Unit 1. Introduction (DBMS) (NOTES)

OVERVIEW OF DBMS

DATA - Data is nothing but facts and statistics stored or free-flowing over a network. Generally it's raw and unprocessed. Data becomes information when it is processed, turning it into something meaningful.

DATABASE - The database is a collection of inter-related, data which is used to retrieve, insert & delete the data efficiently. It is also used to organize the data in the form of table, schema, views and reports etc.

Example - The college Database organizes the data about the admin, staff, students & faculty etc

Example of database are -:

- Microsoft Access
- Oracle
- My SQL (Structured Query Language).
- foxPro etc.

The main characteristics of the database Approach are -:

- Self-describing nature of the database
- Programs and data are insulated such that change in the data doesn't require a change in the program.
- Same data can be shown differently to different viewers.
- Data can be shared between the users & transactions can be interleaved maintaining consistency.

(DBMS) DATABASE MANAGEMENT SYSTEM

A Database management System (DBMS) is a software tool that enables the users to manage a data in database easily. It is also called computerized record-keeping system. It allows users to access and interact with the underlying / desired data in the Database.

Furthermore, DBMS allows users to interact with a database securely and concurrently without interfering with each user and, while maintaining data integrity.

PURPOSE Of DATABASE SYSTEM (3)

The main purpose of Database System is to manage the data. for ex:- Consider a university that keeps the data of students, teachers, courses, books etc. To manage this data, we need to stored this data somewhere where we can add new data, retrieve data, to perform these operations on data we need a database management system that allow us to store the data in such a way so that all these operations can be performed on the data effectively & efficient

CHARACTERISTICS Of DATABASE SYSTEM

1. Data stored into Tables - Data is never directly stored into the database. Data is stored into tables, created inside the database.

2. Reduced Redundancy - In the modern world, hard-drives are very cheap, but earlier when hard-drives were too expensive, unnecessary repitition of data in database was a big problem. But DBMS follows normalization which divides the data in such a way that repetition is minimum.

3. Data consistency - On Live data, i.e. data that is being continuously updated & added maintaining the consistency of the data can become a challenge. But DBMS handles it all by itself.

4. Support for Multiple views of data (Different views of same data can be created to cater the needs of different users).

5. Restriction of unauthorized access.

6. Data Security and Integerity

7. ACID Properties - DBMS adheres to the concept of Acid Properties (Atomicity, Consistency, Isolation and Durability).

8. Query Language - DBMS provide users with a simple Query language, using which data can be fetched, Inserted, deleted and updated in a database.

ADVANTAGES Of DBMS ⑤

1. Data sharing - In DBMS, the authorized users of an organisation can share the data among multiple users.

2. Reduce time - It reduces development time and maintenance need.

3. Easily Maintenance - It can be easily maintainable due to the centralized nature of the database system.

4. Controls database redundancy - It can control data redundancy because it stores all the data in one single database file and that recorded data is placed in the database.

5. Backup and Recovery - It provide backup & recovery subsystems which create automatic backup of data from hardware & software failures and restores the data if required

6. Multiple user Interface - It provides different types of user interfaces like GUI Graphical User Interfaces, API (Application program interfaces

DISADVANTAGES Of DBMS ⑥

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

APPLICATIONS - DBMS

1. Telecom Sector - There is a database Syste to keep the track of the Information, regarding calls made, network usage, custo mes, details etc.

2. Industry - where there is a manufacturing unit, warehouse or distribution centre each one needs a database to keep the records of its Prducts transactions etc.

3. Banking System - for storing customer, Information tracking, day to day credit and debit transactions, generating Bank statements etc.

4. Airlines - To travel through airlines, we make ~~cost~~ early reservations, this reservati on information along with flight schedule is stored in Database.

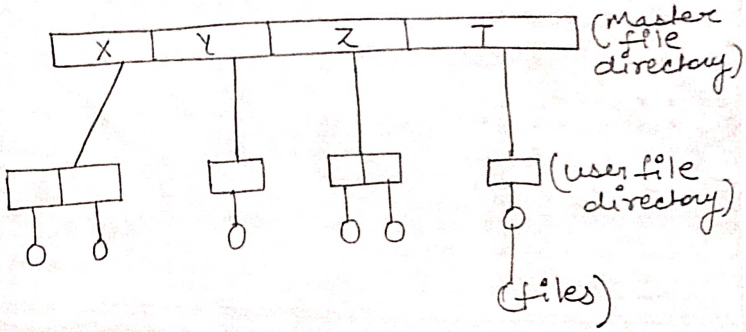
5. Education Sector - Database Systems are frequently used in Schools & Colleges to store and retrieve the data regarding student details, staff details, exam & ee details.

FILE SYSTEM

The file System is basically a way of arranging the files in a storage medium like a hard disk. The file system organizes the files and helps in the retrieval of files, when they are required.

File System consists of different files, which are grouped into directories. The Directories further contain other folders & files. The file system perform basic oper n like file naming, management etc.

Example:- NTfs (New Technology file System)
EXT. (Extended file System)



DBMS & FILE SYSTEM (9)

BASIS	DBMS	FILE SYSTEM
Structure	DBMS or Database Management System is a software that is used for accessing, creating and managing databases.	A file system is software that manages & organizes the files in a storage medium within a computer.
Data Redundancy	In DBMS, the redundancy of data is low.	In file system the redundancy of data is greater.
Data Consistency	There is more data consistency because of the process of normalization.	There is less data consistency, in the file system
Backup & Recovery	DBMS System provides backup & recovery of data even if it is lost.	It doesn't offer backup and Recovery of data if it is lost.

(10)

BASIS	DBMS	FILE SYSTEM
Query Processing	Efficient Query-processing is there in DBMS Using SQL. (Structured Query-language).	There is no efficient query processing in the file System
User Access	Multiple users can access the data at a time.	Only one user can access the data at a time
Security Constraints	Database Management System has more Security Mechanism as compared to file System.	file system provides less security as compared to DBMS. (Data base, Management System).
Data Interdependency	In DBMS, Data Independence exist.	There is no data-Independence.
Transactions	Easy to Implement Complicated Transactions.	Not provide Support for complicated Transactions

INSTANCE IN DBMS -: (12)

The data stored in database at a particular moment of time is called instance of database. Database Schema Defines the Variable declarations in tables that belong to a particular database, the value of these variables at a moment of time is called the Instance of that database.

Data Abstraction in DBMS (13)

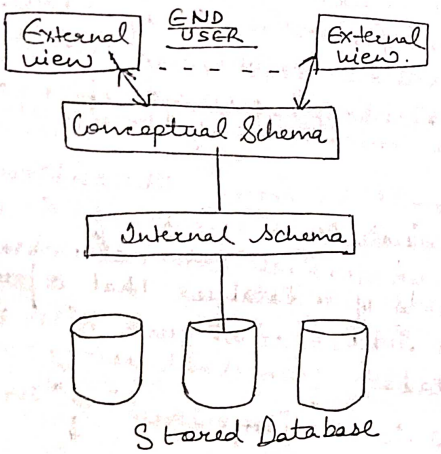
Database System are made up of complex data structures. To ease the user Interaction with Database, the developers hide Internal irrelevant details from users. The process of hiding irrelevant details from user is called data abstraction.

Example of data abstraction

A Switchboard is an example of data-abstraction. (i.e) It hides all the details of the Circuitry of Current flow and provide a simple way to switch ON and OFF electrical appliances.

Abstraction is one of the main feature of Database System.

THREE SCHEMA LEVEL ARCHITECTURE



The Goal of the three-Schema architecture is to seperate the user applications from the physical Database.

1. The internal level - The Internal level has an internal Schema, which Describes the physical storage structure of the database. It also describes the complex details of data storage and access paths for the database. This is also called lowest level of Architecture

2. The Conceptual level - The Conceptual level has a conceptual Schema, which describes the structure of the whole database for a community of users. The conceptual schema, hides the details of physical storage structures and concentrates on describing entities, Data types, relationships, user operations etc. This level is maintained by DBA (Database Administrator)

3. The external or view level - The external or view level includes a number of external Schemas or user views. Each external schema's Describe the parts of a database that a particular user group is interested in and hides the rest of the database from that user group.

External level is the 'top level' of the Architecture.

ARCHITECTURE Of DBMS

- Database Management System architecture, helps us to understand the components of Database System & relation among them.

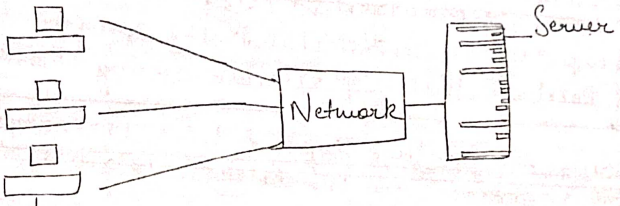
DBMS Architecture depends how the users are connected to the database to get their request/query done.

DBMS ARCHITECTURE ARE Of THREE TYPES

- Single-tier Architecture.
- Two-tier Architecture.
- Three-tier Architecture.

BASIC CLIENT/SERVER

CLIENT/SERVER ARCHITECTURE



Server - A server is a system containing both hardware & software that can provide services to the client machine, i.e. file access or database access.

The basic client/server architecture is used to deal with a large number of PC's (Workstation), web servers, database servers & other component that are connected with network.

client request the server for a service through a network and server provides the request service to the client.

client - A client is typically a user machine that provides user interface capabilities, and local processing.

SINGLE OR ONE TIER ARCHITECTURE

- In single tier or One tier architecture, the database is directly available to the user. The client, server and database all resides or available on the same machine. i.e. (the user can directly sit on the DBMS and uses it).
- Any changes done here will directly be done on the database itself. No Network connection is required to perform the action on the database.
- One tier Architecture is used for development of the local application, where programmer/user can directly communicate with the database for the quick response. But such architecture is rarely used in production.
- It is also used where Data Doesnot change frequently and where no multiple user is their in accessing the system.

Example :- A simple one tier architecture example would be anytime you install a database in your system and access it to practice SQL queries.

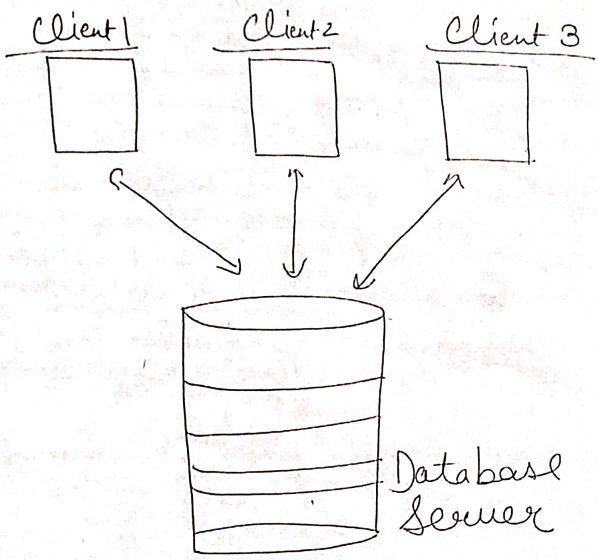
TWO-TIER ARCHITECTURE

(layer)

This Architecture is called two tier because the software component are Distributed over two systems: client and server.

The user interface programs & Application program can run on the client side, when DBMS access is required, the program establishes a connection to the DBMS (which is on the server side). Once the connection is created, the client program can Communicate with DBMS.

A standard called ODBC (Open Database Connectivity), JDBC (Java database Connectivity), provides an API (Application programming Interface), which allows client-side program to call the DBMS, as long as both client and server Machines have the neccessary Software Installed.



2 tier Architecture

(2 layer :- client & Server layer)

- The 2 tier architecture is used inside any organization, where multiple client accessing the database directly.

-) Example :- Railway Reservations from counter, where clerk as a client accesses the railway Server directly.

Advantage :-

- Direct and faster communication.
- Maintenance and understanding is easier.
- Compatible with existing System.

Disadvantage :-

- Scalability i.e. It gives poor performance when there are a large number of users.
- less secure as client can access the Server directly.

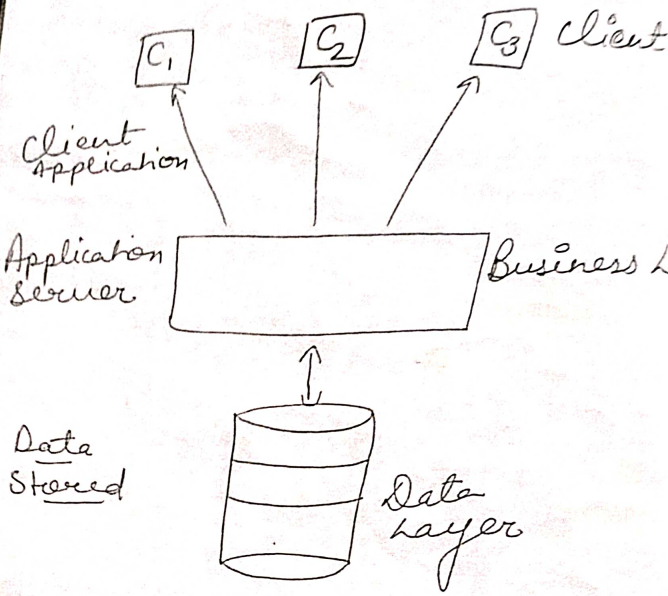
THREE - TIER ARCHITECTURE
(Layer)

The three tier architecture consist of 3 layers;
- client layer
- Business Layer/Application layer.
- Database Server Layer

3 tier Architecture design is an extension of 2 tier client/server architecture.

A 3-Tier Architecture in DBMS is the most popular client server architecture in which the Development and maintenance of functional processes, data access, data storage and user Interface is done Independently as Seperate Modules.

The Business or Application layer resides between the user and the DBMS, which is Responsible for communicating the user's request to the DBMS System & Send the Respons from the DBMS to the User. The Application ayer (Business layer) also processes functional ogic, constraints and rules before passing ata to the user or Down to the DBMS.



A 3 Tier Architecture

DATA-INDEPENDENCE

- Data Independence means a change of data at one level should not affect another level.

Two types of Data Independence -:

① Physical Data independence-
② Conceptual Data Independence.

---

① Physical Data Independence -:
Any change in the physical location of tables and indexes should not affect the conceptual level or external view of data. This data-independence is easy to achieve and implemented by most of the DBMS (Database Management system).

② Conceptual Data Independence :-
The data at conceptual level schema and external level schema must be independent. This means a change in conceptual schema level should not affect the external schema level.

e.g. Adding or deleting attributes of a table should not affect the user's view of a table. But this type of Independence is difficult to achieve as compared to physical data Independence.

DATA MODELS -:

Hierarchical Data Model

In this type of data model, the data is organised into a tree-like structure that has a single root and the data is linked to the root. In this model, the main hierarchy begins from the root and it expands like a tree that has child nodes & further expan in the same manner.

In this model, the child node has one single parent node but one parent can have multiple child nodes. As the data is stored like tree structure in this data model, when data is retrieved the whole tree is traversed from the root node. The hierarchical data model contains a one-to-many relationship between various types of data. The data is stored in the form of a record & is connected through links.

The Main Drawback of this model is that, it can have only one to many relationship between nodes.

2. RELATIONAL MODEL -

(This model was initially described by Edgar f. Codd, in 1969).

- It uses collection of Tables for representing data and the relationships among these data.
- Data is stored in Tables called Relations
- Each table is a group of column & rows, where columns represent attribute of an entity and rows represents records (or tuples)

□ Attribute or field - Each column in a relation is called an attributes. The value of the attribute should be from the same Domain

Example - We have Different attributes of the student like Student_Id, Student_Name, Student_Age etc.

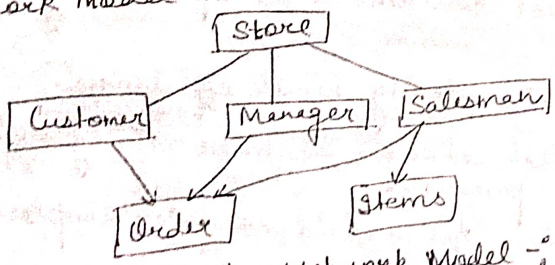
□ Record or Tuple - Each row in a relation Called Tuple. A Tuple Defines a collection of attribute values. So each row in a relation contains unique value.

Example :- Each row has all the Information about any specific Individual like the first row has Information about Student Anil.

NETWORK MODEL (43)

The Network model was created to represent complex data relationships more effectively when compared to hierarchical models, to improve database performance and standards.
It has Entities which are organized in a graphical representation and some entities are accessed through several paths. A user perceives the network model as a collection of records in 1:M Relationships (Many)

Pictorial Representation of Network Model in DBMS



The main feature of a Network Model -
1) Ability to Merge Relationships - In this Model, because of more relationships the data is more relate
It has an ability to manage one to one relationships as well as many-to-many relationships.
2) Many paths - There can be more than one path to the same record because of more relationship
It makes data access fast & simple.

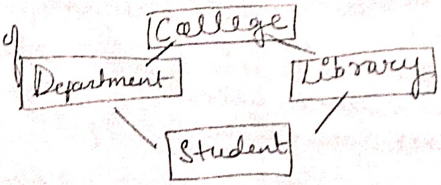
Advantages of Network Model

1) Network Model represent complex data relationships better than the hierarchical models
2) It handles so many relationship types. (1:M).
3) Data access is more flexible than hierarchical models.
4) It improved database performance.
5) It includes Data Definition Language (DDL) & Data Manipulation Language (DML) standards.

Disadvantage of Network Model

1) Database contains a complex array of pointers.
2) System complexity limit efficiency.
3) Structural changes require changes in a application programs.
4) Navigation System yield complex implemen & management.
5) Any change like updating, deletion, Insert is very complex.

example of Network Model



ENTITY - RELATIONSHIP (ER) MODEL

- ER Model is a high-level Data model Diagram.
- ER Model describes the structure of a database with the help of a diagram, which is known as ER (Entity Relationship) Diagram.
- An ER Model is a Design or blueprint of a database that can be later Implemented as a database.
- It is based on the notion of the real world entities & relationship among them.

ER Diagram has the following three Components:

- Entities - Entity is a real-world thing or object. It can be a person, place or even a concept.

Example - Teacher, Students, Course, Building Department etc are some of the entities of a School Management System.

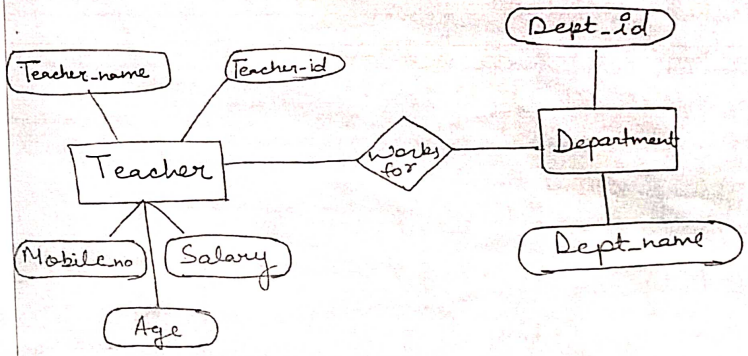
- Attributes - An entity contain real-world property called attributes. This is the characteristics of that attribute.

Example:- The entity teacher has the property like teacher_id, name, salary, age etc.

- Relationships - Relationship tells how two attributes are related.

Example : Teacher work for a Department.

Example :- ER Model



a) The entities are Teacher & Department.

b) The attribute of Teacher entity are Teacher-id, Age, salary, Mobile_no.

c) The attribute of Department entity are Dept_id, Dept_name.

d) The two entities are connected to each other the relationships. Here each Teacher works for a Department.

SQL

What is Database?
Database is a collection of interrelated data.

What is DBMS?
DBMS (*Database Management System*) is software used to create, manage, and organize databases.

What is RDBMS?

- RDBMS (Relational *Database Management System)* - is a DBMS based on the concept of tables (also called relations).
- Data is organized into tables (also known as relations) with rows (records) and columns (attributes).
- Eg - MySQL, PostgreSQL, Oracle etc.

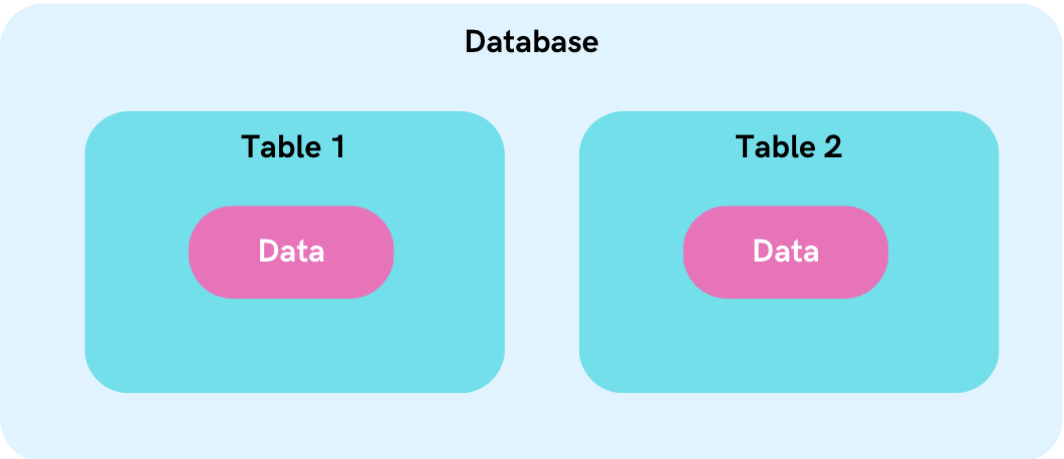
What is SQL?
SQL is *Structured Query Language* - used to store, manipulate and retrieve data from RDBMS.
(It is not a database, it is a language used to interact with database)

We use SQL for *CRUD* Operations :

- **CREATE** - To create databases, tables, insert tuples in tables etc
- **READ** - To read data present in the database.
- **UPDATE** - Modify already inserted data.
- **DELETE** - Delete database, table or specific data point/tuple/row or multiple rows.

*Note - SQL keywords are NOT case sensitive. Eg: select is the same as SELECT in SQL.

SQL v/s MySQL
SQL is a language used to perform CRUD operations in Relational DB, while MySQL is a RDBMS that uses SQL.



SQL Data Types
In SQL, data types define the kind of data that can be stored in a column or variable.

To See all data types of MYSQL, visit :
https://dev.mysql.com/doc/refman/8.0/en/data-types.html

Here are the frequently used SQL data types:

DATATYPE	DESCRIPTION	USAGE
CHAR	string(0-255), can store characters of fixed length	CHAR(50)
VARCHAR	string(0-255), can store characters up to given length	VARCHAR(50)
BLOB	string(0-65535), can store binary large object	BLOB(1000)
INT	integer(-2,147,483,648 to 2,147,483,647)	INT
TINYINT	integer(-128 to 127)	TINYINT
BIGINT	integer(-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)	BIGINT
BIT	can store x-bit values. x can range from 1 to 64	BIT(2)
FLOAT	Decimal number - with precision to 23 digits	FLOAT
DOUBLE	Decimal number - with 24 to 53 digits	DOUBLE
BOOLEAN	Boolean values 0 or 1	BOOLEAN
DATE	date in format of YYYY-MM-DD ranging from 1000-01-01 to 9999-12-31	DATE
TIME	HH:MM:SS	TIME
YEAR	year in 4 digits format ranging from 1901 to 2155	YEAR

*Note - CHAR is for fixed length & VARCHAR is for variable length strings. Generally, VARCHAR is better as it only occupies necessary memory & works more efficiently.

We can also use UNSIGNED with datatypes when we only have positive values to add.
Eg - UNSIGNED INT

Types of SQL Commands:

1. **DQL** *(Data Query Language)* : Used to retrieve data from databases. (SELECT)
2. **DDL** *(Data Definition Language)* : Used to create, alter, and delete database objects like tables, indexes, etc. (CREATE, DROP, ALTER, RENAME, TRUNCATE)
3. **DML** *(Data Manipulation Language)*: Used to modify the database. (INSERT, UPDATE, DELETE)
4. **DCL** *(Data Control Language)*: Used to grant & revoke permissions. (GRANT, REVOKE)
5. **TCL** *(Transaction Control Language)*: Used to manage transactions. (COMMIT, ROLLBACK, START TRANSACTIONS, SAVEPOINT)

---

1. Data Definition Language (DDL)

Data Definition Language (DDL) is a subset of SQL (Structured Query Language) responsible for defining and managing the structure of databases and their objects.

DDL commands enable you to create, modify, and delete database objects like tables, indexes, constraints, and more.

Key DDL Commands are:

- **CREATE TABLE:**
    - Used to create a new table in the database.
    - Specifies the table name, column names, data types, constraints, and more.
    - Example:
      CREATE TABLE employees (id INT PRIMARY KEY, name VARCHAR(50), salary DECIMAL(10, 2));

- **ALTER TABLE:**
    - Used to modify the structure of an existing table.
    - You can add, modify, or drop columns, constraints, and more.
    - Example: ALTER TABLE employees ADD COLUMN email VARCHAR(100);

- **DROP TABLE:**
    - Used to delete an existing table along with its data and structure.
    - Example: DROP TABLE employees;

- **CREATE INDEX:**
    - Used to create an index on one or more columns in a table.
    - Improves query performance by enabling faster data retrieval.
    - Example: CREATE INDEX idx_employee_name ON employees (name);
- **DROP INDEX:**
    - Used to remove an existing index from a table.
    - Example: DROP INDEX idx_employee_name;
- **CREATE CONSTRAINT:**
    - Used to define constraints that ensure data integrity.
    - Constraints include PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL, and CHECK.
    - Example: ALTER TABLE orders ADD CONSTRAINT fk_customer FOREIGN KEY (customer_id) REFERENCES customers(id);
- **DROP CONSTRAINT:**
    - Used to remove an existing constraint from a table.
    - Example: ALTER TABLE orders DROP CONSTRAINT fk_customer;
- **TRUNCATE TABLE:**
    - Used to delete the data inside a table, but not the table itself.
    - Syntax – TRUNCATE TABLE table_name

2. DATA QUERY/RETRIEVAL LANGUAGE (DQL or DRL)

DQL (Data Query Language) is a subset of SQL focused on retrieving data from databases.

The SELECT statement is the foundation of DQL and allows us to extract specific columns from a table.

- **SELECT:**

The SELECT statement is used to select data from a database.

Syntax: SELECT column1, column2, ... FROM table_name;

Here, column1, column2, ... are the field names of the table.

If you want to select all the fields available in the table, use the following syntax:
SELECT * FROM table_name;

Ex: SELECT CustomerName, City FROM Customers;

- **WHERE:**

The WHERE clause is used to filter records.

Syntax: SELECT column1, column2, ... FROM table_name WHERE condition;

Ex: SELECT * FROM Customers WHERE Country='Mexico';

Operators used in WHERE are:

= : Equal
> : Greater than
< : Less than
>= : Greater than or equal
<= : Less than or equal
<> : Not equal.

Note: In some versions of SQL this operator may be written as !=

- **AND, OR and NOT:**

- The WHERE clause can be combined with AND, OR, and NOT operators.
- The AND and OR operators are used to filter records based on more than one condition:
- The AND operator displays a record if all the conditions separated by AND are TRUE.
- The OR operator displays a record if any of the conditions separated by OR is TRUE.
- The NOT operator displays a record if the condition(s) is NOT TRUE.

Syntax:

SELECT column1, column2, ... FROM table_name WHERE condition1 AND condition2 AND condition3 ...;

SELECT column1, column2, ... FROM table_name WHERE condition1 OR condition2 OR condition3 ...;

SELECT column1, column2, ... FROM table_name WHERE NOT condition;

Example:

SELECT * FROM Customers WHERE Country='India' AND City='Japan';

SELECT * FROM Customers WHERE Country='America' AND (City='India' OR City='Korea');

- **DISTINCT:**

Removes duplicate rows from query results.

Syntax: SELECT DISTINCT column1, column2 FROM table_name;

- **LIKE**:

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

There are two wildcards often used in conjunction with the LIKE operator:

- The percent sign (%) represents zero, one, or multiple characters
- The underscore sign (_) represents one, single character

Example: SELECT * FROM employees WHERE first_name LIKE 'J%';

WHERE CustomerName LIKE 'a%'
- Finds any values that start with "a"

WHERE CustomerName LIKE '%a'
- Finds any values that end with "a"

WHERE CustomerName LIKE '%or%'
- Finds any values that have "or" in any position

WHERE CustomerName LIKE '_r%'
- Finds any values that have "r" in the second position

WHERE CustomerName LIKE 'a_%'

- Finds any values that start with "a" and are at least 2 characters in length

WHERE CustomerName LIKE 'a__%'

- Finds any values that start with "a" and are at least 3 characters in length

WHERE ContactName LIKE 'a%o'

- Finds any values that start with "a" and ends with "o"

- **IN:**

Filters results based on a list of values in the WHERE clause.

Example: SELECT * FROM products WHERE category_id IN (1, 2, 3);

- **BETWEEN**:

Filters results within a specified range in the WHERE clause.

Example: SELECT * FROM orders WHERE order_date BETWEEN '2023-01-01' AND '2023-06-30';

- **IS NULL**:

Checks for NULL values in the WHERE clause.

Example: SELECT * FROM customers WHERE email IS NULL;

- **AS**:

Renames columns or expressions in query results.

Example: SELECT first_name AS "First Name", last_name AS "Last Name" FROM employees;

- **ORDER BY**

The ORDER BY clause allows you to sort the result set of a query based on one or more columns.

Basic Syntax:

- The ORDER BY clause is used after the SELECT statement to sort query results.

- Syntax: SELECT column1, column2 FROM table_name ORDER BY column1 [ASC|DESC];

Ascending and Descending Order:

- By default, the ORDER BY clause sorts in ascending order (smallest to largest).
- You can explicitly specify descending order using the DESC keyword.
- Example: SELECT product_name, price FROM products ORDER BY price DESC;

Sorting by Multiple Columns:

- You can sort by multiple columns by listing them sequentially in the ORDER BY clause.
- Rows are first sorted based on the first column, and for rows with equal values, subsequent columns are used for further sorting.
- Example: SELECT first_name, last_name FROM employees ORDER BY last_name, first_name;

Sorting by Expressions:

- It's possible to sort by calculated expressions, not just column values.
- Example: SELECT product_name, price, price * 1.1 AS discounted_price FROM products ORDER BY discounted_price;

Sorting NULL Values:

- By default, NULL values are considered the smallest in ascending order and the largest in descending order.
- You can control the sorting behaviour of NULL values using the NULLS FIRST or NULLS LAST options.
- Example: SELECT column_name FROM table_name ORDER BY column_name NULLS LAST;

Sorting by Position:

- Instead of specifying column names, you can sort by column positions in the ORDER BY clause.
- Example: SELECT product_name, price FROM products ORDER BY 2 DESC, 1 ASC;

- **GROUP BY**

The GROUP BY clause in SQL is used to group rows from a table based on one or more columns.

Syntax:

- The GROUP BY clause follows the SELECT statement and is used to based on specified columns.

- Syntax: SELECT column1, aggregate_function(column2) FROM table_name GROUP BY column1;

- Aggregation Functions:
    - Aggregation functions (e.g., COUNT, SUM, AVG, MAX, MIN) are often used with GROUP BY to calculate values for each group.
    - Example: SELECT department, AVG(salary) FROM employees GROUP BY department;
- Grouping by Multiple Columns:

    - You can group by multiple columns by listing them in the GROUP BY clause.
    - This creates a hierarchical grouping based on the specified columns.
    - Example: SELECT department, gender, AVG(salary) FROM employees GROUP BY department, gender;

- HAVING Clause:

    - The HAVING clause is used with GROUP BY to filter groups based on aggregate function results.
    - It's similar to the WHERE clause but operates on grouped data.
    - Example: SELECT department, AVG(salary) FROM employees GROUP BY department HAVING AVG(salary) > 50000;

- Combining GROUP BY and ORDER BY:

    - You can use both GROUP BY and ORDER BY in the same query to control the order of grouped results.
    - Example: SELECT department, COUNT(*) FROM employees GROUP BY department ORDER BY COUNT(*) DESC;

- **AGGREGATE FUNCTIONS**

These are used to perform calculations on groups of rows or entire result sets. They provide insights into data by summarising and processing information.

Common Aggregate Functions:

- COUNT():
  Counts the number of rows in a group or result set.

- SUM():
  Calculates the sum of numeric values in a group or result set.

- AVG():

Computes the average of numeric values in a group or result set.

- MAX():
  Finds the maximum value in a group or result set.

- MIN():
  Retrieves the minimum value in a group or result set.

3. DATA MANIPULATION LANGUAGE

Data Manipulation Language (DML) in SQL encompasses commands that manipulate data within a database. DML allows you to insert, update, and delete records, ensuring the accuracy and currency of your data.

- **INSERT:**

- The INSERT statement adds new records to a table.
- Syntax: INSERT INTO table_name (column1, column2, ...) VALUES (value1, value2, ...);

- Example: INSERT INTO employees (first_name, last_name, salary) VALUES ('John', 'Doe', 50000);

- **UPDATE**:

- The UPDATE statement modifies existing records in a table.
- Syntax: UPDATE table_name SET column1 = value1, column2 = value2, ... WHERE condition;
- Example: UPDATE employees SET salary = 55000 WHERE first_name = 'John';

- **DELETE**:

- The DELETE statement removes records from a table.
- Syntax: DELETE FROM table_name WHERE condition;
- Example: DELETE FROM employees WHERE last_name = 'Doe';

4. Data Control Language (DCL)

Data Control Language focuses on the management of access rights, permissions, and security-related aspects of a database system.

DCL commands are used to control who can access the data, modify the data, or perform administrative tasks within a database.

DCL is an important aspect of database security, ensuring that data remains protected and only authorised users have the necessary privileges.

There are two main DCL commands in SQL: GRANT and REVOKE.

1. GRANT:

The GRANT command is used to provide specific privileges or permissions to users or roles. Privileges can include the ability to perform various actions on tables, views, procedures, and other database objects.

Syntax:

```
GRANT privilege_type
ON object_name
TO user_or_role;
```

In this syntax:

- privilege_type refers to the specific privilege or permission being granted (e.g., SELECT, INSERT, UPDATE, DELETE).
- object_name is the name of the database object (e.g., table, view) to which the privilege is being granted.
- user_or_role is the name of the user or role that is being granted the privilege.

Example: Granting SELECT privilege on a table named "Employees" to a user named "Analyst":

```
GRANT SELECT ON Employees TO Analyst;
```

2. REVOKE:

The REVOKE command is used to remove or revoke specific privileges or permissions that have been previously granted to users or roles.

Syntax:

```
REVOKE privilege_type
ON object_name
```

FROM user_or_role;

In this syntax:

- privilege_type is the privilege or permission being revoked.
- object_name is the name of the database object from which the privilege is being revoked.
- user_or_role is the name of the user or role from which the privilege is being revoked.

Example: Revoking the SELECT privilege on the "Employees" table from the "Analyst" user:

REVOKE SELECT ON Employees FROM Analyst;

DCL and Database Security:

DCL plays a crucial role in ensuring the security and integrity of a database system.

By controlling access and permissions, DCL helps prevent unauthorised users from tampering with or accessing sensitive data. Proper use of GRANT and REVOKE commands ensures that only users who require specific privileges can perform certain actions on database objects.

5. Transaction Control Language (TCL)

Transaction Control Language (TCL) deals with the management of transactions within a database.
TCL commands are used to control the initiation, execution, and termination of transactions, which are sequences of one or more SQL statements that are executed as a single unit of work.
Transactions ensure data consistency, integrity, and reliability in a database by grouping related operations together and either committing or rolling back changes based on the success or failure of those operations.

There are three main TCL commands in SQL: COMMIT, ROLLBACK, and SAVEPOINT.

1. COMMIT:

The COMMIT command is used to permanently save the changes made during a transaction.

It makes all the changes applied to the database since the last COMMIT or ROLLBACK command permanent.
Once a COMMIT is executed, the transaction is considered successful, and the changes are made permanent.

Example: Committing changes made during a transaction:

```
UPDATE Employees
SET Salary = Salary * 1.10
WHERE Department = 'Sales';

COMMIT;
```

2. ROLLBACK:

The ROLLBACK command is used to undo changes made during a transaction.
It reverts all the changes applied to the database since the transaction began.

ROLLBACK is typically used when an error occurs during the execution of a transaction, ensuring that the database remains in a consistent state.

Example: Rolling back changes due to an error during a transaction:

```
BEGIN;

UPDATE Inventory
SET Quantity = Quantity - 10
WHERE ProductID = 101;

-- An error occurs here

ROLLBACK;
```

3. SAVEPOINT:

The SAVEPOINT command creates a named point within a transaction, allowing you to set a point to which you can later ROLLBACK if needed.

SAVEPOINTs are useful when you want to undo part of a transaction while preserving other changes.

Syntax: SAVEPOINT savepoint_name;

Example: Using SAVEPOINT to create a point within a transaction:

```
BEGIN;
```

```
UPDATE Accounts
SET Balance = Balance - 100
WHERE AccountID = 123;

SAVEPOINT before_withdrawal;

UPDATE Accounts
SET Balance = Balance + 100
WHERE AccountID = 456;

-- An error occurs here

ROLLBACK TO before_withdrawal;

-- The first update is still applied

COMMIT;
```

TCL and Transaction Management:

Transaction Control Language (TCL) commands are vital for managing the integrity and consistency of a database's data.
They allow you to group related changes into transactions, and in the event of errors, either commit those changes or roll them back to maintain data integrity.
TCL commands are used in combination with Data Manipulation Language (DML) and other SQL commands to ensure that the database remains in a reliable state despite unforeseen errors or issues.

JOINS

In a DBMS, a join is an operation that combines rows from two or more tables based on a related column between them.
Joins are used to retrieve data from multiple tables by linking them together using a common key or column.

Types of Joins:

1. Inner Join
2. Outer Join
3. Cross Join
4. Self Join

1) Inner Join

An inner join combines data from two or more tables based on a specified condition, known as the join condition.
The result of an inner join includes only the rows where the join condition is met in all participating tables.
It essentially filters out non-matching rows and returns only the rows that have matching values in both tables.

Syntax:

```
SELECT columns
FROM table1
INNER JOIN table2
ON table1.column = table2.column;
```

Here:

- columns refers to the specific columns you want to retrieve from the tables.
- table1 and table2 are the names of the tables you are joining.
- column is the common column used to match rows between the tables.
- The ON clause specifies the join condition, where you define how the tables are related.

Example: Consider two tables: Customers and Orders.

Customers Table:

CustomerID	CustomerName
1	Alice
2	Bob
3	Carol

Orders Table:

OrderID	CustomerID	Product

101	1	Laptop
102	3	Smartphone
103	2	Headphones

Inner Join Query:

```
SELECT Customers.CustomerName, Orders.Product
FROM Customers
INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

Result:

CustomerName	Product
Alice	Laptop
Bob	Headphones
Carol	Smartphone

2) Outer Join

Outer joins combine data from two or more tables based on a specified condition, just like inner joins. However, unlike inner joins, outer joins also include rows that do not have matching values in both tables.
Outer joins are particularly useful when you want to include data from one table even if there is no corresponding match in the other table.

Types:

There are three types of outer joins: left outer join, right outer join, and full outer join.

1. Left Outer Join (Left Join):

A left outer join returns all the rows from the left table and the matching rows from the right table.

If there is no match in the right table, the result will still include the left table's row with NULL values in the right table's columns.

Example:

```
SELECT Customers.CustomerName, Orders.Product
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

Result:

CustomerName	Product
Alice	Laptop
Bob	Headphones
Carol	Smartphone
NULL	Monitor

In this example, the left outer join includes all rows from the Customers table.

Since there is no matching customer for the order with OrderID 103 (Monitor), the result includes a row with NULL values in the CustomerName column.

2. Right Outer Join (Right Join):

A right outer join is similar to a left outer join, but it returns all rows from the right table and the matching rows from the left table.

If there is no match in the left table, the result will still include the right table's row with NULL values in the left table's columns.

Example: Using the same Customers and Orders tables.

```
SELECT Customers.CustomerName, Orders.Product
FROM Customers
RIGHT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

Result:

CustomerName	Product
Alice	Laptop
Carol	Smartphone
Bob	Headphones
NULL	Keyboard

Here, the right outer join includes all rows from the Orders table. Since there is no matching order for the customer with CustomerID 4, the result includes a row with NULL values in the CustomerName column.

3. Full Outer Join (Full Join):

A full outer join returns all rows from both the left and right tables, including matches and non-matches.

If there's no match, NULL values appear in columns from the table where there's no corresponding value.

Example: Using the same Customers and Orders tables.

```
SELECT Customers.CustomerName, Orders.Product
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

Result:

CustomerName	Product
Alice	Laptop
Bob	Headphones
Carol	Smartphone

NULL	Monitor
NULL	Keyboard

In this full outer join example, all rows from both tables are included in the result. Both non-matching rows from the Customers and Orders tables are represented with NULL values.

3) Cross Join

A cross join, also known as a Cartesian product, is a type of join operation in a Database Management System (DBMS) that combines every row from one table with every row from another table.

Unlike other join types, a cross join does not require a specific condition to match rows between the tables. Instead, it generates a result set that contains all possible combinations of rows from both tables.

Cross joins can lead to a large result set, especially when the participating tables have many rows.

Syntax:

```
SELECT columns
FROM table1
CROSS JOIN table2;
```

In this syntax:

- columns refers to the specific columns you want to retrieve from the cross-joined tables.
- table1 and table2 are the names of the tables you want to combine using a cross join.

Example: Consider two tables: Students and Courses.

Students Table:

StudentID	StudentName
1	Alice

2	Bob

Courses Table:

CourseID	CourseName
101	Maths
102	Science

Cross Join Query:

```
SELECT Students.StudentName, Courses.CourseName
FROM Students
CROSS JOIN Courses;
```

Result:

StudentName	CourseName
Alice	Maths
Alice	Science
Bob	Maths
Bob	Science

In this example, the cross join between the Students and Courses tables generates all possible combinations of rows from both tables. As a result, each student is paired with each course, leading to a total of four rows in the result set.

4) Self Join

A self join involves joining a table with itself.

This technique is useful when a table contains hierarchical or related data and you need to compare or analyse rows within the same table.

Self joins are commonly used to find relationships, hierarchies, or patterns within a single table.

In a self join, you treat the table as if it were two separate tables, referring to them with different aliases.

Syntax:

The syntax for performing a self join in SQL is as follows:

```
SELECT columns
FROM table1 AS alias1
JOIN table1 AS alias2 ON alias1.column = alias2.column;
```

In this syntax:

- columns refers to the specific columns you want to retrieve from the self-joined table.
- table1 is the name of the table you're joining with itself.
- alias1 and alias2 are aliases you assign to the table instances for differentiation.
- column is the column you use as the join condition to link rows from the same table.

Example: Consider an Employees table that contains information about employees and their managers.

Employees Table:

EmployeeID	EmployeeName	ManagerID
1	Alice	3
2	Bob	3
3	Carol	NULL
4	David	1

Self Join Query:

```
SELECT e1.EmployeeName AS Employee, e2.EmployeeName AS Manager
FROM Employees AS e1
JOIN Employees AS e2 ON e1.ManagerID = e2.EmployeeID;
```

Result:

Employee	Manager

Alice	Carol
Bob	Carol
David	Alice

In this example, the self join is performed on the Employees table to find the relationship between employees and their managers. The join condition connects the ManagerID column in the e1 alias (representing employees) with the EmployeeID column in the e2 alias (representing managers).

SET OPERATIONS

Set operations in SQL are used to combine or manipulate the result sets of multiple SELECT queries.
They allow you to perform operations similar to those in set theory, such as union, intersection, and difference, on the data retrieved from different tables or queries.

Set operations provide powerful tools for managing and manipulating data, enabling you to analyse and combine information in various ways.

There are four primary set operations in SQL:

- UNION
- INTERSECT
- EXCEPT (or MINUS)
- UNION ALL

1. UNION:

The UNION operator combines the result sets of two or more SELECT queries into a single result set.
It removes duplicates by default, meaning that if there are identical rows in the result sets, only one instance of each row will appear in the final result.

Example:

Assume we have two tables: Customers and Suppliers.

Customers Table:

CustomerID	CustomerName
1	Alice
2	Bob

Suppliers Table:

SupplierID	SupplierName
101	SupplierA
102	SupplierB

UNION Query:

```
SELECT CustomerName FROM Customers
UNION
SELECT SupplierName FROM Suppliers;
```

Result:

CustomerName
Alice
Bob
SupplierA
SupplierB

2. INTERSECT:

The INTERSECT operator returns the common rows that exist in the result sets of two or more SELECT queries.

It only returns distinct rows that appear in all result sets.

Example: Using the same tables as before.

```
SELECT CustomerName FROM Customers
INTERSECT
SELECT SupplierName FROM Suppliers;
```

Result:

CustomerName

In this example, there are no common names between customers and suppliers, so the result is an empty set.

3. EXCEPT (or MINUS):

The EXCEPT operator (also known as MINUS in some databases) returns the distinct rows that are present in the result set of the first SELECT query but not in the result set of the second SELECT query.

Example: Using the same tables as before.

```
SELECT CustomerName FROM Customers
EXCEPT
SELECT SupplierName FROM Suppliers;
```

Result:

CustomerName
Alice
Bob

In this example, the names "Alice" and "Bob" are customers but not suppliers, so they appear in the result set.

4. UNION ALL:

The UNION ALL operator performs the same function as the UNION operator but does not remove duplicates from the result set. It simply concatenates all rows from the different result sets.

Example: Using the same tables as before.

```
SELECT CustomerName FROM Customers
UNION ALL
SELECT SupplierName FROM Suppliers;
```

Result:

CustomerName
Alice
Bob
SupplierA
SupplierB

Difference between Set Operations and Joins

Aspect	Set Operations	Joins
Purpose	Manipulate result sets based on set theory principles.	Combine data from related tables based on specified conditions.

Data Source	Result sets of SELECT queries.	Tables that are related by common columns.
Combining Rows	Combine rows from different result sets. May remove duplicates.	Combine rows from different tables based on specified conditions.
Output Columns	Require the SELECT queries to have the same number of output columns and compatible data types.	Can combine columns from different tables, regardless of data types or column numbers.
Common Operations	UNION, INTERSECT, EXCEPT (MINUS).	INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN.
Conditional Requirements	No specific join conditions are required.	Require specified join conditions for combining data.
Handling Duplicates	UNION removes duplicates by default.	Joins do not inherently handle duplicates; it depends on the join type and data.

Usage Scenarios	Useful for combining and analysing related data from different queries or tables.	Used to retrieve and relate data from different tables based on their relationships.
Result Set Structure	Result sets may have different column names, but data types and counts must match.	Result sets can have different column names, data types, and counts.
Performance Considerations	Generally faster and less complex than joins.	Joins can be more complex and resource-intensive, especially for larger datasets.

SUB QUERIES

Subqueries, also known as nested queries or inner queries, allow you to use the result of one query (the inner query) as the input for another query (the outer query).

Subqueries are often used to retrieve data that will be used for filtering, comparison, or calculation within the context of a larger query.

They are a way to break down complex tasks into smaller, manageable steps.

Syntax:

SELECT columns

FROM table
WHERE column OPERATOR (SELECT column FROM table WHERE condition);

In this syntax:

- columns refers to the specific columns you want to retrieve from the outer query.
- table is the name of the table you're querying.
- column is the column you're applying the operator to in the outer query.
- OPERATOR is a comparison operator such as =, >, <, IN, NOT IN, etc.
- (SELECT column FROM table WHERE condition) is the subquery that provides the input for the comparison.

Example: Consider two tables: Products and Orders.

Products Table:

ProductID	ProductName	Price
1	Laptop	1000
2	Smartphone	500
3	Headphones	50

Orders Table:

OrderID	ProductID	Quantity
101	1	2
102	3	1

For Example: Retrieve the product names and quantities for orders with a total cost greater than the average price of all products.

SELECT ProductName, Quantity
FROM Products
WHERE Price * Quantity > (SELECT AVG(Price) FROM Products);

Result:

ProductName	Quantity

Laptop	2

Differences Between Subqueries and Joins:

Aspect	Subqueries	Joins
Purpose	Retrieve data for filtering, comparison, or calculation within the context of a larger query.	Combine data from related tables based on specified conditions.
Data Source	Result of one query used as input for another query.	Data from multiple related tables.
Combining Rows	Not used for combining rows; used to filter or evaluate data.	Combines rows from different tables based on specified join conditions.
Result Set Structure	Subqueries return scalar values, single-column results, or small result sets.	Joins return multi-column result sets.
Performance Considerations	Subqueries can be slower and less efficient, especially when dealing with large datasets.	Joins can be more efficient for combining data from multiple tables.
Complexity	Subqueries can be easier to understand for simple tasks or smaller datasets.	Joins can become complex, but are more suited for handling large-scale data retrieval and combination tasks.
Versatility	Subqueries can be used in various clauses: WHERE, FROM, HAVING, etc.	Joins are primarily used in the FROM clause for combining tables.

DBMS

Code-501

UNIT 3:- FILE ORGANISATION

File Organization

- The **File** is a collection of records. Using the primary key, we can access the records. The type and frequency of access can be determined by the type of file organization which was used for a given set of records.
- File organization is a logical relationship among various records. This method defines how file records are mapped onto disk blocks.
- File organization is used to describe the way in which the records are stored in terms of blocks, and the blocks are placed on the storage medium.
- The first approach to map the database to the file is to use the several files and store only one fixed length record in any given file. An alternative approach is to structure our files so that we can contain multiple lengths for records.
- Files of fixed length records are easier to implement than the files of variable length records.

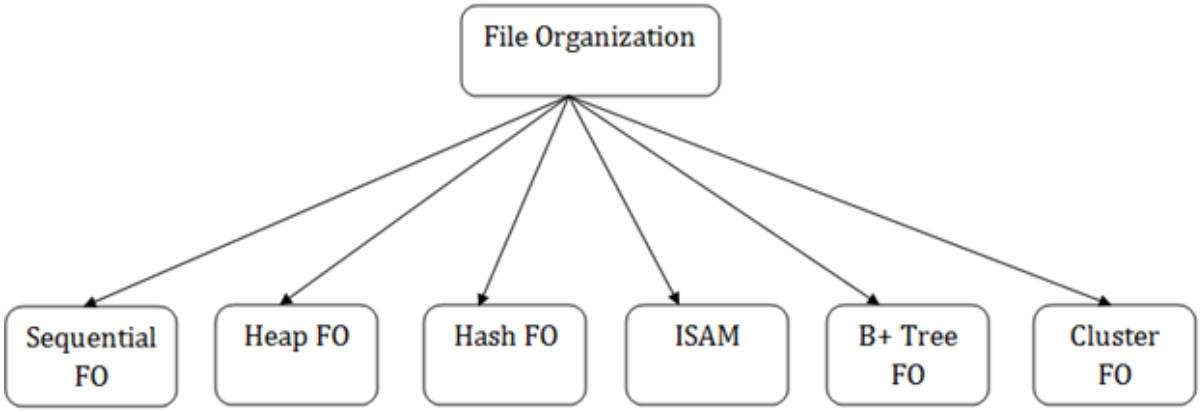
Objective of file organization

- It contains an optimal selection of records, i.e., records can be selected as fast as possible.
- To perform insert, delete or update transaction on the records should be quick and easy.
- The duplicate records cannot be induced as a result of insert, update or delete.
- For the minimal cost of storage, records should be stored efficiently.

Types of file organization:

File organization contains various methods. These particular methods have pros and cons on the basis of access or selection. In the file organization, the programmer decides the best-suited file organization method according to his requirement.

Types of file organization are as follows:

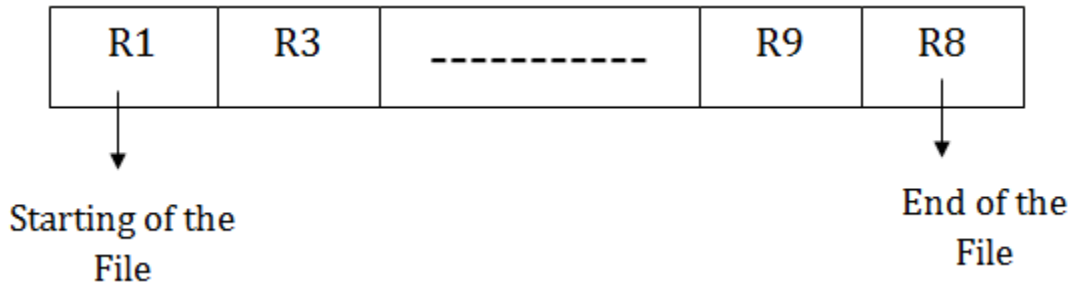


Sequential File Organization

This method is the easiest method for file organization. In this method, files are stored sequentially. This method can be implemented in two ways:

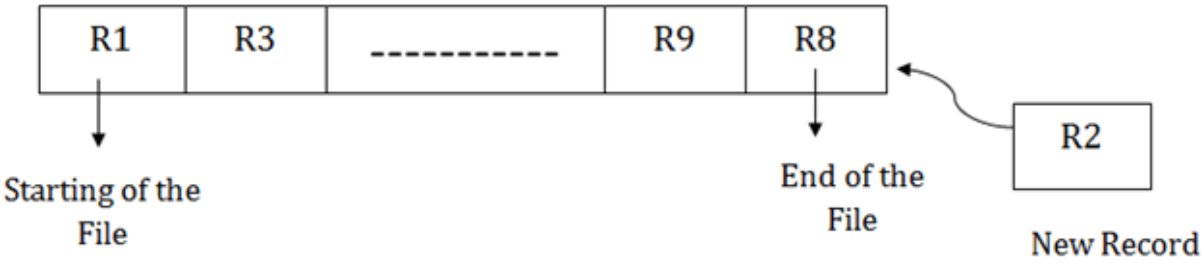
1. Pile File Method:

- It is a quite simple method. In this method, we store the record in a sequence, i.e., one after another. Here, the record will be inserted in the order in which they are inserted into tables.
- In case of updating or deleting of any record, the record will be searched in the memory blocks. When it is found, then it will be marked for deleting, and the new record is inserted.



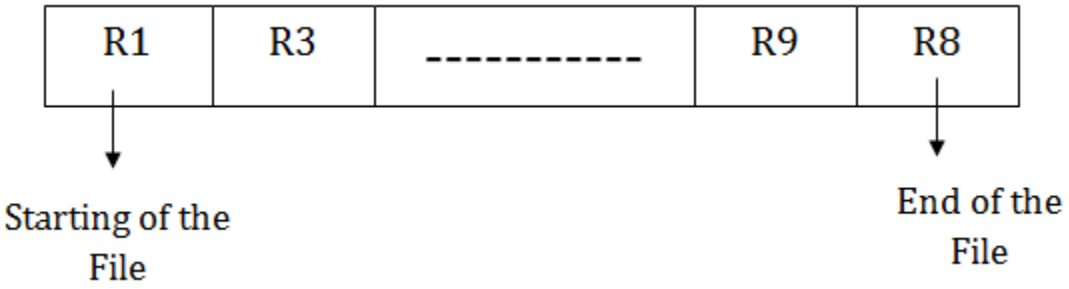
Insertion of the new record:

Suppose we have four records R1, R3 and so on upto R9 and R8 in a sequence. Hence, records are nothing but a row in the table. Suppose we want to insert a new record R2 in the sequence, then it will be placed at the end of the file. Here, records are nothing but a row in any table.



2. Sorted File Method:

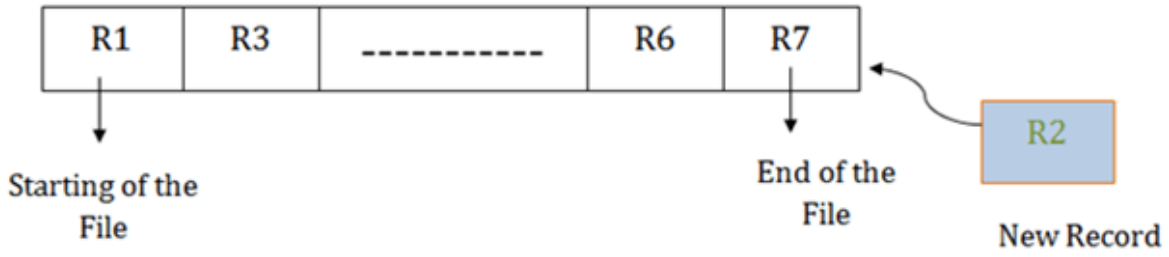
- In this method, the new record is always inserted at the file's end, and then it will sort the sequence in ascending or descending order. Sorting of records is based on any primary key or any other key.
- In the case of modification of any record, it will update the record and then sort the file, and lastly, the updated record is placed in the right place.

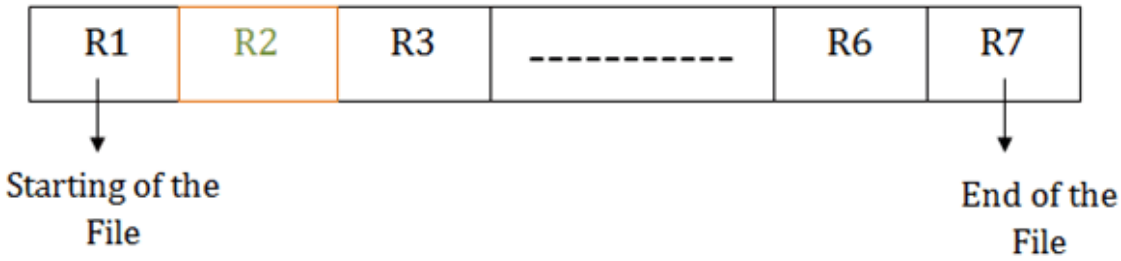


Insertion of the new record:

Suppose there is a preexisting sorted sequence of four records R1, R3 and so on upto R6 and R7. Suppose a new record R2 has to be inserted in the sequence, then it

inserted at the end of the file, an





Pros of sequential file organization

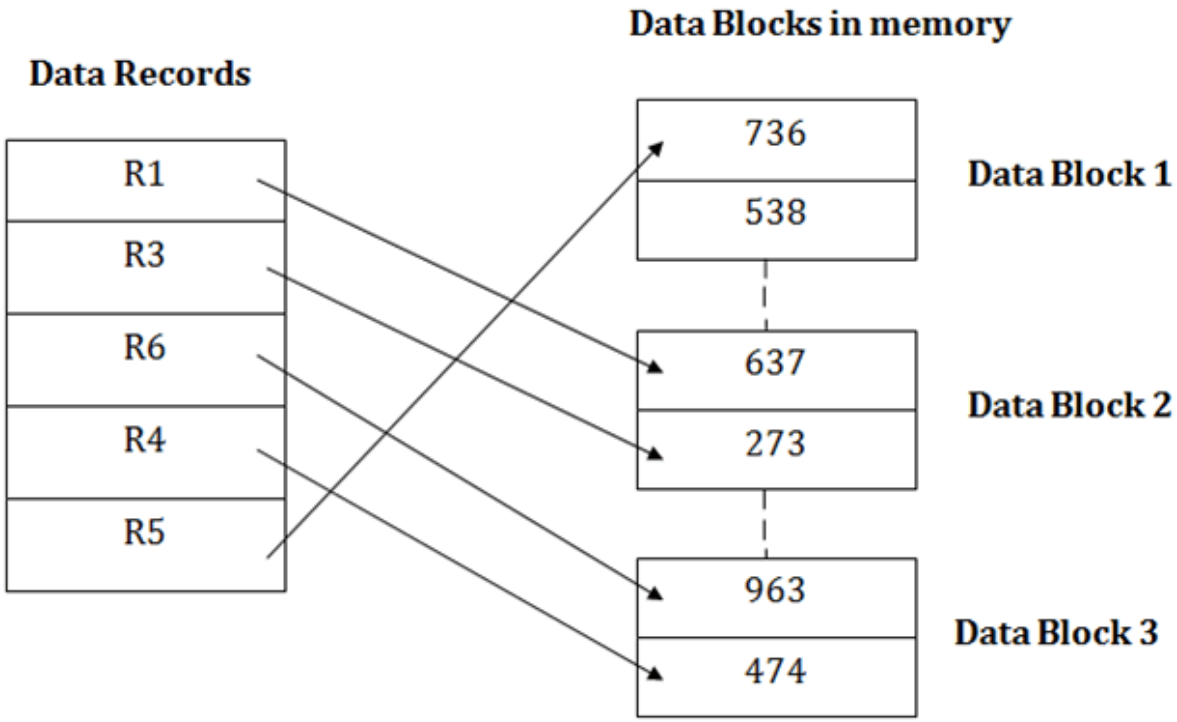
- It contains a fast and efficient method for the huge amount of data.
- In this method, files can be easily stored in cheaper storage mechanism like magnetic tapes.
- It is simple in design. It requires no much effort to store the data.
- This method is used when most of the records have to be accessed like grade calculation of a student, generating the salary slip, etc.
- This method is used for report generation or statistical calculations.

Cons of sequential file organization

- It will waste time as we cannot jump on a particular record that is required but we have to move sequentially which takes our time.
- Sorted file method takes more time and space for sorting the records.

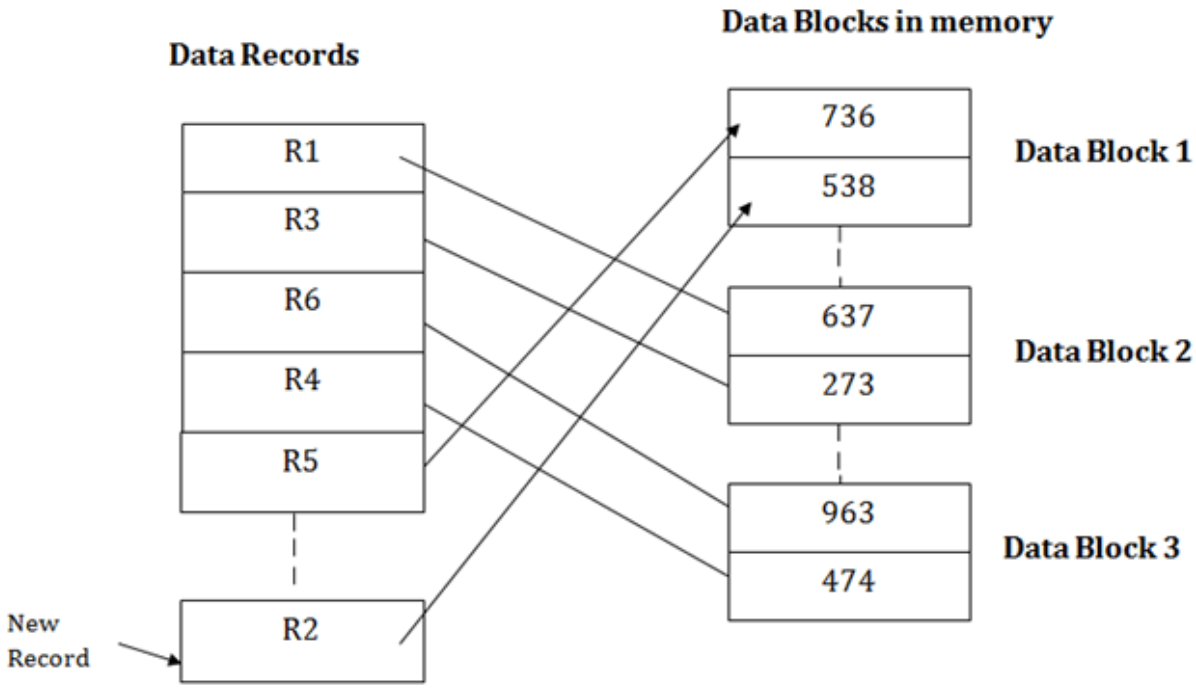
Heap file organization

- It is the simplest and most basic type of organization. It works with data blocks. In heap file organization, the records are inserted at the file's end. When the records are inserted, it doesn't require the sorting and ordering of records.
- When the data block is full, the new record is stored in some other block. This new data block need not to be the very next data block, but it can select any data block in the memory to store new records. The heap file is also known as an unordered file.
- In the file, every record has a unique id, and every page in a file is of the same size. It is the DBMS responsibility to store and manage the new records.



Insertion of a new record

Suppose we have five records R1, R3, R6, R4 and R5 in a heap and suppose we want to insert a new record R2 in a heap. If the data block 3 is full then it will be inserted in any of the database selected by the DBMS, let's say data block 1.



If we want to search, update or delete the data in heap file organization, then we need to traverse the data from staring of the file till we get the requested record.

If the database is very large then searching, updating or deleting of record will be time-consuming because there is no sorting or ordering of records. In the heap file organization, we need to check all the data until we get the requested record.

Pros of Heap file organization

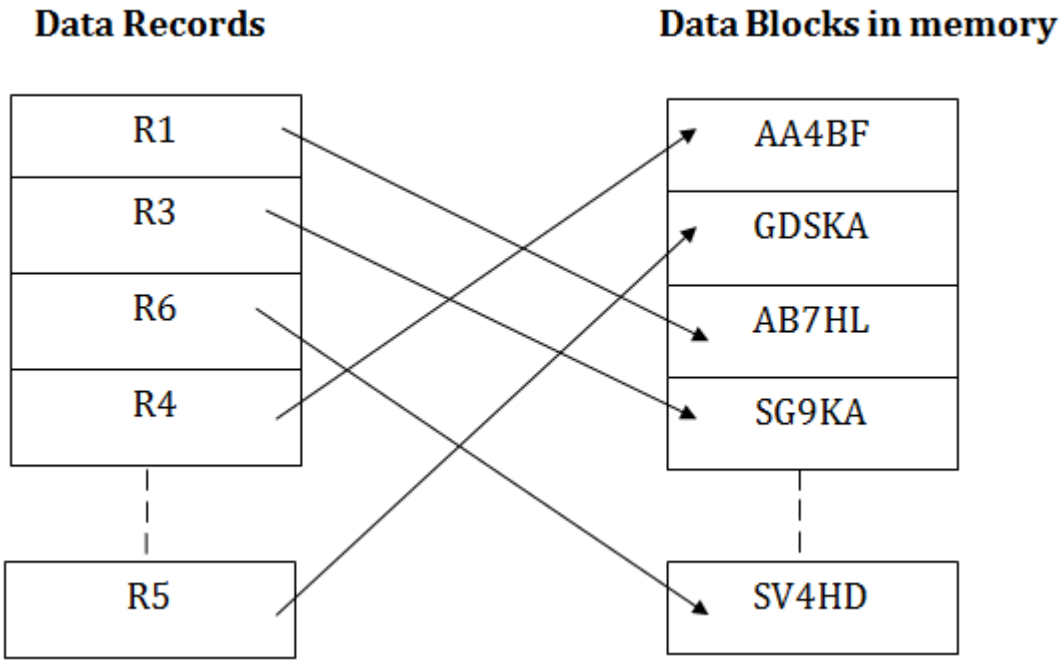
- It is a very good method of file organization for bulk insertion. If there is a large number of data which needs to load into the database at a time, then this method is best suited.
- In case of a small database, fetching and retrieving of records is faster than the sequential record.

Cons of Heap file organization

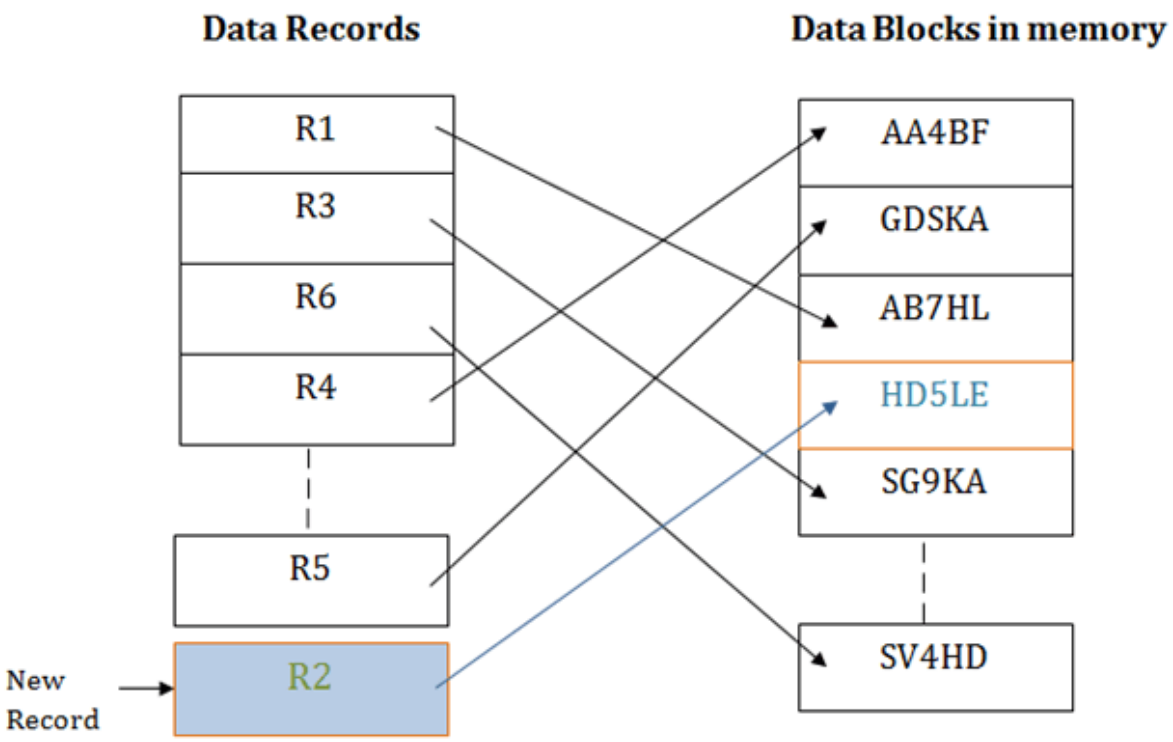
- This method is inefficient for the large database because it takes time to search or modify the record.
-
- This method is inefficient for large databases.

Hash File Organization

- Hash File Organization uses the computation of hash function on some fields of the records. The hash function's output determines the location of disk block where the records are to be placed.

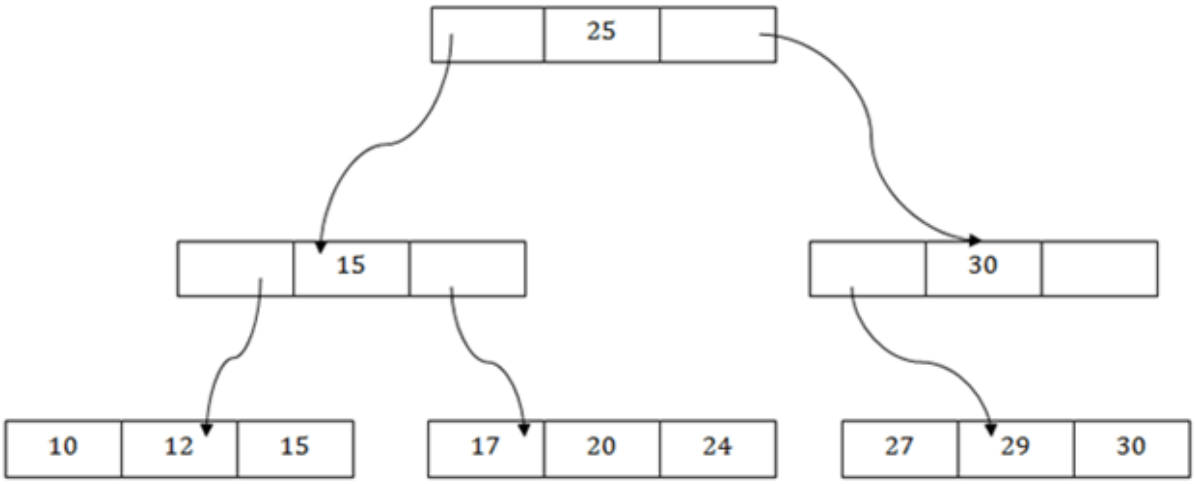


- When a record has to be received using the hash key columns, then the address is generated, and the whole record is retrieved using that address. In the same way, when a new record has to be inserted, then the address is generated using the hash key and record is directly inserted. The same process is applied in the case of delete and update.
- In this method, there is no effort for searching and sorting the entire file. In this method, each record will be stored randomly in the memory.



B+ File Organization

- B+ tree file organization is the advanced method of an indexed sequential access method. It uses a tree-like structure to store records in File.
- It uses the same concept of key-index where the primary key is used to sort the records. For each primary key, the value of the index is generated and mapped with the record.
- The B+ tree is similar to a binary search tree (BST), but it can have more than two children. In this method, all the records are stored only at the leaf node. Intermediate nodes act as a pointer to the leaf nodes. They do not contain any records.



The above B+ tree shows that:

- There is one root node of the tree, i.e., 25.
- There is an intermediary layer with nodes. They do not store the actual record. They have only pointers to the leaf node.
- The nodes to the left of the root node contain the prior value of the root and nodes to the right contain next value of the root, i.e., 15 and 30 respectively.
- There is only one leaf node which has only values, i.e., 10, 12, 17, 20, 24, 27 and 29.
- Searching for any record is easier as all the leaf nodes are balanced.
- In this method, searching any record can be traversed through the single path and accessed easily.

Pros of B+ tree file organization

- In this method, searching becomes very easy as all the records are stored only in the leaf nodes and sorted the sequential linked list.
- Traversing through the tree structure is easier and faster.
- The size of the B+ tree has no restrictions, so the number of records can increase or decrease and the B+ tree structure can also grow or shrink.
- It is a balanced tree structure, and any insert/update/delete does not affect the performance of tree.

Cons of B+ tree file organization

- This method is inefficient for the static method.

Cluster file organization

- When the two or more records are stored in the same file, it is known as clusters. These files will have two or more tables in the same data block, and key attributes which are used to map these tables together are stored only once.
- This method reduces the cost of searching for various records in different files.
- The cluster file organization is used when there is a frequent need for joining the tables with the same condition. These joins will give only a few records from both tables. In the given example, we are retrieving the record for only particular departments. This method can't be used to retrieve the record for the entire department.

EMPLOYEE

EMP_ID	EMP_NAME	ADDRESS	DEP_ID
1	John	Delhi	14
2	Robert	Gujarat	12
3	David	Mumbai	15
4	Amelia	Meerut	11
5	Kristen	Noida	14
6	Jackson	Delhi	13
7	Amy	Bihar	10
8	Sonoo	UP	12

DEPARTMENT

DEP_ID	DEP_NAME
10	Math
11	English
12	Java
13	Physics
14	Civil
15	Chemistry

Cluster Key

DEP_ID	DEP_NAME	EMP_ID	EMP_NAME	ADDRESS
10	Math	7	Amy	Bihar
11	English	4	Amelia	Meerut
12	Java	2	Robert	Gujarat
12		8	Sonoo	UP
13	Physics	6	Jackson	Delhi
14	Civil	1	John	Delhi
14		5	Kristen	Noida
15	Chemistry	3	David	Mumbai

In this method, we can directly insert, update or delete any record. Data is sorted based on the key with which searching is done. Cluster key is a type of key with which joining of the table is performed.

Types of Cluster file organization:

Cluster file organization is of two types:

1. Indexed Clusters:

In indexed cluster, records are grouped based on the cluster key and stored together. The above EMPLOYEE and DEPARTMENT relationship is an example of an indexed cluster. Here, all the records are grouped based on the cluster key- DEP_ID and all the records are grouped.

2. Hash Clusters:

It is similar to the indexed cluster. In hash cluster, instead of storing the records based on the cluster key, we generate the value of the hash key for the cluster key and store the records with the same hash key value.

Pros of Cluster file organization

- The cluster file organization is used when there is a frequent request for joining the tables with same joining condition.
- It provides the efficient result when there is a 1:M mapping between the tables.

Cons of Cluster file organization

- This method has the low performance for the very large database.
- If there is any change in joining condition, then this method cannot use. If we change the condition of joining then traversing the file takes a lot of time.
- This method is not suitable for a table with a 1:1 condition.

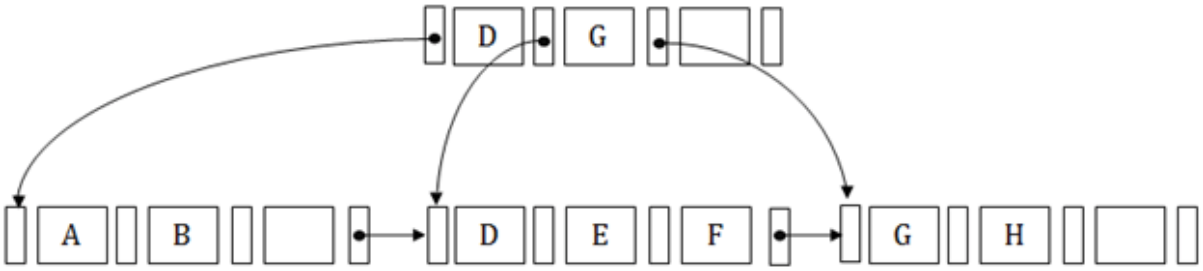
B+ Tree

- The B+ tree is a balanced binary search tree. It follows a multi-level index format.
- In the B+ tree, leaf nodes denote actual data pointers. B+ tree ensures that all leaf nodes remain at the same height.

- In the B+ tree, the leaf nodes are linked using a link list. Therefore, a B+ tree can support random access as well as sequential access.

Structure of B+ Tree

- In the B+ tree, every leaf node is at equal distance from the root node. The B+ tree is of the order n where n is fixed for every B+ tree.
- It contains an internal node and leaf node.



Internal node

- An internal node of the B+ tree can contain at least n/2 record pointers except the root node.
- At most, an internal node of the tree contains n pointers.

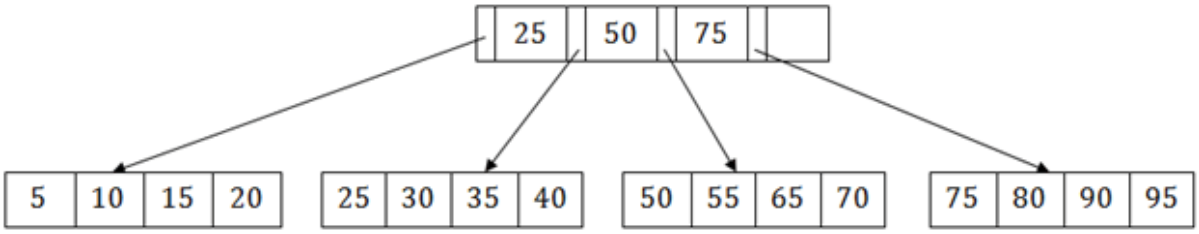
Leaf node

- The leaf node of the B+ tree can contain at least n/2 record pointers and n/2 key values.
- At most, a leaf node contains n record pointer and n key values.
- Every leaf node of the B+ tree contains one block pointer P to point to next leaf node.

Searching a record in B+ Tree

Suppose we have to search 55 in the below B+ tree structure. First, we will fetch for the intermediary node which will direct to the leaf node that can contain a record for 55.

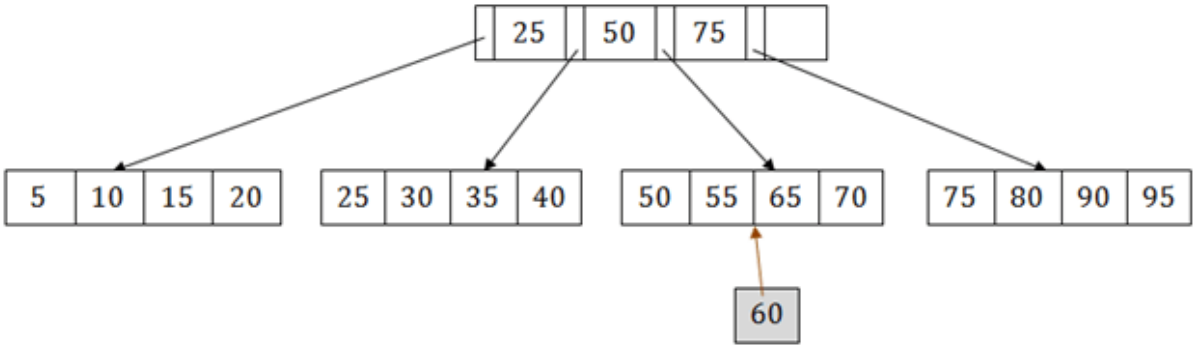
So, in the intermediary node, we will find a branch between 50 and 75 nodes. Then at the end, we will be redirected to the third leaf node. Here DBMS will perform a sequential search to find 55.



B+ Tree Insertion

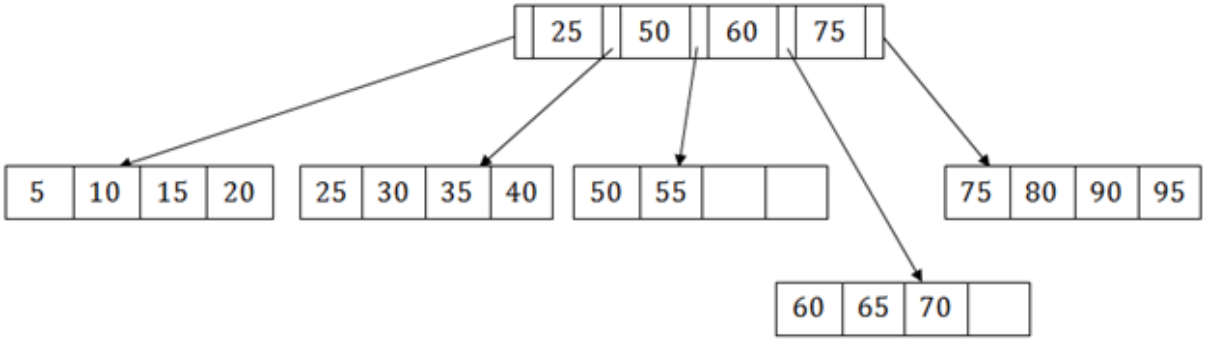
Suppose we want to insert a record 60 in the below structure. It will go to the 3rd leaf node after 55. It is a balanced tree, and a leaf node of this tree is already full, so we cannot insert 60 there.

In this case, we have to split the leaf node, so that it can be inserted into tree without affecting the fill factor, balance and order.



The 3rd leaf node has the values (50, 55, 60, 65, 70) and its current root node is 50. We will split the leaf node of the tree in the middle so that its balance is not altered. So we can group (50, 55) and (60, 65, 70) into 2 leaf nodes.

If these two has to be leaf nodes, the intermediate node cannot branch from 50. It should have 60 added to it, and then we can have pointers to a new leaf node.

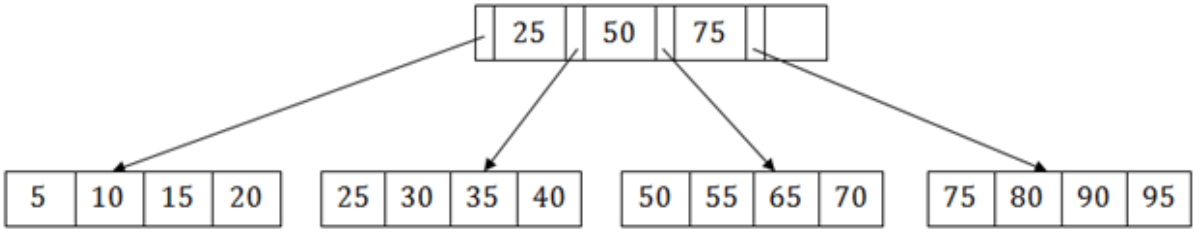


This is how we can insert an entry when there is overflow. In a normal scenario, it is very easy to find the node where it fits and then place it in that leaf node.

B+ Tree Deletion

Suppose we want to delete 60 from the above example. In this case, we have to remove 60 from the intermediate node as well as from the 4th leaf node too. If we remove it from the intermediate node, then the tree will not satisfy the rule of the B+ tree. So we need to modify it to have a balanced tree.

After deleting node 60 from above B+ tree and re-arranging the nodes, it will show as follows:

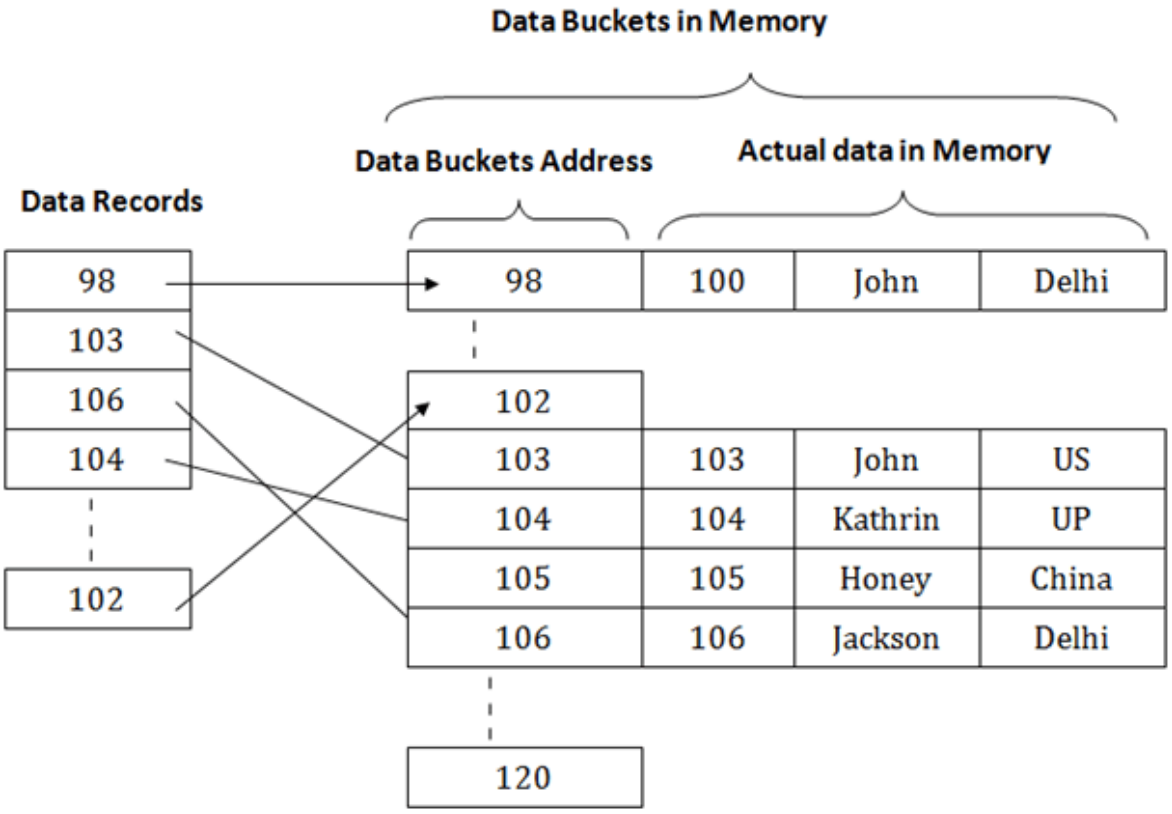


Hashing in DBMS

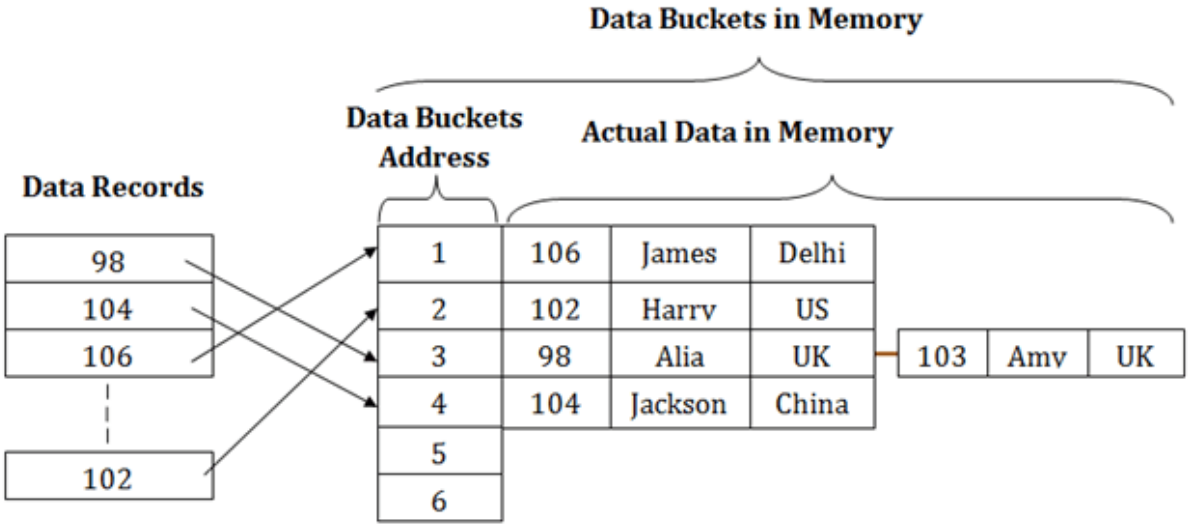
In a huge database structure, it is very inefficient to search all the index values and reach the desired data. Hashing technique is used to calculate the direct location of a data record on the disk without using index structure.

In this technique, data is stored at the data blocks whose address is generated by using the hashing function. The memory location where these records are stored is known as data bucket or data blocks.

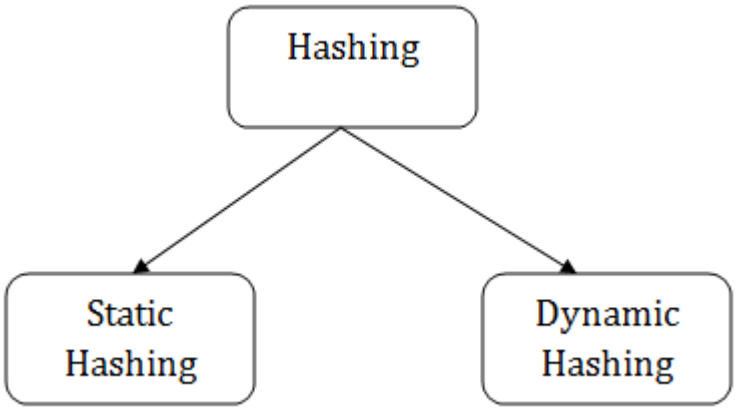
In this, a hash function can choose any of the column value to generate the address. Most of the time, the hash function uses the primary key to generate the address of the data block. A hash function is a simple mathematical function to any complex mathematical function. We can even consider the primary key itself as the address of the data block. That means each row whose address will be the same as a primary key stored in the data block.



The above diagram shows data block addresses same as primary key value. This hash function can also be a simple mathematical function like exponential, mod, cos, sin, etc. Suppose we have mod (5) hash function to determine the address of the data block. In this case, it applies mod (5) hash function on the primary keys and generates 3, 3, 1, 4 and 2 respectively, and records are stored in those data block addresses.



Types of Hashing:



- Static Hashing
- Dynamic Hashing

Static Hashing

In static hashing, the resultant data bucket address will always be the same. That means if we generate an address for EMP_ID =103 using the hash function mod (5) then it will always result in same bucket address 3. Here, there will be no change in the bucket address.

Operations of Static Hashing

- **Searching a record**

When a record needs to be searched, then the same hash function retrieves the address of the bucket where the data is stored.

- **Insert a Record**

When a new record is inserted into the table, then we will generate an address for a new record based on the hash key and record is stored in that location.

- **Delete a Record**

To delete a record, we will first fetch the record which is supposed to be deleted. Then we will delete the records for that address in memory.

- **Update a Record**

To update a record, we will first search it using a hash function, and then the data record is updated.

If we want to insert some new record into the file but the address of a data bucket generated by the hash function is not empty, or data already exists in that address. This situation in the static hashing is known as **bucket overflow**. This is a critical situation in this method.

To overcome this situation, there are various methods. Some commonly used methods are as follows:

Dynamic Hashing

- The dynamic hashing method is used to overcome the problems of static hashing like bucket overflow.
- In this method, data buckets grow or shrink as the records increases or decreases. This method is also known as Extendable hashing method.
- This method makes hashing dynamic, i.e., it allows insertion or deletion without resulting in poor performance.

Advantages of dynamic hashing

- In this method, the performance does not decrease as the data grows in the system. It simply increases the size of memory to accommodate the data.
- In this method, memory is well utilized as it grows and shrinks with the data. There will not be any unused memory lying.
- This method is good for the dynamic database where data grows and shrinks frequently.

Disadvantages of dynamic hashing

- In this method, if the data size increases then the bucket size is also increased. These addresses of data will be maintained in the bucket address table. This is because the data address will keep changing as buckets grow and shrink. If there is a huge increase in data, maintaining the bucket address table becomes tedious.
- In this case, the bucket overflow situation will also occur. But it might take little time to reach this situation than static hashing.

RAID (Redundant Array of Independent Disk)

RAID refers to redundancy array of the independent disk. It is a technology which is used to connect multiple secondary storage devices for increased performance, data redundancy or both. It gives you the ability to survive one or more drive failure depending upon the RAID level used.

It consists of an array of disks in which multiple disks are connected to achieve different goals.

RELATIONAL DATA MODEL AND LANGUAGE

(UNIT-4)

Relational Model was proposed by E.F. Codd to Model data in the form of relations or tables. After designing the Conceptual model of Database using ER diagram, we need to convert the conceptual model in the relational model which can be implemented using any RDMS languages like Oracle SQL, MySQL etc.

WHAT IS RELATIONAL MODEL? (CONCEPT OF RELATIONAL MODEL)

Relational Model represents how data is stored in Relational Databases. A Relational Database stores data in the form of tables (relations).

Consider a relation STUDENT with attributes ROLL-NO, NAME, ADDRESS, PHONE.

(Tablename) (Attributes) (no. of Attributes = 4) (Degree).

STUDENT

ROLL-NO	NAME	ADDRESS	PHONE
1	ANIL	DELHI	9152780120
2	ROSHNI	MEERUT	9625981012
3	RAJESH	GHAZIABAD	8492723058
4	RIYA	DELHI	9182675081

Record or Tuple [no. of Tuples (Cardinality)

ATTRIBUTE : Attributes are the properties that defines a relation. e.g. ROLL-NO, NAME etc.

TUPLE : Each row in a relation is known as tuple. The above table have 4 tuples.

RELATION SCHEMA: A relation Schema represents name of the Relation with its attributes.
e.g. STUDENT(ROLL-NO, NAME, ADDRESS, PHONE) is relation Schema for STUDENT. If a Schema has more than 1 relation, it is called Relational Schema.

RELATION INSTANCE: The set of tuples of a relation at a particular instance of time is called as relation Instance. Above Table shows the relation Instance of student at a particular time. It can change whenever there is insertion, deletion or updation in the database.

Degree : The number of attributes in the relation is known as degree of the relation. The STUDENT relation defined above has degree 4.

Cardinality : The number of tuples in a relation is known as cardinality. The STUDENT relation defined above has 4.

Column : Column represent the set of values [illegible]. The column ROLL NO

RELATIONAL MODEL CONSTRAINTS (3)

Relational Model constraints are restrictions specified to the data values in the relational database.

There are various types of constraints follows as under -:

1) INHERENT MODEL BASED CONSTRAINTS - The Constraints that are implicit in a data mode are Inherent model based constraints.

for example:- A Relation in a Database must not have duplicate tuples, there is no constraints in the ordering of the tuples and attributes.

2) APPLICATION BASED CONSTRAINTS -

The Constraints that cannot be applied whil defining the Database schema are expressed i application program.

for example:- The salary of an Employee, cannot be more than his supervisor.

3) INTEGRITY

8: INTEGRITY CONSTRAINTS (4)

OR (SCHEMA BASED CONSTRAINT)

Integrity constraints are pre-defined set of rules that are applied on the table fields (Columns) or relations to ensure that the overall validity, Integrity and consistency of the Data present in the database table is maintained. Evaluation of all the conditions or rules mentioned in the Integrity constraint is done every time a table insert, update, delete or alter operation is performed. The data can be inserted, updated, deleted or altered only if the result of the constraint comes out to be true. Thus Integrity constraints are useful in preventing any accidental damage to the database by an authorized user.

TYPES Of INTEGRITY CONSTRAINTS

- Domain constraint.
- Entity Integrity Constraint.
- Referential Integrity Constraint.
- Key Constraint.

DOMAIN CONSTRAINT

Domain Constraint contains a certain set of rules or condition to restrict the kind of attributes or values a column can hold in the Database table.

The data type of a Domain can be string, integer, character, etc.

EXAMPLE

Consider a student's table having Roll No, Name, class.

Roll No	Name	class
201	Anil	6
202	Radha	8
203	John	12
204	Riya	A

In this student table, the value A in the last row, last column violates the Domain integrity constraint because the class attribute contains only integer value while A is a character.

(2) ENTITY INTEGRITY CONSTRAINT

Entity Integrity constraint is used to ensure that the primary key cannot be null. A primary key is used to identify individual record in a table, and if the primary key has a null value, then we can't identify these records. There can be null values anywhere in the table except the primary key column.

EXAMPLE

table having Id, Name and

ID	Name	Salary
2101	Anil	40000
2102	Ramesh	50000
2103	Sunita	60000
	Kamla	12000

On this Employee, table we can see that ID column is a primary key and contains a null value in the last ~~box~~ row, which violates the entity integrity constraint.

3) REFERENTIAL INTEGRITY CONSTRAINT

Referential Integrity Constraint ensures that there must always exist a valid relationship between two relational database tables. This valid relationship between the two tables confirms that a foreign key exist in a table. It should always reference a corresponding value or attribute in the other table or be null.

Example:

Consider an Employee and a Department table where Dept-ID acts as a foreign key between the two tables.

EMPLOYEES TABLE

ID	Name	Salary	Dept-ID
1201	Tiya	40000	3
1202	Subhash	80000	2
1203	Renuka	900000	4
1204	Sarika	100000	3
1205	Aman	36000	1

DEPARTMENT TABLE

Dept-ID	Dept-Name
1	Sales
2	HR
3	Technical

In the Above example, Dept-ID act as a foreign key in the Employees table and a primary key in the department table.
Row having Dept ID = 4 violates the referential Integrity constraint since Dept ID 4 is not defined as a primary key column in the Departments table.

1. KEY CONSTRAINT (8)

Keys are the set of entities that are used to identify an entity within its entity set uniquely. There could be multiple keys in a single entity set, but out of these multiple keys, only one key will be the primary key. A primary key can only contain unique and not null values in the relational Database table.

EXAMPLE

Consider a student's table

Roll NO	Name	class
101	Anil	6
102	Saurabh	8
103	Monu	12
104	Kamlesh	4
102	Sonali	2

The last row of the student's table, violates the Key Integrity constraint since ROLL NO- 102, is repeated twice in the primary Key column.

A primary key must be unique and not null therefore duplicate values are not [illegible] the ROLL NO. Column of the

SQL (Structured Query Language): (9)

- SQL (Structured Query Language) is used for accessing, manipulating and communicating with the database. Almost every function such as retrieving data from the database, creating a new database, manipulating data and databases such as insertion, deletion and updation can be performed using SQL. It is a user-friendly and domain specific language.

NEED Of SQL -:

1) It is widely used in the Business Intelligence tool.
2) Data Manipulation and data testing are done through SQL.
3) Data science tools depends highly on SQL (Structured Query language). Big data tools such as Spark, Impala are dependent on SQL.
- It is one of the Demanding Industrial Skills.

ADVANTAGES Of SQL (Structured Query language)

SQL has many advantages which makes it popular and highly Demanded. It is a reliable and efficient language used for communicating with the database.

Some advantages of SQL are as follows -:

4. Faster Query Processing - Large amount of data is retrieved quickly and efficiently. Operations like insertion, deletion, manipulation of data is also done in almost no time.

5. No Coding Skills - For data retrieval, large number of lines of code is not required. All basic keywords such as SELECT, INSERT INTO, UPDATE, etc are used and also the syntactical rules are not complex in SQL, which makes it a user-friendly language.

6. Standardized language - Due to Documentation and long establishment (over years), it provides a uniform platform worldwide to all its users.

7. Portable - It can be used in programs in PC's, Server, laptop independent of any platform (operating System, etc). Also it can be embedded with other Applications as per need/requirement/use.

8. Interactive language - Easy to learn and understand, answers to complex queries can be received in seconds.

9. Multiple Data views.

DISADVANTAGE Of SQL (Structured Query language)

Various Disadvantages of SQL are as follows:

1. Complex Interface - SQL has a difficult Interface that makes few users uncomfortable while dealing with the database.

2. Cost - Some versions are costly and hence programmers cannot access it.

3. Partial Control - Due to hidden business rules

• APPLICATIONS Of SQL

1) SQL is used by developers and DBAs (Database Administrators) in writing Data Integration Scripts.
2) It is used to deal with analytical queries to analyze the data and get insights from it.
3) Retrieving Information.
4) Modification / Manipulation of Data and Database table such as Insertion, Deletion and updation.

UNIT - 2

ENTITY - RELATIONSHIP (ER) MODEL

- ER Model is a high-level Data model Diagram.
- ER Model describes the structure of a database with the help of a diagram, which is known as ER (Entity Relationship) Diagram.
- An ER Model is a Design or blueprint of a database that can be later Implemented as a database.
- It is based on the notion of the real world entities & relationship among them.

ER Diagram has the following three Components:-

- Entities - Entity is a real-world thing or object. It can be a person, place or even a concept.

Example - Teacher, Students, Course, Building, Department etc are some of the entities of a School Management System.

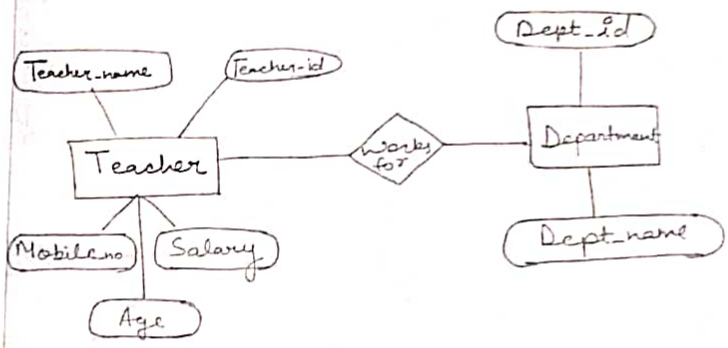
- Attributes - An entity contain real-world property called attributes. This is the characteristics of that attribute.

Example:- The entity teacher has the property like teacher_id, name, salary, age etc.

- Relationships - Relationship tells how two attributes are related.

Example :- Teacher work for a Department.

Example - ER Model



- The entities are Teacher & Department
- The attribute of Teacher entity are Teacher_name, Teacher_id, Age, salary, Mobile_no.
- The attribute of Department entity are Dept_id, Dept_name.
- The two entities are connected to using the relationships. Here each Teacher works for a Department.

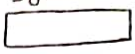
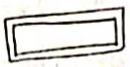
E-R (Entity-Relationship) Diagram

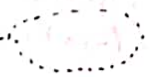
(Proposed by Peter Chen in 1971)

Entity Relationship diagram displays the entity sets stored in a database. In other words, ER diagram helps to explain the logical structure of a databases. An ER diagram looks very similar to the flowchart.

ER Diagram includes many specialized symbols, and its meaning make it unique. The purpose of ER Diagram is to represent the entity framework infrastructure.

Entity Relationship Diagram Symbols & Notations.

	Symbol	Meaning
1.		(This Rectangle represent the Entity)
2.		(This represents the weak entity)
3.		(This Diamond Symbol represents the Relationship)

	Symbol	Meaning
4.		(This Represents the Identifying Relationship)
5.		(This Oval/Ellipses Shaped Represent the Attributes)
6.		(It Represents the Key-attributes)
7.		(The Double Ellipses Represents the Multivalued attributes)
8.		(It Represents the Composite Attribute)
9.		(It Represents the Derived attribute)

Components Of ER-Diagram (48)

This model is based on three concept:-

1. Entities
2. Attributes
3. Relationships.

1) **Entity** :- Real world thing either living or non-living that is easily recognizable and non-recognizable. An entity can be a place, person, object, event or a Concept which stores data in the database. The characteristics of entities are must have an attribute or a unique key. Every Entity is made of some 'attributes', which Represent that entity.

Example of Entities :-

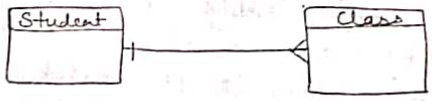
- Person :- Employee, student, Patient etc
- Place :- Stores, Buildings etc.
- Object :- Machines, products & Cars etc.
- Event :- Sale, Registration, Renewal etc.
- Concept :- Account, Course etc.

2) **Entity Set** :-

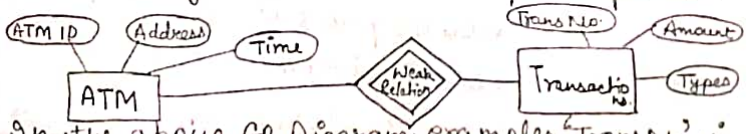
An entity set is a group of similar kind of entities. It may contain entities with attribute showing similar values. Entities are represent by their properties, which, also called attributes. All Attributes have their seperate values.

for example :- a student entity may have a name, age, class as their attributes.

3) Roles



4) Weak Entities - Entity types that do not have key attributes of their own are called weak entity types. It can be identified uniquely by considering the primary key of another entity. For that, weak entity sets need to have participation.



In the above ER Diagram examples, 'Trans No' is a Discriminator within a group of transaction in an ATM.

Strong Entity Set

- Strong Entity set always has a primary key.
- It is represented by Rectangle Symbol.
- It contains a primary key represented by underline Symbol.
- The member of strong entity set is called as dominant entity set.
- In the ER-Diagram the relationship b/w two strong entity shown by Diamond Symbol.

Weak Entity Set

- It doesnot have Enough Attributes to build a primary key.
- It is Represented by Double Rectangle Symbol.
- It contains a Partial Key which is Represented by a Dashed underline Symbol.
- The member of a weak entity set called as a Subordinate entity set.
- The Relationship b/w one strong of a weak entity set shown by using the Double Diamond Symbol.

Strong Entity

- The connecting line of the strong entity set with the relationship is single.
- Primary Key is one of its attributes which helps to identify its member.

Weak Entity

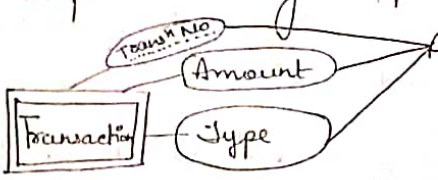
- The line connecting the weak entity set for identifying relationship is Double
- In a weak entity set it is a combination of primary key & partial key of the strong entity set.

2. Attributes -

It is a single-valued property of either an entity-type or a relationship-type.

for example :- A lecture might have an attributes : Time,
Date,
duration
& place etc.

An Attribute in an ER-Diagram examples is represented by Ellipse or Oval Shaped.



TYPES OF ATTRIBUTES -:

1. Single-valued Attribute.
2. Composite Attribute.
3. Derived Attribute.
4. Multivalued Attribute.
5. Simple Attribute.

- Single-Valued Attribute - Most Attributes have a single value for a particular entity, Such Attributes are called single-valued Attribute. for example :- Age is a single-valued Attribute.

- Composite Attribute - Composite Attributes is an attribute where the value of that attribute can be further subdivided into meaningful sub-parts.
for example :- Student's ~~name~~ full name may be further divided into first name, second name and last name.

- Derived Attribute - A Derived Attribute is an Attribute that can be derived from other attributes of the entity types.
eg :- Age (can be Derived from DOB)
The age of the student can be derived or calculated from Date of the Birth-o

It is Represented by

⑥ CARDINALITY

Cardinality Defines the numerical Attributes of the relationship between two entities or entity set.

- Different types of cardinal relationship are -:

① One-to-one Relationship.
② One-to-many Relationship.
③ Many to one Relationship.
④ Many to Many Relationship.

① One-to-one -
One entity from entity X can be associated with at most one entity of entity set Y and vice-a-versa.

Example:- One student can Register for Numerous Courses. However all those Courses have a single line back to that one Student.

② One-to-many - One entity from entity set X can be associated with Multiple entities of entity set Y, but an entity from entity set Y can be associated with atleast one entity.

for example:- One class is consisting of multiple students.

③ Many to One - More than One entity set X can be associated with at most one entity of entity set Y. However, an entity from entity set Y may or may not be associated with more than one entity from entity set X.

for example:- Many students belong to the same class.

4. Many-to-Many- One Entity from X can be associated with more than one entity from Y and vice-a-versa.

for example - Students as a group are associated, with multiple faculty members, and faculty members can be associated with multiple students.

HOW TO CREATE ER DIAGRAM

Step 1. Entity Identification

Step 2. Relationship Identification.

Step 3. Cardinality Identification

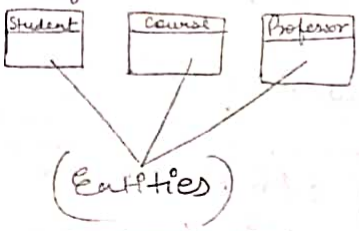
Step 4. Identify Attributes

Step 5. Create ERD (Entity - Relationship Diagram).

1. Entity Identification -

We have three entities -:

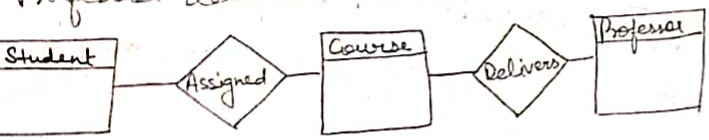
1. Student
2. Course
3. Professor



2. Relationship Identification

We have the following two relationships

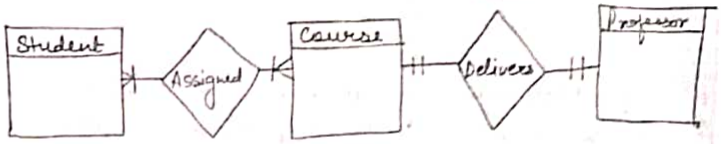
- The student is Assigned a Course
- Professor delivers a course.



3. Cardinality Identification -:

From them problem statement we know that -:

- A student can be assigned Multiple Courses.
- A professor can deliver only one course.

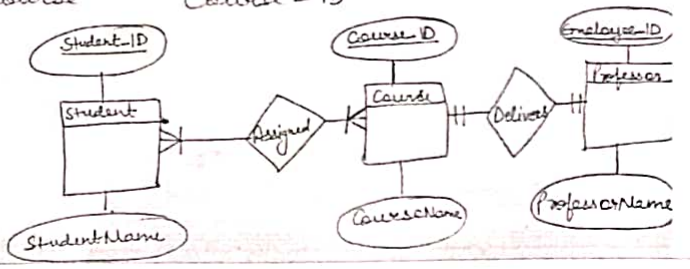


4. Identify Attributes -:

Once we have a list of Attributes, we need to map them to the identified entities.

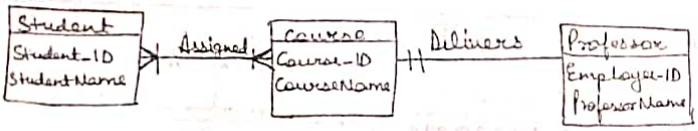
Ensure an Attribute is to be paired with exactly one entity. Once the mapping is done, identify the primary keys. If a unique key is not readily available, create one.

Entity	Primary Key	Attribute
Student	Student_ID	StudentName
Professor	Employee_ID	ProfessorName
Course	Course_ID	CourseName



5. Create the ERD (Entity-Relationship) Diagram.

A More Modern representation of Entity - Relationship Diagram Example



KEYS IN DBMS-

- A Key refers to an attribute/a set of attributes that help us identify a row (or tuple) uniquely in a table. (Relation).

A Key is also used when we want to establish relationships between the different columns & tables of a relational database. The individual values present in a key are commonly referred to as Key values.

• Why do we require a key in DBMS

We use a key for defining various types of Integrity constraints in a database. A Table on the other hand, represents a collection of the records of various events for any relation. Now there might be thousand of these records and some of these might be duplicated. Thus, we need a way in which one can identify all of these records uniquely and seperately, i.e. without any duplicates. This hassle is removed with the help of Keys.

1. PRIMARY KEY - Primary Keys in DBMS is a column or group of column in a table that uniquely identify every row in that table. The primary key can't be a duplicate meaning, the same value can't be appear more than once in the table. A table cannot have more than one primary key.

Rules for Defining Primary Key:

- Two rows can't have the same primary key value.
- It must for every row to have a primary key value.
- The primary key field cannot be null.
- The value in a primary key column can never be modified or updated if any foreign key refers to that primary key.

Example

Employee ID	First Name	Last Name
11	Anil	Sharma
20	Sita	Verma
32	Mukesh	Singh

In this example, Employee ID is a primary key because it uniquely identifies an employee record. On this table, no other employee can have the same employee ID.

2. CANDIDATE KEY - Candidate Key in SQL is a set of attributes that uniquely identifies tuple in a table. Candidate Key is a Super Key with no repeated Attributes. The primary Key should be selected from the Candidate Keys. Every table must have atleast a Single Candidate Key. A table can have multiple candidate key, but only a single primary key.

Properties of a Candidate Key -:

- It must contain unique values.
- A Candidate Key in SQL may have multiple attributes.
- Must not contain null values.
- It should contain minimum fields to ensure uniqueness.
- Uniquely identify each record in a table.

for example :-

StudID	ROLLNO	FIRSTNAME	LASTNAME	E-MAIL
1	10	RADHA	SINGH	abc@gmail.com
2	11	AMAN	VERMA	xyz@gmail.com
3	12	ASHU	JAIN	uvz@yahoo.com

In the given table StudID, ROLL NO, and email are candidate keys which helps us to uniquely identify the student record in the table.

---

3. FOREIGN KEY - 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 b/w two different instances of an entity. It act as a cross-reference between two tables as it references the primary key of another table.

EXAMPLE OF FOREIGN KEY =>

Dept Code	Dept Name
001	Computer
002	Managemant
003	Englich

TeacherID	firstname	Lastname
C0028	Sarita	Sharma
C0012	Renu	Verma
C1102	Sunita	Jain

In above, we have two examples tables, teacher and department in a School, However, there is no way to see which teacher work in which Department.

In this table by adding the foreign key, in Deptcode to the Teacher name, we can create a relationship between the two tables.

TeacherID	Dept Code	firstname	Lastname
C0028	001	Sarita	Sharma
C0012	002	Renu	Verma
C1102	003	Sunita	Jain

4. SUPER KEY - A Superkey is a group of Single or Multiple Keys which Identifies rows in a table. A Super Key may have additional attributes that are not needed for unique Identification.

EXAMPLE -

EmpSSN	EmpNum	Emp NAME
740028772	C208	Rakesh
900821100	C218	John
821200187	C202	Amrish

In the above given example, EmpSSN and EmpNum names are Super keys.

5. ALTERNATE KEY - Alternate Keys is a column or group of columns in a table, that uniquely identify every row in that table. A table can have multiple choices for a primary key but only one can be set as the primary key. All of the keys which are not primary key are called an Alternate key.

EXAMPLE OF ALTERNATE KEY -

Stud ID	Roll No	FirstName	LastName	Email
1	10	Reena	Sharma	ab@gmail.com
2	11	Bina	Singh	xyz@gmail.com
3	12	Atul	Shah Raj	tom@yahoo.com

In this table, Stud ID, Roll No, Email are qualified to become a primary key. But Since Stud ID is the primary key, Roll No, Email becomes the Alternate key.

5. COMPOSITE KEY – Composite key is a combination of two or more columns that uniquely identify rows in a table. The combination of columns guarantees uniqueness, though individually uniqueness is not guaranteed. Hence, they are combined to uniquely identify Records in a table.

7. SURROGATE KEY - Surrogate Keys is an artificial key which aims to uniquely identify each record. This kind of partial key in DBMS is unique because it is created when you don't have any natural primary key. They donot lend any meaning to the data in the table.

Surrogate Keys is a value generated right before the record is inserted into a table.

Firstname	Lastname	Start time	End time
Khushbu	Singh	09:00	18:00
Komal	Sharma	08:00	19:00
Rati	Agnihotri	11:00	20:00
Tiya	Kalyan	14:00	23:00

Above example - It shows the shift timings of the different Employee. In this example a Surrogate Key is needed to uniquely identify each employee

SURROGATE KEYS IN SQL are allowed when

(a) No property has the parameter of the primary key.

(b) In the table when the primary key is too big or Complicated.

Q.20. Define specialisation with a suitable example. *(2013)*

Ans. Specialisation : Specialisation is the process of designating subgroupings within an entity set. It is the abstracting process of introducing new characteristics to an existing class of objects to create one or more new classes of objects. It is a top-down process. Specialisation involves taking a higher level entity and using additional characteristics, generating lower-level entities. Specialisation can be taken as an opposite (or reverse) process of generalisation. An entity set may be specialised by more that one distinguisting feature.

For example : The specialisation of person allows us to distinguish among persons, according to whether they are employee or customers.

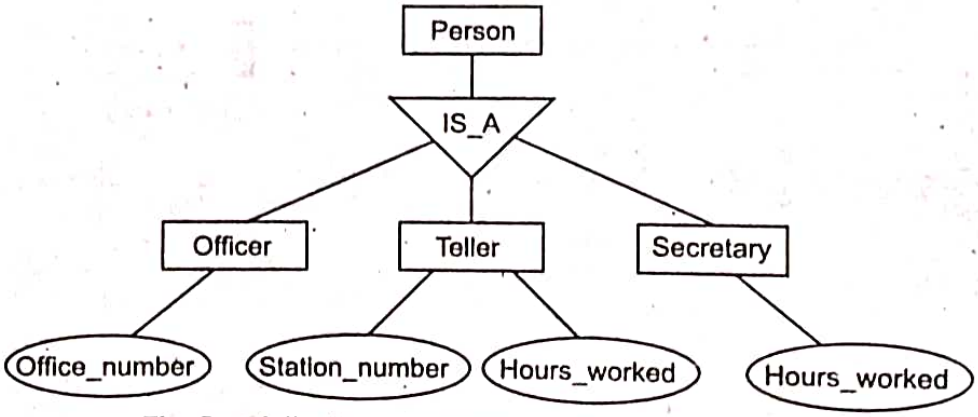


Fig. Specialisation of person into various employees.

Q.11. Explain generalisation. *(2013)*

Ans. Generalisation : Generalisation is defined as the process of identifying some common characteristics of a collection of entity sets and creating a new entity set that contains entities processing these common features.

In other words, as the process of minimising the differences between entities by identifying their common characteristics. It is a bottom-up approach. Generalisation is an IS-A relationship as shown in Fig. (a).

For example : Customer and Employee are lower level entities and Person is a higher level entity. Customer and Employee are generalised to form an entity person. Person contains the attributes that are common in both Customer and Employee. The person entity type is the superclass of Customer and Employee subclasses. Generalisation of Customer and Employee can be shown in Fig. (b).

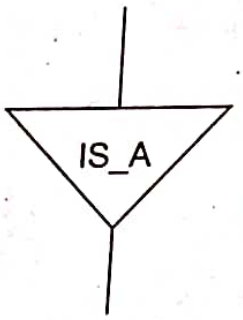


Fig. (a)

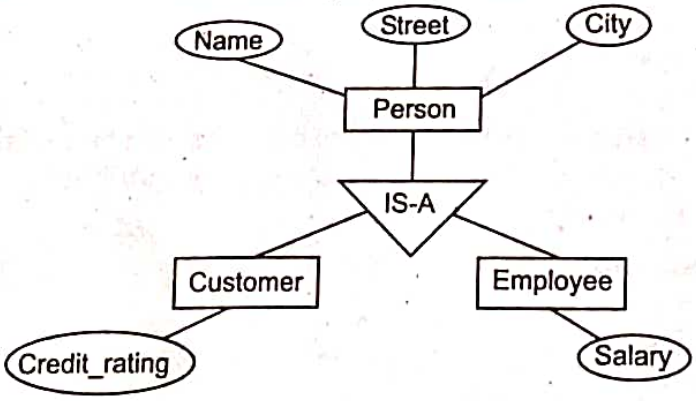


Fig. (b) E-R diagram with generalisation.

UNIT 5

EER and ER to relational mapping:

Entity-Relationship (ER) modelling is a popular approach for designing database schemas. The Enhanced Entity-Relationship (EER) model is an extension of the ER model that includes additional concepts to represent more complex relationships and constraints. Here are the key components and steps to design a database using the EER model:

Components of EER Model:

1. **Entities:**
   - Represent real-world objects.
   - Have attributes that describe properties of the entity.
2. **Relationships:**
   - Define associations between entities.
   - Can have attributes.
3. **Attributes:**
   - Properties or characteristics of entities.
   - Can be simple or composite.
4. **Key Attributes:**
   - Unique attributes used to identify instances of an entity.
5. **Multivalued Attributes:**
   - Attributes that can have multiple values.
6. **Derived Attributes:**
   - Attributes whose values can be derived from other attributes.
7. **Weak Entities:**
   - Entities that do not have a primary key attribute.
   - Existence depends on being related to another entity.
8. **Subtypes and Supertypes:**
   - Used to model generalization/specialization relationships.
   - Supertype represents a generalized entity, and subtypes represent more specific entities.
9. **Inheritance:**
   - Subtypes inherit attributes and relationships from their supertype.

10. **Aggregation:**
    - Represents a higher-level entity composed of other entities.

Steps for EER Database Design:

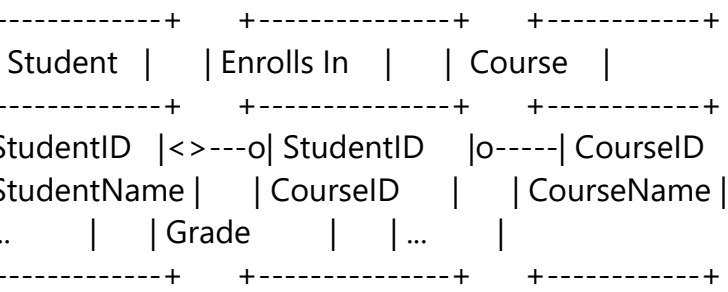
1. **Identify Entities:**
    - Identify the main entities in the system.
2. **Define Relationships:**
    - Establish relationships between entities.
3. **Identify Attributes:**
    - Determine the attributes for each entity.
    - Identify key attributes for each entity.
4. **Refine Relationships:**
    - Refine relationships by adding attributes or additional constraints.
5. **Identify Generalization/Specialization:**
    - Identify if there are generalization/specialization hierarchies among entities.
6. **Identify Aggregations:**
    - Identify if there are aggregations that represent higher-level entities.
7. **Normalize the Schema:**
    - Apply normalization techniques to ensure the database is well-structured and minimizes redundancy.
8. **Validate and Iterate:**
    - Validate the model with stakeholders and iterate as needed.
9. **Implement Constraints:**
    - Implement constraints, such as primary and foreign keys, to maintain data integrity.
10. **Document the Model:**
    - Create documentation that includes entity-relationship diagrams, attributes, relationships, and constraints.

Example:

Let's consider a simple example:

Entities: Student, Course Relationship: Enrolls In Attributes: StudentID, StudentName, CourseID, CourseName, Grade

Here's a basic EER diagram for this example:



In this example, "Enrolls In" is a relationship with an additional attribute (Grade), and it connects the Student and Course entities.

Remember, the complexity of your EER model will depend on the specific requirements of your application or system.

DATA BASE DESIGN

Database design is the process of creating a detailed data model of a database. It involves defining the structure that will organize and store data efficiently, ensuring data integrity, and providing mechanisms for data retrieval and manipulation. Effective database design is crucial for building scalable, maintainable, and high-performance databases. The process typically involves the following key steps:

1. **Requirement Analysis:**
   - Understand the requirements of the system or application that the database will support.
   - Identify the data that needs to be stored and the relationships between different pieces of information.
2. **Conceptual Design:**
   - Create a high-level conceptual model that represents the entities, relationships, and constraints in the system.
   - Use techniques such as Entity-Relationship (ER) modeling to capture the abstract structure of the data.

3. **Normalization:**
    - Apply normalization techniques to eliminate data redundancy and ensure that data is stored efficiently.
    - Normalize the data model to reduce the chances of update anomalies and ensure data consistency.
4. **Logical Design:**
    - Transform the conceptual model into a logical model that can be implemented using a specific database management system (DBMS).
    - Define tables, columns, data types, and relationships in a way that reflects the conceptual model.
5. **Data Integrity:**
    - Enforce data integrity constraints to maintain the accuracy and reliability of the data.
    - Define primary keys, foreign keys, unique constraints, and other rules to prevent inconsistent or invalid data.
6. **Normalization (Again):**
    - Refine the logical design based on normalization principles.
    - Ensure that the design is free from update, insertion, and deletion anomalies.
7. **Physical Design:**
    - Determine how the database will be implemented at the physical level.
    - Consider factors such as storage structures, indexing, and optimization for performance.
    - Choose appropriate indexing strategies to enhance query performance.
8. **Security and Access Control:**
    - Define security measures and access control policies to protect the database from unauthorized access.
    - Implement user roles, permissions, and encryption mechanisms to secure sensitive data.
9. **Transaction Management:**
    - Establish transaction management mechanisms to ensure data consistency and integrity during database operations.
    - Consider features such as transactions, ACID properties, and isolation levels.
10. **Backup and Recovery:**

- Develop backup and recovery strategies to safeguard against data loss and system failures.
- Implement regular backup procedures and define recovery mechanisms.

11. **Documentation:**
    - Create comprehensive documentation that describes the database schema, relationships, constraints, and other relevant information.
    - Document any specific considerations or business rules that apply to the database.
12. **Testing and Optimization:**
    - Test the database design against realistic scenarios to ensure that it meets performance and functionality requirements.
    - Optimize queries and indexing as needed for efficient data retrieval.

Database design is an iterative process, and adjustments may be made as the system evolves or new requirements emerge. It requires collaboration between database designers, developers, and stakeholders to create a database that effectively supports the goals of the organization or application.

ER To Relational Mapping

Entity-Relationship (ER) to Relational mapping is the process of translating an Entity-Relationship diagram, typically represented using the ER model, into a relational schema. The relational schema is then used to create tables in a relational database management system (RDBMS). This mapping is necessary when implementing a database system based on an ER model, as relational databases are the most common type of databases used in practice.

Here are the key steps involved in ER to Relational mapping:

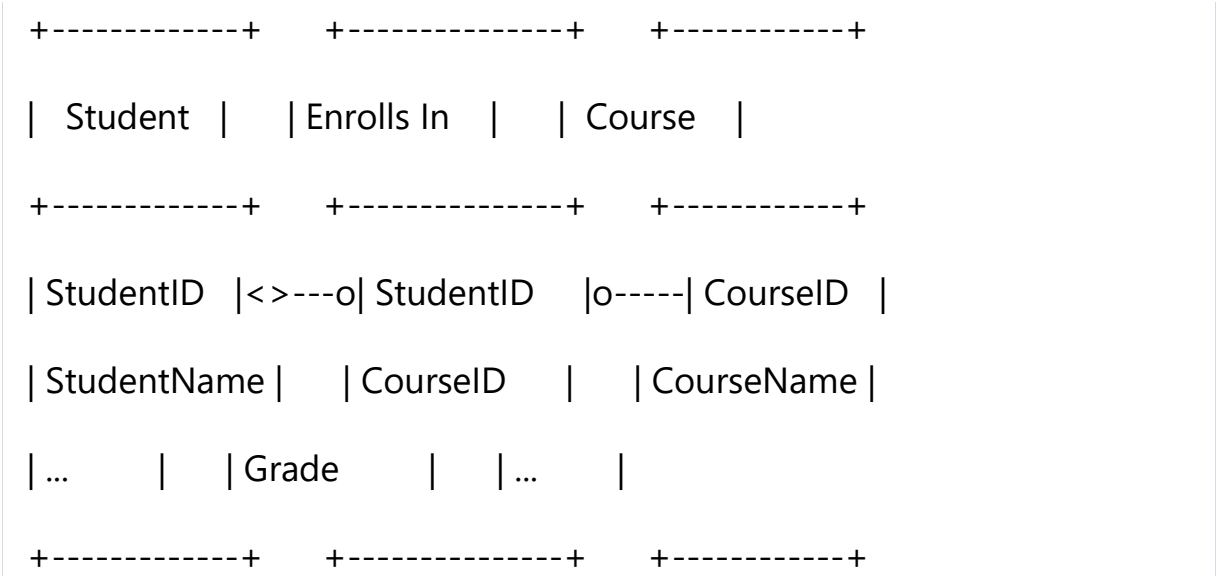
1. **Entities to Tables:**
    - Each entity in the ER diagram becomes a table in the relational database.
    - The attributes of the entity become columns in the table.

- The primary key attribute of the entity becomes the primary key of the table.

2. **Relationships to Tables:**
   - For binary relationships (relationships between two entities), create a table for each relationship.
   - Include foreign keys in these tables to represent the relationships between entities.
   - If the relationship has attributes, include them as columns in the relationship table.
3. **One-to-One Relationships:**
   - If there is a one-to-one relationship, choose one of the entities and include its primary key as a foreign key in the other entity's table.
4. **One-to-Many and Many-to-Many Relationships:**
   - For one-to-many relationships, include the primary key of the "one" side as a foreign key in the table of the "many" side.
   - For many-to-many relationships, create a new table to represent the relationship. Include the primary keys of both entities involved in the relationship as foreign keys in this table.
5. **Attributes:**
   - Map each attribute to a column in the corresponding table.
6. **Composite and Multivalued Attributes:**
   - If an entity has composite attributes, create separate columns for each component.
   - If an entity has multivalued attributes, create a separate table for the multivalued attribute with a foreign key referencing the original entity.
7. **Keys:**
   - Identify and enforce primary keys for each table.
   - Foreign keys should reference the primary keys of related tables.
8. **Normalization:**
   - Apply normalization techniques to ensure the relational schema is free from redundancy and update anomalies.

Example:

Consider the following ER diagram:



The corresponding relational schema would be:

Student Table:

StudentID	StudentName
1	Alice
2	Bob

Course Table:

CourseID	CourseName

| 101 | Math |

| 102 | English |

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

EnrollsIn Table:

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

| StudentID | CourseID | Grade |

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

| 1 | 101 | A |

| 2 | 102 | B |

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

In this example, each entity is mapped to a table, and the relationship between Student and Course is represented by the EnrollsIn table with foreign keys referencing the Student and Course tables. The Grade attribute is included in the EnrollsIn table. This is a simplified example, and in real-world scenarios, the mapping may involve additional considerations and normalization steps.

UNIT 6

Data Normalisation & Concurrency Control

Data normalization in the context of database management refers to the process of organizing and structuring data in a database to reduce redundancy and improve data integrity. The goal of normalization is to design a database schema in such a way that data is stored efficiently and relationships between different pieces of information are well-defined.

In easy language, data normalization is like tidying up your room. Imagine you have a messy room with clothes scattered all around. Normalization is like organizing these clothes in a systematic way. Instead of having the same shirt in multiple places, you put it in one designated spot. This helps you find things easily and keeps your room neat.

In a similar way, normalization in databases involves organizing data so that each piece of information is stored in one place. This avoids repeating the same data in multiple locations, making it easier to manage and preventing inconsistencies. It's like arranging your data neatly, which makes it more efficient and reduces the chances of errors.

What are Functional Dependencies?

Functional dependencies are rules that describe how one set of data uniquely determines another set of data in a database. In simpler terms, they define the relationships between different pieces of information.

2. Understanding the Relationship

- **Example:** Imagine you have a table with people's names and their social security numbers. In this case, the name uniquely determines the social security number.
- **Functional Dependency:** Name → Social Security Number

3. Key Concepts

- **Left-hand Side (LHS):** The set of data that determines another set.
- **Right-hand Side (RHS):** The set of data that is determined by another set.
- **Symbolically:** LHS → RHS

4. Why Functional Dependencies Matter

- **Data Integrity:** By understanding how data is related, we can ensure that changes to one piece of information don't lead to inconsistencies or errors in the database.
- **Database Design:** Functional dependencies play a crucial role in designing an efficient and well-organized database schema.

5. Normalization and Functional Dependencies

- **Normalization Process:** When designing a database, functional dependencies are used to identify and eliminate redundancy, making the data storage more efficient.
- **Example:** If a social security number depends only on the employee ID, it helps in creating a more organized and normalized database.

6. Types of Functional Dependencies

- **Full Functional Dependency:** The determination is not dependent on any subset of the LHS.
- **Partial Dependency:** The determination depends only on a part of the LHS, not the whole.

7. Identifying Functional Dependencies

- **Observing Data Patterns:** Look for patterns in the data to identify how one set of values uniquely determines another.
- **Understanding Business Rules:** Functional dependencies often reflect real-world relationships and business rules.

8. Challenges and Considerations

- **Changing Requirements:** Functional dependencies might need to be adjusted if there are changes in business rules or requirements.
- **Trade-offs:** Balancing between normalization and performance considerations in database design.

9. Conclusion

- **Summary:** Functional dependencies are essential for maintaining data integrity and designing efficient databases by understanding how different pieces of data relate to each other.
- **Continuous Process:** Identifying and managing functional dependencies is an ongoing process as the database evolves.

---

1. Introduction to Normalization:

- **Objective:** The goal of normalization is to organize data in a relational database to reduce redundancy and improve data integrity.
- **Key Idea:** Breaking down large tables into smaller, related tables to minimize data duplication.

2. First Normal Form (1NF):

- **Requirement:** Each column in a table must contain atomic (indivisible) values, and each record should be unique.
- **Example:** If you have a table with a column for phone numbers where multiple numbers are stored as a single string, it violates 1NF.

3. Second Normal Form (2NF):

- **Requirement:** The table must be in 1NF, and all non-key attributes should be fully functionally dependent on the primary key.
- **Key Concept:** Eliminating partial dependencies.
- **Example:** If you have a composite primary key (e.g., (StudentID, CourseID)), all other attributes should depend on both parts of the key.

4. Third Normal Form (3NF):

- **Requirement:** The table must be in 2NF, and there should be no transitive dependencies.
- **Transitive Dependency:** A non-prime attribute depends on another non-prime attribute, which, in turn, depends on the primary key.

- **Example:** If you have a table where a non-prime attribute (e.g., Address) depends on another non-prime attribute (e.g., City), and City depends on the primary key, it violates 3NF.

5. Illustrative Example:

- **Original Table (Not in 3NF):**

  cssCopy code

  1

  2

- **After 3NF Normalization:**

  markdownCopy code

6. Benefits of Normalization:

- **Reduces Redundancy:** Information is stored in a more compact form, minimizing data duplication.
- **Enhances Integrity:** Updates and inserts are less error-prone because data is organized more logically.

7. Considerations:

- **Trade-offs:** While normalization improves data integrity, it might involve more complex queries and joins.
- **Application Context:** The level of normalization depends on the specific requirements and use cases of the application.

In summary, normalization is a process that involves progressively organizing a database to ensure data integrity and minimize redundancy. The 1NF, 2NF, and 3NF are stages of this process, each addressing specific issues related to atomic values, partial dependencies, and transitive dependencies.

Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.

2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
BCNF	A stronger definition of 3NF is known as Boyce Codd's normal form.
4NF	A relation will be in 4NF if it is in Boyce Codd's normal form and has no multi-valued dependency.
5NF	A relation is in 5NF. If it is in 4NF and does not contain any join dependency, joining should be lossless.

Concurrency control in a database management system (DBMS) is like traffic management at a busy intersection. Imagine multiple cars (transactions) trying to cross the intersection simultaneously. To avoid collisions and ensure a smooth flow, traffic lights and rules are in place. Similarly, in a DBMS, when multiple users or processes are accessing and modifying data at the same time, concurrency control ensures that their actions don't interfere with each other, maintaining the consistency and reliability of the database.

Here's an explanation of concurrency control in easy language:

1. What is Concurrency Control?

- **Definition:** Concurrency control is a set of techniques used to manage and coordinate multiple transactions in a database system to prevent conflicts and maintain data integrity.

2. Why Do We Need Concurrency Control?

- **Scenario:** Imagine two people trying to update the same bank account balance simultaneously. Without concurrency control, one person's update might overwrite the other's, leading to incorrect balances.

3. Common Issues Without Concurrency Control:

- **Lost Updates:** One update overwrites another, and changes are lost.
- **Dirty Reads:** Reading uncommitted changes that might get rolled back.
- **Inconsistent Analysis:** Drawing conclusions from partially updated data.

4. Concurrency Control Techniques:

- **Locking:** Transactions request locks on data to ensure exclusive access. It's like one car getting the green light while others wait.
- **Timestamping:** Each transaction gets a unique timestamp, and the system uses these timestamps to determine the order of execution.
- **Isolation Levels:** Define the degree to which one transaction's changes are visible to other transactions.
- **Optimistic Concurrency Control:** Transactions are allowed to proceed without locks, but conflicts are checked before the final update.

5. Example - Locking:

- **Scenario:** Two users want to update the same bank account.
- **Locking Approach:** One user locks the account while updating, and the other has to wait. Once the first is done, the second user can proceed.

6. Isolation Levels - Read Committed:

- **Definition:** A transaction sees only committed changes made by other transactions.
- **Analogy:** You only see traffic updates after they have successfully passed through the intersection.

7. Isolation Levels - Serializable:

- **Definition:** Transactions are executed in a way that the final result is the same as if they were executed one after another.
- **Analogy:** Each car goes through the intersection as if it's the only one, avoiding any interference.

8. Benefits of Concurrency Control:

- **Data Integrity:** Ensures that changes to the database are correct and consistent.
- **Efficiency:** Allows multiple transactions to occur simultaneously without causing conflicts.

9. Challenges:

- **Performance Impact:** Introducing locks or coordination mechanisms may affect system performance.
- **Deadlocks:** Situations where transactions are waiting for each other, causing a standstill.

In a nutshell, concurrency control in a DBMS is like a traffic management system for database transactions, preventing data collisions and ensuring a smooth and reliable operation of the database even when multiple users are interacting with it concurrently.

---

1. What is Transaction Processing?

- **Definition:** Transaction processing is like making a deal or a trade in the world of databases. It's a way to ensure that a series of operations on the database either happen completely or don't happen at all.

2. Think of a Transaction as a Deal:

- **Example:** Imagine buying something online. The process involves adding items to your cart, entering your address, and making the payment. All these steps together form a transaction.

3. Properties of Transactions:

- **Atomicity:** The "all or nothing" rule. If any part of the transaction fails, the entire transaction is canceled, and the database remains unchanged.

- **Consistency:** Ensures that the database moves from one valid state to another after a transaction.
- **Isolation:** Transactions are independent of each other. One transaction's changes don't affect others until it's complete.
- **Durability:** Once a transaction is committed, its changes are permanent and survive any subsequent failures.

4. Example - Online Purchase:

- **Atomicity:** If you run out of money or there's an issue during payment, the entire transaction (purchase) is canceled.
- **Consistency:** The database reflects the accurate number of items in stock after the purchase.
- **Isolation:** Other people making purchases don't interfere with your transaction.
- **Durability:** Once you've paid and received a confirmation, the purchase is final.

5. Transaction Steps:

- **Begin Transaction:** Start of the transaction. It's like saying, "I'm about to do something important."
- **Execute Operations:** Perform the database operations (like inserting, updating, or deleting records).
- **Commit or Rollback:** If everything went well, commit the changes (making them permanent). If there's a problem, rollback the transaction, undoing any changes.

6. Transaction Processing System (TPS):

- **Definition:** It's a system designed to handle and manage transactions efficiently. Think of it as a smart cashier at a store managing multiple purchases simultaneously.

7. Importance of Transactions:

- **Data Integrity:** Ensures that the database remains accurate and reliable, even when multiple users are making changes simultaneously.
- **Reliability:** If something goes wrong during a transaction, the system can recover without leaving the database in a messed-up state.

8. Challenges:

- **Concurrency Control:** Managing transactions from different users happening at the same time without conflicts.
- **Performance Impact:** Ensuring that transaction processing doesn't slow down the overall system.

9. Real-World Analogy:

- **Picture It Like This:** When you withdraw money from an ATM, it's a transaction. The system ensures that the money is deducted from your account correctly, and if anything goes wrong, it rolls back to the previous state.

In summary, transaction processing in a DBMS is like making a deal or a trade. It ensures that changes to the database happen reliably, completely, and without messing up the overall state of the data, even when multiple things are happening at the same time.

1. What are Locking Techniques?

- **Definition:** Locking techniques are like a way of ensuring order in a busy kitchen. When multiple chefs (transactions) are working on different parts of a meal (data), locks prevent them from interfering with each other's work.

2. Why Do We Need Locks?

- **Scenario:** Imagine two people updating the same bank account balance at the same time. Locks prevent one person's changes from conflicting with the other's, avoiding errors.

3. Common Types of Locks:

- **Shared Lock:** Like borrowing a book from a library. Many people can read it, but only one person can take it out at a time.
- **Exclusive Lock:** Like having a "Do Not Disturb" sign on a hotel room. Only one person (transaction) can access the data exclusively.

4. How Locking Works:

- **Requesting a Lock:** When a transaction wants to read or modify data, it asks for a lock.
- **Granting a Lock:** If the data is available, the system grants the lock. If not, the transaction may need to wait until the data is free.

5. Example - Shared Lock:

- **Scenario:** Two people want to check their bank balances at the same time.
- **Shared Lock Approach:** Both can view their balances simultaneously because reading doesn't interfere with others.

6. Example - Exclusive Lock:

- **Scenario:** Two people want to update their contact information in the same database.
- **Exclusive Lock Approach:** The system allows only one person to update at a time to avoid conflicting changes.

7. Lock Duration:

- **Short-Term Locks:** Applied only during the specific operation and released immediately after.
- **Long-Term Locks:** Applied for a more extended period, useful for a series of related operations.

8. Benefits of Locking:

- **Data Integrity:** Prevents conflicts and ensures that changes happen in an orderly manner.

- **Isolation:** Each transaction operates independently without interference from others.

9. Drawbacks and Challenges:

- **Deadlocks:** Situations where transactions are waiting for each other, causing a standstill.
- **Performance Impact:** Too many locks can slow down the system.

10. Real-World Analogy:

- **Think of It Like This:** When you're at a gas station, if someone is using a pump, it's like they have an exclusive lock on it. Others have to wait until it's free.

In summary, locking techniques in a DBMS are like rules in a busy kitchen, ensuring that transactions don't interfere with each other when accessing or modifying data. Shared locks allow multiple transactions to read simultaneously, while exclusive locks ensure only one transaction can modify data at a time, preventing conflicts and maintaining order.