

Consistency: Consistency ensures that the data must meet all the validation rules. In simple words, you can say that your transaction never leaves the database without completing its state.

Isolation: The main goal of isolation is concurrency control.

Durability: Durability means that if a transaction has been committed, it will occur whatever may come in between such as power loss, crash, or any sort of error.

25.What are Entities and Relationships?

Entities: A person, place, or thing in the real world about which data can be stored in a database. Tables store data that represents one type of entity.

For example – A bank database has a customer table to store customer information. The customer table stores this information as a set of attributes (columns within the table) for each customer.

Relationships: Relation or links between entities that have something to do with each other.

For example – The customer's name is related to the customer account number and contact information, which might be in the same table. There can also be relationships between separate tables (for example, customer to accounts).

26.What are joins in SQL?

A JOIN clause is used to combine rows from two or more tables, based on a related column between them. It is used to merge two tables or retrieve data from there.

There are 4 types of joins,

Inner join: Inner Join in SQL is the most common type of join. It is used to return all the rows from multiple tables where the join condition is satisfied.

Left Join: Left Join in SQL is used to return all the rows from the left table but only the matching rows from the right table where the join condition is fulfilled.

Right Join: Right Join in SQL is used to return all the rows from the right table but only the matching rows from the left table where the join condition is fulfilled.

Full Join: Full join returns all the records when there is a match in any of the tables. Therefore, it returns all the rows from the left-hand side table and all the rows from the right-hand side table.

27. What do you mean by “Trigger” in SQL?

Trigger in SQL is a special type of stored procedure that is defined to execute automatically in place or after data modifications. It allows you to execute a batch of code when an insert, update, or any other query is executed against a specific table.

28. Are NULL values the same as that of zero or a blank space?

A NULL value is not at all same as that of zero or a blank space. The NULL value represents a value that is unavailable, unknown, assigned, or not applicable whereas zero is a number and blank space is a character.

29. What is a subquery in SQL?

A subquery is a query inside another query where a query is defined to retrieve data or information back from the database. In a subquery, the outer query is called as the main query whereas the inner query is called subquery. Subqueries are always executed first and the result of the subquery is passed on to the main query. It can be nested inside a SELECT, UPDATE or any other query. A subquery can also use any comparison operators such as $>$, $<$ or $=$.

There are two types of subqueries namely, Correlated and Non-Correlated.

Correlated subquery: These are queries that select the data from a table referenced in the outer query. It is not considered as an independent query as it refers to another table and refers to the column in a table.

Non-Correlated subquery: This query is an independent query where the output of the subquery is substituted in the main query.

30. What is the SELECT statement?

SELECT is a SQL command that is used for selecting data from a database. The data which is returned is saved in a result table, called the result-set.

Example: `SELECT EMP_ID FROM EMPLOYEE;`

Note: In the above SQL statement, ‘EMP_ID’ is a column and ‘EMPLOYEE’ is a Table.

31.What is a stored procedure?

A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again. Stored Procedure is a function that consists of many SQL statements to access the database system. Stored procedure supports faster execution instead of executing multiple queries. This reduces network traffic and provides better security to the data.

The disadvantage is that it can be executed only in the Database and uses more memory for storage.

32.What is CLAUSE in SQL?

SQL clause helps to limit the result set by providing a condition to the query. A clause helps to filter the rows from the entire set of records.

For example – WHERE, HAVING clause.

33.What is a View? What are Views used for?

A view is a virtual table that consists of a subset of data contained in a table. Since views are not present, it takes less space to store. The view can have data of one or more tables combined and it depends on the relationship. A view refers to a logical snapshot based on a table or another view.

It is used for the following reasons:

- Restricting access to data.
- Making complex queries simple.
- Ensuring data independence.
- Providing different views of the same data.

34.What is MS Access?

- MS Access was launched in 1992 by Microsoft Corporation as part of MS Office.
- Microsoft Access is an entry-level database management software. It is not only inexpensive but also a powerful database for small-scale projects.
- MS Access uses the Jet database engine which utilizes a specific SQL language dialect (sometimes referred to as Jet SQL).

- MS Access comes with the professional edition of MS Office package. MS Access is user-friendly database management system.

35.What is Oracle?

Oracle is a relational database management system developed by ‘Oracle Corporation and launched in 1977. Oracle supports all major Operating systems including MS Windows, NetWare, UnixWare, OS/2, and most UNIX flavours.

36.What is MS SQL Server?

MS SQL Server is a Relational Database Management System developed by Microsoft Inc. Its primary query languages are T-SQL and ANSI SQL.

37.What is Sybase?

Sybase is a computer software company, their primary product is Sybase DBMS, which is a relational database management system based upon structured query language.

38.What is MySQL?

- MySQL is an open-source Database Management System, developed by Swedish company MySQL AB.
- MySQL Supports many different platforms including Microsoft Windows, the major Linux distributions, UNIX, and Mac OS X.
- MySQL has free and paid versions, depending on its usage (non-commercial/commercial) and features. MySQL comes with a very fast, multi-threaded, multi-user, and robust SQL database server.

39.What is DB2?

DB2 is the short name used for DATABASE 2. It is a relational database product developed by IBM. in 1983

40.What is DB/400?

It is one of the flavors of IBM DB2

41.What are the categories of operators available in SQL?

- Arithmetic operators
- Comparison operators
- Logical operators

42.What are Arithmetic operators in SQL?

Operator	Description
+(Addition)	Adds values
-(Subtraction)	Subtracts Right side value from Left side value
*(Multiplication)	Multiplies values on either side of the operator
/(Division)	Divides left hand operand by right hand operand
% (Modulus)	Divides left hand operand by right hand operand and returns remainder

43.What are Comparison operators in SQL?

For example $x = 1$, $y = 2$

Operator Example

=	$(x = y)$ is False
!=	$(x \neq y)$ is True or $(x \neq y)$ is true.
>	$(x > y)$ is False
<	$(x < y)$ is True
>=	$(x \geq y)$ is False
<=	$(x \leq y)$ is True
!<	$(x \neq y)$ is False
!>	$(x \neq y)$ is True.

Note: Comparison Operators return Logical Results

44.What are Logical operators in SQL?

Operator Description

NOT	Returns TRUE if the following condition is FALSE. Returns FALSE if it is TRUE.
AND	Returns TRUE if both component conditions are TRUE. Returns FALSE if either is FALSE.
OR	Returns TRUE if either component condition is TRUE. Returns FALSE if both are FALSE.

45.What is a Data Relationship and What are they?

Database Relationship is the connection between the tables in a database. There are 4 types of relationships, and they are:

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

46.What are Important Data Types in SQL?

Data Type	Syntax
character	char(x)
integer	integer
numeric	numeric(p,s)
decimal	decimal(p,s)
float	float(p)
date	date
time	time
character varying	varchar2(x)
bit	bit(x)
real	real
smallint	smallint

47.What are the Data Definition Language Commands and Operations?

Important Data Definition Language Commands

- Create
- Alter
- Drop
- Truncate
- Rename

48.Important Data Definition Language Operations

- Create a Database
- Use Database
- Rename a Database

d. Drop Database

e. Create a Table

- f. Rename Table
- g. Add a Column to existing Table
- h. Add multiple columns to existing Table
- i. Modify an existing column
- j. Rename a Column
- k. Drop a Column
- l. Truncate a Table
- m. Drop a Table

Note: Download and Install Oracle 11g version (It is Free Edition) and practice SQL Commands and Operations.

49. How to Create a Database?

Syntax: Create Database databaseName;

Example: Create Database studentDB;

50. How to Select a Database?

Syntax: Use databaseName;

Example: Use studentDB;

51. How to Rename a Database?

Syntax: Alter Database databaseName Modify Name = newdatabaseName;

Example: Alter Database studentDB Modify Name = Hyderabad Or Alter Database studentDB Modify Name = Hyderabad;

52. How to Drop a Database?

Syntax: Drop Database databaseName;

Example: Drop Database studentDB;

53. How to Create a Table?

Syntax: Create Table tableName(column1_name dataType(size),column2_name dataType(size),...);

Example: Create Table Students(STID int(10),STName char(50));

54. How to View Table info

Syntax: Select * from tablename;

Example: Select * from Students;

55. View Table Schema

Select * from INFORMATION_SCHEMA.COLUMNS where TABLE_NAME = 'Students';

56. How to Rename a Table?

Syntax: EXEC sp_rename 'old_tableName', 'new_tableName';

Example: EXEC sp_rename 'Students', 'newStudents';

57. How to Add a Column to an Existing Table?

Syntax: Alter Table table_name add column_name dataType(size);

Example: Alter Table newStudents add City char(50);

58. How to Add multiple columns to an Existing Table?

Syntax: Alter Table table_name add column1_name dataType(size), column2_name dataType(size); Or Alter Table table_name add column1_name dataType(size), column2_name dataType(size), ..;

Example: Alter Table newStudents add add1 char(100), add2 char(70); Or Alter Table newStudents add add3 char(100), add4 char(70), add5 char(100), phone int;

59. How to Modify an existing column?

Syntax: Alter Table table_name Alter Column column_name dataType(size);

Example: Alter Table newStudents Alter Column add1 varchar(150);

60. How to Rename a Column?

Syntax: EXEC sp_rename 'table_name.old_column_name', 'new_column_name';

Example: EXEC sp_rename 'newStudents.phone', 'mobile'

61. How to Drop a Column?

Syntax: Alter Table table_name Drop Column column_name;

Example: Alter Table newStudents Drop Column City;

62. How to Truncate a Table?

Truncate Table command is used to delete complete data from an existing table

Syntax: Truncate Table table_name;

Example: Truncate Table newStudents;

63. How to Drop a Table?

Drop Table command is used to delete complete Table (Data and Table Structure) from the Database.

Syntax: Drop Table table_name;

Example: Drop Table newStudents;

64. How to add a new record into a Table?

Using INSERT INTO statement, we can insert new rows

Syntax: INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)
VALUES (value1, value2, value3,...valueN);

65. How to fetch data from a Database Table?

Using SELECT Statement, we can fetch data from a Database Table

Syntax: SELECT column1, column2, columnN FROM table_name; Or SELECT * FROM table_name;

66. Explain about IN Operator?

The IN operator implements comparison to a list of values, that is, it tests whether a value matches any value in a list of values. IN comparisons have the following general format:
value-1 [NOT] IN (value-2 [, value-3] ...)

This comparison tests if value-1 matches value-2 or matches value-3, and so on. It is equivalent to the following logical predicate:

value-1 = value-2 [OR value-1 = value-3] ...

67. Explain about FROM Clause in SQL?

The FROM clause always follows the SELECT clause. It lists the tables accessed by the query.

For example, SELECT * FROM S;

When the From List contains multiple tables, commas separate the table names. For example,
SELECT sp.*, city FROM sp, s WHERE sp.sno=s.sno;

When the From List has multiple tables, they must be joined together.

68. What is the parameter substitution symbol used with the INSERT INTO command?

The parameter substitution symbol used with the INSERT INTO command is &.

69.What are the various uses of database triggers?

Database triggers can be used to enforce business rules, maintain derived values, and perform value-based auditing.

70.What is an event handler in SQL?

An event handler is a routine that is written to respond to a particular event.

71.What are the two methods of retrieving SQL?

The two methods of retrieving SQL are

1-Select

2-using Cursor.

72.What is a synonym? How is it used?

A synonym is used to reference a table or view by another name. The other name can then be written in the application code pointing to test tables in the development stage and to production entities when the code is migrated. The synonym is linked to the AUTHID that created it.

73.What is referential integrity?

Referential integrity refers to the consistency that must be maintained between primary and foreign keys, i.e. every foreign key value must have a corresponding primary key value.

74.Explain the EXPLAIN statement?

The ‘explain’ statement provides information about the optimizer’s choice of access path of the SQL.

75.How is the SUBSTR keyword used in SQL?

SUBSTR is used for string manipulation with column name, first position and string length used as arguments. E.g. SUBSTR (NAME, 1 3) refers to the first three characters in the column NAME.

76.What is the difference between group by and order by?

Group by controls the presentation of the rows, order by controls the presentation of the columns for the results of the SELECT statement.

77.What is a subselect? Is it different from a nested select?

A subselect is a select that works in conjunction with another select. A nested select is a kind of subselect where the inner select passes to the where criteria for the outer select.

78.What is the use of CASCADE CONSTRAINTS?

When this clause is used with the DROP command, a parent table can be dropped even when a child table exists.

79.How do you prevent output from coming to the screen?

The SET option TERMOUT controls output to the screen. Setting TERMOUT OFF turns off screen output. This option can be shortened to TERM.

80.Can the Primary key be a Foreign Key on the same table?

Yes, the Primary key is a Foreign Key on the same table.

81.How do you execute a host operating system command from within SQL?

By use of the exclamation point “!” (in UNIX and some other OS) or the HOST (HO) command.

82.What is a Cartesian product?

A Cartesian product is the result of an unrestricted join of two or more tables. The result set of a three-table Cartesian product will have $x * y * z$ number of rows where x, y, z correspond to the number of rows in each table involved in the join.

83.How can variables be passed to a SQL routine?

By use of the & symbol. For passing in variables the numbers 1-8 can be used (&1, &2,...,&8) to pass the values after the command into the SQL PLUS session. To be prompted for a specific variable, place the ampersand variable in the code itself: “select * from dba_tables where owner=&owner_name;” . Use of double ampersands tells SQLPLUS to resubstitute the value for each subsequent use of the variable, a single ampersand will cause a repromoted for the value unless an ACCEPT statement is used to get the value from the user.

84.What command is used to get back the privileges offered by the GRANT command?

Revoke command is used to get back the privileges offered by the GRANT command.

85.What are the advantages of procedures?

- Loaded once and used many times.
- Performance better coz all SQL statements are sent in one go from the application to the database.
- Security (no object privileges are given directly).

- Invoker's rights are possible.
- Data integrity, productivity.

86.What is Parsing?

Parsing checks syntax checks privileges and allocating Private SQL Area.

87.What is Cursor?

Name or handle to a private SQL area where Oracle parses and fetches query results.

88.What is Datawarehouse?

Datawarehouse is a central repository of data from multiple sources of information. Those data are consolidated, transformed, and made available for mining and online processing. Warehouse data have a subset of data called Data Marts.

89.In what sequence SQL statements are processed?

The clauses of the select are processed in the following sequence

- FROM clause
- WHERE clause
- GROUP BY clause
- HAVING clause
- SELECT clause
- ORDER BY clause
- TOP clause

90.Write down the general syntax for a SELECT statement covering all the options.

Here's the basic syntax: (Also checkout SELECT in books online for advanced syntax)

- SELECT select_list
- [INTO new_table_]
- FROM table_source
- [WHERE search_condition]
- [GROUP BY group_by expression]
- [HAVING search_condition]
- [ORDER BY order expression [ASC | DESC]]

91.What is a correlated subquery?

When a subquery is tied to the outer query. Mostly used in self joins.

92. How to fetch common records from two tables?

Common records result set can be achieved by -.

Select studentID from student INTERSECT Select StudentID from Exam

93. How to select unique records from a table?

Select unique records from a table by using DISTINCT keyword.

Example: Select DISTINCT StudentID, StudentName from Student.

94. Which operator is used in query for pattern matching?

LIKE operator is used for pattern matching, and it can be used as -.

- % – Matches zero or more characters.
- _(Underscore) – Matching exactly one character.
Example:
- Select * from Student where studentname like 'a%';
- Select * from Student where studentname like 'ami_';

95. What are the types of subquery?

There are two types of subquery– Correlated and Non-Correlated.

A correlated subquery cannot be considered as an independent query, but it can refer to the column in a table listed in the FROM the list of the main query.

96. What is SQL?

- SQL stands for Structured Query Language
- SQL was initially developed at IBM in the 1970s
- SQL is the standard language to communicate with relational database management systems, like Oracle, MS Access, MS SQL Server, MySQL, DB2, Sybase Etc...

Purpose of SQL

- SQL is used to Create New Databases
- SQL is used to Create New Tables in a Database
- SQL is used to Insert records in a Database
- SQL is used to Update records in a Database
- SQL is used to Delete records in a Database

- SQL is used to Retrieve data from a Database

- SQL is used to execute queries against a Database
- SQL can set permissions on tables, procedures and views
- SQL is used to Create stored procedures in a Database
- SQL is used to Create views in a Database

97. Who should learn SQL?

- Database Developers

Design and deploy Database table structures, forms, reports and queries etc...

- Database Administrators (DBA)

Keeping databases up to date and managing database access

Writing Reports, documentation, and operating manuals

- Database Testers

Verify Data Integrity

Verify Data Manipulations (Add, Update, and Delete)

Verify Data comparisons

98. What are the subsets of SQL?

SQL Commands can be classified into groups based on their nature, they are,

- Data Definition Language
- Data Manipulation Language
- Data Control Language

99. Important Commands and Operations in Data Definition Language [DDL].

Data Definition Language Commands are used to Create, Modify, and Drop the Structure of Database Objects like **Table, View, and Index**, etc...

- **Create:** To create databases and database objects

Syntax: Create table < Table name> (col name 1 datatype (width), colname 2 datatype (width)...);

Ex: SQL > create table student (Sno number (10), Sname varchar2 (20),.....);

- **Alter:** To modify existing database objects.

Alter table to add column:

Syntax: SQL> alter table < table name> add(colname datatype (width));

Ex: SQL> alter table student add (group varchar (6));

□ **To modify size (OR) column:**

Syntax: SQL> alter table <table name> modify (column datatype (width));

Ex: SQL> alter table student modify (sgroup varchar(10));

□ **To Rename column:**

Syntax: alter table tablename rename existing columnname to new columnname;

Ex: alter table student rename column sid to StudentId;

• **Drop:** To drop databases and databases objects

Syntax: ALTER TABLE tablename DROP column columnname;

Ex: SQL> ALTER TABLE student DROP column studentid;

• **Truncate:** To remove all records from a table

Syntax: TRUNCATE TABLE table_name;

Example: TRUNCATE TABLE EMPLOYEE

• **Rename:** To rename database objects(Table).

Syntax: rename Old tablename to new tablename

Example: SQL > rename student tostudent1;

100. Important DDL Operations.

- Create a Database
- Use Database
- Rename a Database
- Drop Database
- Create a Table
- Rename Table
- Add a Column to exiting Table
- Add multiple columns to existing Table
- Modify an existing column
- Rename a Column
- Drop a Column
- Truncate a Table
- Drop a Table

101. Important Commands and Operations in Data Manipulation Language [DML].

- **Select:** To select specific data from a database. To fetch records from the tables and views stored in the database, the Oracle SELECT statement is used.

- To select all fields from a table:**

Syntax: SELECT * FROM table_name;

Example: SELECT * from students;

- To select specific fields from a table:**

Syntax: SELECT field1,field2 FROM table_name;

Example: SELECT name, age FROM students

- To select specific fields from a table using conditions:**

Syntax: SELECT field1,field2 FROM table_name WHERE conditions;

Example: SELECT name, age FROM students WHERE age>10

ORDER BY name;

- To select specific fields from multiple tables:**

Syntax: SELECT expressions FROM table1 INNER JOIN table2 ON join_predicate;

Example: SELECT students.name, teachers.subject FROM students INNER JOIN teachers ON students.student_id = student_id ;

ORDER BY name;

- **Insert:** To insert new records in a table

- Inserting values for all columns:**

Syntax: insert into tablename values(Column1value,Column2value,Co3umn1value,.....);

Ex: insert into employee values(1001,'krishna',50000,'05-may-2020');

- Inserting values to specific columns:**

Syntax: insert into tablename(columnname,columnname) values(columndata, cloumndata);

Ex: SQL> insert into employee (eid,ename) values(1003,'Supriya');

- Inserting values for all columns at a time:**

Syntax:insert into tablename values(&column1name,&column2name. ...);

Ex: SQL> insert into employee values(&eid,'&ename','&salery','&doj');

- **Update:** To update existing records.

Syntax: SQL>update < table name> set colname -values..... where < condition>;

Ex: SQL> update emp set da = sal * 10/100;

SQL> update emp set da = sal * 20/100 where ename = 'smith';

- Delete: To delete existing records from a table.

- Delete the all rows from a table:**

Syntax: SQL> delete from <table name> where <condition>;

Ex: SQL> delete from student;

- Delete particular row from a table:**

Ex: SQL> delete from student where Sno=101;

102. Important Commands and Operations in Data Control Language [DCL].

- Grant: To provide access to the Database objects to the users

Syntax: SQL> grant privileges on <table name> to user name;

Ex: SQL> grant select on emp to krishdb;

- Revoke: to remove user access rights to the database objects

Syntax: SQL> revoke privileges on <table name> from user name;

Ex: SQL> revoke select on emp from krishdb;

103. Important Commands and Operations in Data Control Language [TCL].

This language is used to transaction process, it consists of commit & Roll back commands.

- **Commit command:** It is used to make database changes permanently.

Syntax: SQL> commit;

- **Roll back command:** It is used to cancel the committed transactions.

Syntax: SQL> roll back;

104. Popular Database Management Systems.

- Oracle
- Microsoft SQL Server
- MySQL
- PostgreSQL
- MS Access

104. SQL Syntax.

- A database contains one or more tables. Each table is identified by a name, Tables contain records (rows) with data.

- Most of the actions we need to perform on a database are done with SQL statements.
- SQL keywords are NOT case sensitive: select is the same as SELECT
- All the SQL statements start with any of the keywords like SELECT, INSERT, UPDATE, DELETE etc...and all the statements end with a semicolon (;), the semicolon is the standard way to separate SQL Statements.

105. SQL Data Types.

A data type defines what kind of value a column can contain, we have to use data types while creating database tables, choose a particular data type for a table column based on our requirement.

Example:

- Character Data Types
- Numeric Data Types
- Date and Time Data Types etc...

Note: Data Types vary from One Database Management System to another

106. SQL Operators

Operators are used to perform Arithmetic, Comparison and Logical Operations.

- Arithmetic Operators
- Comparison Operators
- Logical Operators

107. SQL Functions

SQL has many built-in functions for performing processing on data.

- Aggregate Functions
- String Functions
- Date Functions Etc...

108. SQL Joins

The SQL Joins clause is used to combine records from two or more tables in a database.

Different SQL Joins:

- INNER JOIN (EQUI JOIN and NON-EQUIJOIN)
- Left Join
- Right Join

- Full Join
- Cross Join
- Self-Join

Popular Database Management Systems:

109. Oracle:

Oracle database is a relational database management system (RDBMS) developed by Oracle Corporation. Important Oracle editions are,

- Enterprise Edition: Offers all features, including superior performance and security.
- Standard Edition: Contains base functionality for users.
- Express Edition: The lightweight, free and limited Windows and Linux edition
- Oracle Lite: For mobile devices

110. Microsoft SQL Server

Microsoft SQL Server is a relational database management system developed by Microsoft. Important Microsoft SQL Server editions are,

- Datacenter: SQL Server 2008 R2 Datacenter is a full-featured edition of SQL Server.
- Enterprise: SQL Server Enterprise Edition includes both the core database engine and add-on services.
- Standard: SQL Server Standard edition includes the core database engine, along with the stand-alone services.
- Web: SQL Server Web Edition is for Web hosting.
- Workgroup: SQL Server Workgroup Edition includes the core database functionality only.

111. MySQL

MySQL is an Open Source Relational SQL database management system used for developing web-based software applications. Important MySQL Editions are,

- Standard Edition: Standard Edition enables us to deliver high-performance and scalable Online Transaction Processing (OLTP) applications.
- Enterprise Edition: Enterprise Edition includes the most comprehensive set of advanced features and management tools.

- Cluster Carrier Grade Edition: Cluster enables users to meet the database challenges of next generation web, cloud, and communications services with uncompromising scalability, uptime and agility.

112. PostgreSQL

PostgreSQL is a powerful, open-source database management system. It runs on all major operating systems, including Linux, UNIX, Mac OS X, Solaris, and MS Windows.

113. MS Access

Microsoft Access is bundled as part of the Microsoft Office suite. It is only available on the PC version. It is a desktop database system because its functions are intended to be run from a single computer

114. What is Data?

In computing, data is information that has been translated into a form that is efficient for movement or processing. Data is defined as facts or figures, or information that's stored in or used by a computer.

115. What is a Database?

- A Database is a systematic collection of data.
- Databases support storage and manipulation of data.
- Databases make data management easy.

116. Database Objects in Relational databases

A database object in a relational database is a data structure used to either store or reference data. Most of the major database engines offer the same set of major database object types:

- Tables
- Indexes
- Sequences
- Views
- Synonyms

117. Database Server

- Database Server instance contains multiple databases
- Database contains multiple Tables
- Table contains multiple records (rows and columns)

118. Table and Record

- A Table in a Relational Database is a predefined format of rows and columns that define an entity.
- Each column contains a different type of attribute and each row corresponds to a single record.
- Each Table is provided with a name.

Example: Table name: Students

SID	Name	Address	Contact-No
1001	Venkat	Abcd, xyz	9876989890
1002	Ramya	Asdf, ghj	8888887676
1003	Basha	Ert,	yuio

119. What is DBMS?

- A Database management System is software designed to assist in maintaining and utilizing a large collection of data.
- The alternative to using a DBMS is to store the data in files and write application-specific code to manage it.

Using a DBMS to manage data has many advantages:

- Data Independence
- Efficient Data Access
- Data Integrity and security
- Data Administration
- Concurrent Access and Data Recovery etc...

120. Types of Database Management System

Types of Database Management System

- Hierarchical DBMS
- Network DBMS
- Relational Database Management System (RDBMS)
- Example: Oracle, MS SQL Server, MySQL, DB2, etc...

- Non-Relational Database Management System
- Example: MongoDB, Apache Cassandra, Redis, Couchbase, and Apache HBase etc,

121. What are the types of databases?

- SQL/Relational databases
- NoSQL/Non-Relational databases
- Cloud databases
- Object-oriented databases
- Centralised database.
- Distributed database.
- Personal database.
- Commercial database.

122. What is Big Data?

Big Data is a collection of data that is huge in volume, yet growing exponentially with time. It is data with so large size and complexity that none of the traditional data management tools can store it or process it efficiently. Big data is also data but with huge size.

Types of Big Data:

- Structured data
- Unstructured data
- Semi-structured data

123. SQL Language Elements

- **Identifiers:** Names of Database Objects such as Tables, Views, Columns etc...
- **Data Types:** Define the type of data that is contained by a Column
- **Constants:** Symbols that represent specific data types
- **Operators:** Perform Arithmetic, Comparison, and Logical Operations
- **Functions:** Built-in Functions to perform specific operations
- **Clauses:** Constituent components of statements and queries.
- **Expressions:** Produce scalar values, or tables containing of columns and rows of data.
- **Queries:** Retrieve the data based on specific criteria, this an important element of SQL.

- **Statements etc...**

Data Definition Language Queries:

124. Create a Database

Syntax: Create Database databaseName;

Example: Create Database practiceDB;

125. Use Database

Syntax: Use databaseName;

Example: Use practiceDB;

126. Rename a Database

Syntax: Alter Database databaseName Modify Name = newdatabaseName;

Example: Alter Database practiceDB Modify Name = mydb;

127. Drop a Database

Syntax: Drop Database databaseName;

Example: Drop Database gcreddyDB;

128. Create a Table

Syntax: Create Table tableName(column1_name dataType(size),column2_name dataType(size),...);

Example: Create Table Students(STID int(5),STName char(50));

129. View Table info

Syntax: Select * from tablename;

Example: Select * from Students

130. View Table Schema

Syntax: desc tablename;

Example: desc Student;

131. Rename Table

Syntax: ALTER TABLE old_tablename RENAME TO new_tablename;

Example: ALTER TABLE Students RENAME TO newStudents;

132. Add a Column to the existing Table

Syntax: alter Table table_name add (column_name dataType(size));

Example: alter Table newStudents add (City char(50));

133. Add multiple columns to an existing Table

Syntax: Alter Table table_name add column1_name dataType(size), column2_name dataType(size);

Example: alter Table newStudents add add1 char(100), add2 char(70);

134. Modify an existing column

Syntax: alter table <table name> modify (column datatype (width));

Example: alter table student modify (sgroup varchar(10));

135. Rename a Column

Syntax: alter table tablename rename existing columnname to new columnname;

Example: alter table student rename column sid to StudentId;

136. Drop a Column

Syntax: Alter Table table_name Drop Column column_name;

Example: Alter Table newStudents Drop Column City;

137. Truncate a Table:

Truncate Table command is used to delete complete data from an existing table.

Syntax: Truncate Table table_name;

Example: Truncate Table newstudents;

138. Rename a Table:

It is a DDL Command, which is used to change the name of the data base table.

Syntax: rename Old tablename to new tablename

Example: SQL > rename student tostudent1;

139. Drop a Table:

Drop Table command is used to delete the complete Table (Data and Table Structure) from the Database.

Syntax: Drop Table table_name;

Example: Drop Table newStudents;

i. **Data Manipulation Language Queries:** Data Control Language etc...

Data Manipulation Language commands are used to store, modify, retrieve, and delete data

from database tables. In this category we have Select, Insert, Update, Delete Commands...

140. Inserting values for all columns:

Syntax: insert into tablename values(Column1value,Column2value,Column3value,.....);

Example: insert into employee values(1001,'krishna',50000,'05-may-2020');

o/p: 1 row created.

141. Inserting values to specific columns:

Syntax: insert into tablename(columnname,columnname) values(columndata, columndata);

Example: SQL> insert into employee (eid,ename) values(1003,'Supriya');

o/p: 1 row created.

Note: The other columns are assigned with "Null values".

142. Inserting values for all columns at a time:

Syntax: insert into tablename values(&column1name,&column2name....);

Column1value:

Column2value:

Example: SQL> insert into employee values(&eid,'&ename','&salery','&doj');

Enter value for eid: 1005

Enter value for ename: archana

Enter value for salery: 61454

Enter value for doj: 25-jan-1999

o/p: 1 row created.

143. Update command:

Syntax: SQL>update <table name> set colname -values where <condition>;

Example: SQL> update emp set da = sal * 10/100;

SQL> update emp set da = sal * 20/100 where ename = 'smith';

144. Delete all rows from a table command:

Syntax: SQL>delete from <table name> ;

Example: SQL> delete from student;

145. Delete specific rows from a table command:

Syntax: SQL>delete from <table name> where <condition>;

SQL> delete from student where Sno=101;

NOTE: Delete command removes all rows but the structure of a table remains unchanged.

146. To select all fields from a table:

Syntax: SELECT * FROM table_name;

Example: SELECT * from students;

147. To select specific fields from a table:

Syntax: SELECT field1,field2 FROM table_name;

Example: SELECT name, age FROM students

148. To select specific fields from a table using conditions:

Syntax: SELECT field1,field2 FROM table_name WHERE conditions;

Example: SELECT name, age FROM students WHERE age>10 ORDER BY name;

149. To select specific fields from multiple tables:

Syntax: SELECT expressions FROM table1 INNER JOIN table2 ON join_predicate;

Example: SELECT students.name, teachers.subject FROM students INNER JOIN teachers ON students.student_id = teacher_id ;ORDER BY name;

150. What is the difference between a left join and a right join?

A left join returns all rows from the left table and matching rows from the right table, while a right join returns all rows from the right table and matching rows from the left table.

151. What is a correlated subquery and how is it used?

A correlated subquery is a subquery that references a column from the outer query. It is used to retrieve data based on the values in the outer query, such as finding the most recent order for each customer.

152. What is a database index and how is it used?

A database index is a data structure used to speed up data retrieval by allowing the database engine to quickly locate specific data based on the values in one or more columns. It is used to improve query performance and reduce data access time.

153. What is a view and how is it used?

A view is a virtual table that is derived from one or more tables or other views in a database. It is used to simplify complex queries, limit data access, and provide a level of abstraction between the user and the underlying data.

154. What is a database trigger and how is it used?

A database trigger is a special type of stored procedure that automatically executes in response to certain events, such as inserting, updating, or deleting data in a table. It is used to enforce business rules, audit changes, or automate complex tasks.

155. What is normalization and why is it important in database design?

Normalization is the process of organizing data in a database to minimize redundancy and improve data integrity. It is important in database design because it helps to reduce data anomalies, improve data consistency, and simplify data maintenance.

156. What is a full outer join and how is it used?

A full outer join returns all rows from both tables being joined, along with matching rows from each table where available. It is used to combine data from two tables where not all rows have a matching value.

157. What is a constraint and how is it used?

A constraint is a rule that limits the type or value of data that can be inserted, updated, or deleted in a table. It is used to enforce data integrity and prevent data inconsistencies.

158. What is the difference between a clustered and a non-clustered index?

A clustered index determines the physical order of data in a table based on the values of the indexed columns, while a non-clustered index contains a separate data structure that maps the indexed columns to the corresponding rows in the table.

159. What is a materialized view and how is it used?

A materialized view is a precomputed result set that is stored on disk for fast data retrieval. It is used to speed up complex queries and reduce response time for data analysis.

160. What is a data warehouse and how is it different from a traditional database?

A data warehouse is a centralized repository that stores large amounts of historical and current data for analysis and decision-making. It is optimized for read-intensive workloads and data analysis, while a traditional database is optimized for transactional processing.

161. What is the difference between a group by and a having clause?

A group by clause is used to group the results of a query by one or more columns, while a having clause is used to filter the results of a group by based on some condition.

162. What is a foreign key and how is it used?

A foreign key is a column or combination of columns that references a primary key in another table. It is used to establish a relationship between two tables, ensuring data integrity and enforcing referential integrity constraints.

163. What is a self-join and how is it used?

A self-join is a join where a table is joined with itself using a common column. It is used to retrieve related data from the same table, such as finding all employees who have the same manager.

164. What is a primary key in SQL?

A primary key is a column or set of columns in a table that uniquely identifies each row in the table. Primary keys are used to establish relationships between tables and enforce data integrity.

165. What is the difference between a DELETE and TRUNCATE statement in SQL?

A DELETE statement is used to delete specific rows from a table, while a TRUNCATE statement is used to delete all data from a table, effectively resetting the table to its initial state.

166. How do you select distinct values from a column?

Example: `SELECT DISTINCT column_name FROM table_name;`

167. How do you sort a query by a specific column?

Example: `SELECT column1, column2 FROM table_name ORDER BY column1;`

168. How do you limit the number of rows returned in a query?

Example: `SELECT column1, column2 FROM table_name LIMIT 10;`

169. How do you join multiple tables in a query?

Example: `SELECT * FROM table1 JOIN table2 ON table1.column_name = table2.column_name;`

170. How do you create a primary key for a table?

Example: `CREATE TABLE table_name (id INT PRIMARY KEY, column1 VARCHAR(255), column2 VARCHAR(255));`

171. How do you create a foreign key for a table?

Example: CREATE TABLE table2 (id INT, FOREIGN KEY (id) REFERENCES table1(id));

172. How do you create an index in a table?

Example: CREATE INDEX index_name ON table_name (column_name);

173. How can you use the GROUP BY clause in SQL?

Example: SELECT column1, aggregate_function(column2) FROM table_name GROUP BY column

174. How can you sort the results of a query in SQL?

Example: SELECT column1, column2, ... FROM table_name ORDER BY column1 [ASC|DESC], column2 [ASC|DESC], ...;

175. How can you use the BETWEEN operator in SQL?

Example: SELECT column1, column2, ... FROM table_name WHERE column_name BETWEEN value1 AND value2;

176. How can you use the LIKE operator in SQL?

Example: SELECT column1, column2, ... FROM table_name WHERE column_name LIKE 'pattern';

177. How can you use the LIMIT clause in SQL?

Example: SELECT column1, column2, ... FROM table_name LIMIT number_of_rows;

178. How can you use the NOT IN operator in SQL?

Example: SELECT column1, column2, ... FROM table_name WHERE column_name NOT IN (value1, value2, ...);

179. How can you use the IS NULL operator in SQL?

Example: SELECT column1, column2, ... FROM table_name WHERE column_name IS NULL;

180. How can you use the UNION operator in SQL?

Example: SELECT column1, column2, ... FROM table1 UNION SELECT column1, column2, ... FROM table2;

181. How can you use the AVG() function in SQL?

Example: SELECT AVG(column_name) FROM table_name;

182. What is Data Integrity?

Data integrity is referred to the data's precision and consistency (validity) over its lifecycle. It ensures an organization's data's accuracy, completeness, consistency, and validity.

Every time data is copied or moved, it should remain unchanged and unaltered between updates. Error checking methods and validation procedures are typically responsible for ensuring the integrity of data transmitted or reproduced without the purpose of alteration.

183. What are ACID Properties?

Atomicity: means that the transaction takes place at once or doesn't happen at all;

Consistency: means that integrity constraints must be maintained so that the database is consistent before and after the transaction;

Isolation: ensures that each transaction is independent and occurs without interference;

Durability: ensures that after a transaction has completed execution, the updates and modifications to the database are stored in and written to disk, and they persist even if a system failure occurs.

184. Difference between CHAR and VARCHAR2

The CHAR data type stores character values. It is a fixed-length data type and has a maximum length of 2000 bytes of characters. Because char is a fixed size, it will often lead to memory wastage. (If you have a char(10), and want to store "Alex" in it will result in "Alex"). It has 6 additional blank characters.

The VARCHAR2 also stores character values, but it has variable length and can have a maximum length of 4000 bytes of characters. It helps in saving memory bytes.

185. SQL COUNT function

Example: SELECT COUNT(NAME) FROM employee WHERE manager_id = 1;

186. Different Types of Relationships?

One-to-one: occurs when each row in the user table has only one corresponding row in the password table.

One-to-many: one-to-many relationship occurs when one record in the class table is related to one or more records in the student table. However, one record in the student table cannot be related to more than one record in the class table.

Many-to-many: multiple records in one table (books table) are related to various records in another table (authors table).

187. What is a Live Lock?

Locks are essential parts of databases. If our application scales, it will receive many requests, which may result in a situation where two or more sessions will try to edit the same row. That may result in some data integrity problems.

A Live lock happens when a request for an exclusive lock is denied continuously because a sequence of other shared locks keeps interfering with each other and changing the status. Therefore no one will complete the transaction.

An excellent example of a live lock would be two cars meeting on a one-way street, both of them will turn left or right at the same time to let the other car pass, but as these actions are done at the same time, neither of them will advance.

188. SQL Query to Find Names Starting with a Substring?

```
select name from employee where name like 'a%'
```

189. SQL Query to Find and Delete Duplicate Records from a Table?

Example: `DELETE employee1 FROM employee_2 employee1 INNER JOIN employee_2 employee2 WHERE employee1.id < employee2.id AND employee1.email = employee2.email;`

190. How to Remove Duplicate Rows from a Query?

`SELECT DISTINCT` fetches all the distinct data from a query. For example, let's say that we want all the distinct countries where our employees are located. For this task, we will use the following SQL query:

Example: `SELECT DISTINCT country FROM employee`

191. How to Use LIKE operator in SQL?

The `LIKE` operator is used in `WHERE` clauses to check if a value fits a given string pattern. Here is an example of a `LIKE` operator:

Example: `SELECT * FROM employee WHERE name like "Alexandru";`

With this command, we can pull all the records where the first name is like "Alexandru". The result will look like this:

192. What is a Schema in SQL?

- Our database holds a lot of different objects, such as tables, relations, triggers, indexes, their relationships and so on. So, we usually make a database schema to create a clear vision of our database and organize all tables and relations between them.
- So, we can view schema as the logical relationship between all the different tables in the database. Overall, we can consider a schema as a blueprint for the database. Usually, the schema is the first thing created when creating a database.

193. What are UNION, MINUS and INTERACT commands?

- The **UNION** operator merges the results of two tables and eliminates duplicate rows from the tables.
- The **MINUS** returns the rows from the first query but not from the second query.
- The **INTERSECT** returns the rows that are returned by both of the queries.

194. Differences between Views and Tables?

Views and tables may seem pretty similar, but they have some differences. Both of them consist of rows and columns, but they differ in terms of persistence.

A **table** stores and retrieves data from persistent storage whenever the user wants. On the other side, the view is only a virtual table based on the result of some SQL queries, meaning that this will disappear after the session ends.

195. Difference between ORDER BY and GROUP BY clauses?

Using **ORDER BY**, we can sort the result set in ascending or descending order.

GROUP BY clause is used to group the rows with the same value.

196. Difference between WHERE and HAVING?

With the **WHERE** clause, we can filter the result set according to given conditions. It can be used with SELECT, UPDATE, and DELETE statements.

Example: `SELECT ID, NAME FROM EMPLOYEE WHERE ID >= 100`

HAVING clause is used as a filter with the GROUP BY statement. Those grouped rows that will satisfy the given condition in HAVING will appear in the final result. HAVING Clause can only be used with SELECT statement.

Example: `SELECT ID, NAME FROM EMPLOYEE GROUP BY NAME HAVING AGE > 18;`

197. What are constraints? Different types of Constraints?

Constraints are the rules imposed on the database contents and processes to ensure data integrity. There are different types of constraints:

Domain constraint specifies the attribute's domain or set of values (if we define a number column, we cannot insert 'A' in the column).

Tuple Uniqueness constraint specifies that all the rows must be necessarily unique in any table.

Key constraint defines that every primary key should be unique and not null.

Entity Integrity constraint specifies that no attribute that composes the primary key must contain a null value in any relation.

Referential Integrity constraint establishes that all the foreign key values must either be in the relation to the primary key or be null.

198. What is a View in a Database?

A view is a virtual table in the database that contains data from one or more tables, but it takes less memory because it is not physically stored in the database. The names of the views are always unique. Database views are saved in the database as named queries and can be used to save frequently used complex queries. Views are an excellent tool to restrict access to the data in such a way that a user can see and (sometimes) modify exactly what they need and no more.

Example: CREATE VIEW EMPLOYEE_VIEW AS SELECT ID, NAME, DOB FROM EMPLOYEE;

199. Difference between SQL and PL/SQL?

- Even though SQL and PL/SQL (Procedural Language extension to SQL) are pretty similar, there are several differences in how they work, they differ in performance, error handling capabilities, and interaction with the databases. For example:
- SQL is a query language with few keywords. And PL/SQL is a programming language using SQL for a database and has variables, data types, loops etc. PL/SQL also offers error and exception handling features that do not exist in SQL.
- SQL is declarative; PL/SQL is procedural.

- While SQL executes one query at a time, PL/SQL can run an entire block or multiple operations in a single execution.
- PL/SQL does not interact directly with the database server, while SQL does.
- Generally, SQL is used for DDL and DML statements, whereas PL/SQL is used to write program blocks, functions, procedures, triggers and packages.

200. Difference between Relational and Non-relational Databases

A **relational database** is organized in tables. Usually, the data within these tables have relationships with one another or dependencies.

A **non-relational database** is document-oriented, meaning all information gets stored in more of a laundry list order. As a result, we will have all our data listed within a single construct or document.

201. What is a Query?

A query, in SQL, is a command used to request data or update information from a database table or combination of tables. Users can write and execute queries to retrieve, delete or update information in a database. Usually, a query response returns data from different tables within the database, but this does not apply every time (for example, a delete query will not return anything).

For example, let's say that we have an employee table. We want to get all employees from the table that are born before 1990. For this case, we will use a SELECT statement because all we want is to retrieve data from a table.

Example: `SELECT * FROM employees WHERE year_born < 1990`

202. What is a Primary Key?

- A primary key refers to a column or a set of columns containing values uniquely identifying each row in a table. A database table uses a primary key to uniquely insert, update, restore or delete data from a database table.
- Generally, a primary key is a sequentially generated long number and its whole purpose is to uniquely identify a particular row in the table. It does not have any business value in itself.
- We can define a primary key in the employee table as follows. In this example, we are marking ID column as the primary key for the table Employee.

Example: CREATE TABLE Employee (id INT PRIMARY KEY, name VARCHAR(45), email VARCHAR(45), manager_id INT, country VARCHAR(45));

203. What is a Unique Key?

A unique key refers to a value of one column (or combination of more than one column) that uniquely identifies a record in the database table. Not like primary key, a unique key has a business meaning and puts a certain constraint on the table as defined by the business.

For example, for an EMPLOYEE table, an EMAIL can be a unique key because each employee is guaranteed to have a unique key throughout the organization.

Example: CREATE TABLE Employee (id INT PRIMARY KEY, name VARCHAR(45), email VARCHAR(45) UNIQUE, manager_id INT, country VARCHAR(45));

If the table and the column already exist, we can add the constraint in the following way:

Example: ALTER TABLE Employee ADD UNIQUE (email);

204. Difference between Primary key and Unique Key?

- A primary key column cannot contain NULL values, but the unique key can accept one NULL value.
- Every database table can have just one primary key, but multiple unique keys.
- The primary key generates a clustered index, but the unique key generates a non-clustered index.

205. What is a Foreign Key?

A FOREIGN KEY is a column (or collection of columns) in one table that links the PRIMARY KEY in another table.

We can add a foreign key to our employee table on the manager_id column because a manager is also an employee. We will link the manager_id column with the id from the same table. If we are creating a new table we can implement the foreign key in the following way:

Example: CREATE TABLE Employee (id INT PRIMARY KEY, name VARCHAR(45), email VARCHAR(45) UNIQUE, manager_id INT, country VARCHAR(45), FOREIGN KEY (manager_id) REFERENCES Employee(id));

And if we already have the table and the column created we can add the constraint altering the table in the following way:

Example: ALTER TABLE employee ADD FOREIGN KEY (manager_id) REFERENCES Employee(id);

206. What are Database Normalization and Denormalization?

Database normalization is an essential process that consists of structuring a relational database by a sequence of so-called normal forms to ease redundancy and improve data integrity. To say that the data from our database is normalized, it should meet the following two requirements:

- No redundancy of data. The data is held in just one place and is not duplicated in other tables.
- Data dependencies are logical and separated. All related data are stored together.
- There are a few rules for database normalization. Each rule is called a “normal form.” If the first rule is observed, the database is said to be in “first normal form.” If the first three rules are observed, the database is considered to be in “third normal form.” The third normal form is considered the highest level necessary for most applications.

Here is a **list** of Normal Forms in SQL:

1NF (First Normal Form): Each table cell should contain a single value. Each record needs to be unique.

2NF (Second Normal Form): In addition to 1NF, all non-key attributes are fully dependent on the primary key. It helps to eliminate redundant data.

3NF (Third Normal Form): In addition to 2NF, there are no transitive functional dependencies. In 3NF, values in a record that are not part of that record’s key do not belong in the table.

BCNF (Boyce-Codd Normal Form): is just a more strong form of 3NF. Sometimes also referred to as 3.5NF.

4NF (Fourth Normal Form): In addition to BCNF, the database has no multi-valued dependency.

5NF (Fifth Normal Form) In addition to 4NF, the database does not contain any join dependency, joining should be lossless.

On the opposite, **denormalization** is the exact opposite process of normalization. Here, we intentionally add redundancy to the data to improve the specific application's performance and protect data integrity.

Although denormalization is not recommended, it may be necessary in some very specific cases.

207. Arithmetic Operators:

Arithmetic operators can perform arithmetical operations on numeric operands involved.

Operator	Description	Example
+ Addition	Adds values on either side of the operator	$a + b$ will give 30
- Subtraction	Subtracts right hand operand from left hand operand	$a - b$ will give -10
*	Multiplies values on either side of the operator	$a * b$ will give 200
/ Division	Divides left hand operand by right hand operand	b / a will give 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	$b \% a$ will

208. Comparison Operators:

A comparison (or relational) operator is a mathematical symbol that is used to compare between two values. Comparison operators are used in conditions that compare one expression with another. The result of a comparison can be TRUE, FALSE, or UNKNOWN (an operator that has one or two NULL expressions returns UNKNOWN).

Operator	Description	Example
=	Checks if the values of two operands are equal or not, if yes then condition becomes true.	$(a = b)$ is not true. $(10 = 20)$
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	$(a != b)$ is true. $(10 != 20)$
\diamond	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	$(a \diamond b)$ is true. $(10 \diamond 20)$

>	Checks if the value of left operand is greater than the value of right operand, if yes then	($a > b$) is not true. ($10 > 20$)
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	($a < b$) is true. ($10 < 20$)
\geq	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	($a \geq b$) is not true. ($10 \geq 20$)
\leq	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	($a \leq b$) is true. ($10 \leq 20$)
$!<$	Checks if the value of left operand is not less than the value of right operand, if yes then	($a !< b$) is false. ($10 !< 20$)

209. Logical Operators

There are three Logical Operators namely, AND, OR, and NOT. These operators compare two conditions at a time to determine whether a row can be selected for the output. When retrieving data using a SELECT statement, You can use logical operators in the WHERE clause, which allows you to combine more than one condition.

OPERATOR	DESCRIPTION
ALL	The ALL operator is used to compare a value to all values in another value set.
AN D	The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
AN Y	The ANY operator is used to compare a value to any applicable value in the list according to the condition.
BETWE EN	The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.
EXISTS	
	[313]

IN	<p>The EXISTS operator is used to search for the presence of a row in a specified table that meets certain criteria.</p>
IN	<p>The IN operator is used to compare a value to a list of literal values that have been specified.</p>
LIKE	<p>The LIKE operator is used to compare a value to similar values using wildcard operators.</p>
NOT	<p>The NOT operator reverses the meaning of the logical operator with which it is used.</p>
	<p>[Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. This is a negate operator.]</p>
OR	<p>The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.</p>
IS NULL	<p>The NULL operator is used to compare a value with a NULL value.</p>
UNIQUE	<p>The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).</p>

210. What is an Alias in SQL?

An alias is a feature of SQL that is supported by most, if not all, RDBMSs. It is a temporary name assigned to the table or table column for the purpose of a particular SQL query. In addition, aliasing can be employed as an obfuscation technique to secure the real names of database fields. A table alias is also called a correlation name.

An alias is represented explicitly by the AS keyword but in some cases, the same can be performed without it as well. Nevertheless, using the AS keyword is always a good practice.

Syntax for column: Column_name AS alias_name;

Here, column_name: original name of the column and alias_name: temporary name

```
SQL> SELECT C.ID AS CUSTID, C.NAME AS CUSTNAME, O.OID AS ORDERID
```

```
FROM CUSTOMERS C JOIN ORDERS O ON C.ID = O.ID;
```

Syntax for table:

```
CREATE TABLE new_table_name AS SELECT * FROM existing_table_name;
```

```
SQL> CREATE TABLE orderNew AS SELECT * FROM orders;
```

Table created.



*****THE END*****