MSc Computational Sciences (FYIMP)

Department of IT

Kannur University

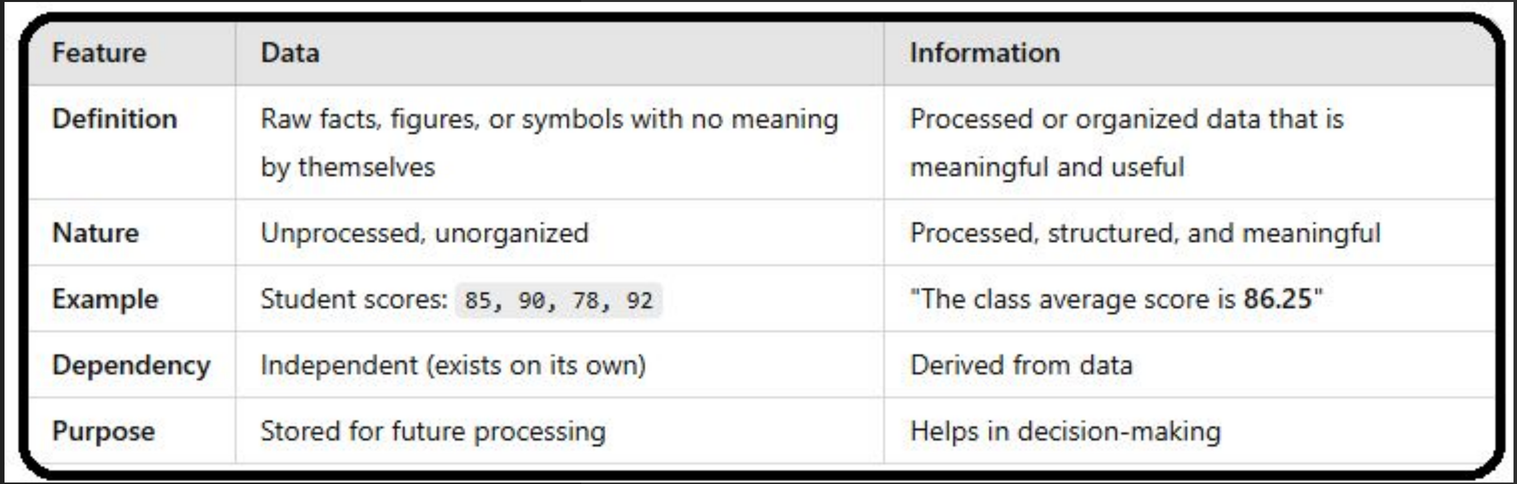
Class 1

Unit II
Database Management System: Introduction - Definition of Database and Database Management System (DBMS) - Unstructured, Semi Structured and Structured Data - Characteristics of DBMS - Advantages of DBMS - Components of DBMS - Database Users - Data Models and Schema - Three Schema architecture - Database Languages - Database Architectures and Classification. Case Study: Structured Query Language (SQL) - Types of Commands: DML - DDL - DCL. Table: Definition - Data Types - Primary Key - Foreign Key - Creation - Deletion - Modification - Updation. Constraints: NULL - NOT NULL - UNIQUE. Displaying Data from Tables: Select - Logical operators - WHERE - Pattern matching - ORDER BY.

Foundations of Computer Science

KU2DSCCSE102

Data?!
Information?!



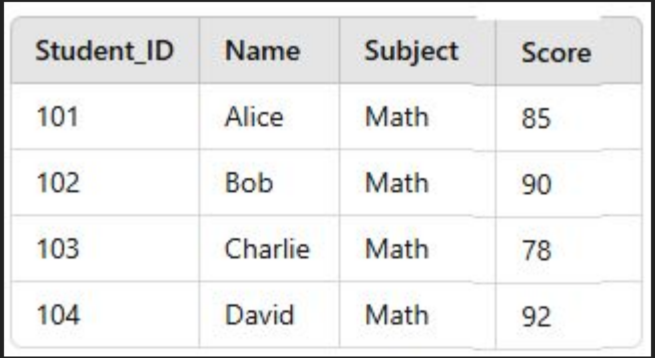
Feature	Data	Information
Definition	Raw facts, figures, or symbols with no meaning by themselves	Processed or organized data that is meaningful and useful
Nature	Unprocessed, unorganized	Processed, structured, and meaningful
Example	Student scores: `85, 90, 78, 92`	"The class average score is **86.25**"
Dependency	Independent (exists on its own)	Derived from data
Purpose	Stored for future processing	Helps in decision-making

Data?! Information?!

Which Comes First? - Data or Information?!

- Data comes first because information is derived from data

- Without raw data, there would be nothing to process into meaningful insights

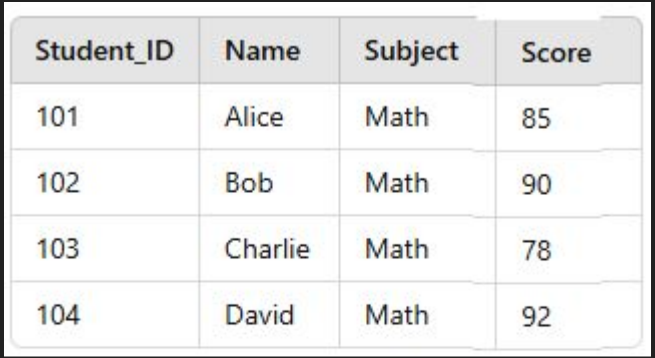


Student_ID	Name	Subject	Score
101	Alice	Math	85
102	Bob	Math	90
103	Charlie	Math	78
104	David	Math	92

Which Comes First? - Data or Information?!

- Consider the following table
- This raw data alone does not provide insights!

- Now, let's say we process this data to find meaningful insights
  - How the students performed in Mathematics?
  - The average score is 86.25!



Student_ID	Name	Subject	Score
101	Alice	Math	85
102	Bob	Math	90
103	Charlie	Math	78
104	David	Math	92

Which Comes First? - Data or Information?!

- Consider the following table
- This raw data alone does not provide insights!

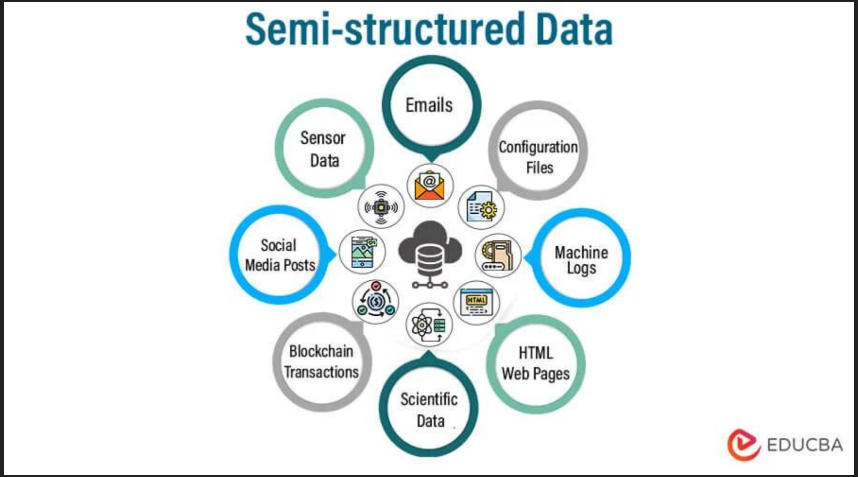
- Now, let's say we process this data to find meaningful insights
    - Ok! Who is the topper?!
    - David! His score is 104!

These processed insights are information, which can be used for decision-making, such as rewarding the top student or identifying students who need improvement!

Examples of unstructured data
Agent notes
Surveys
Web forms
Mail
Chats
Quality evaluations

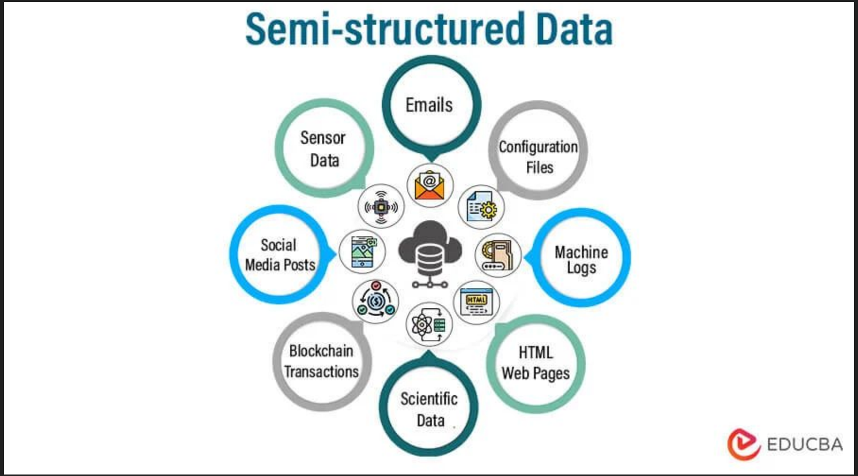
Unstructured Data

- Unstructured data is raw
- It is not organized in any predefined format
- It lacks a clear structure
- Examples of unstructured data include videos, emails, images, and HTML content
- This kind of data makes up between 80 and 90 percent of all data generated globally



Semi structured Data

- Semi-structured data has some organization but does not follow a strict structure like rows and columns
- Markup languages like HTML and XML are also a common source for semi-structured data
- It is more flexible in nature and can contain tags or markers to separate elements



Semi structured Data

- Eg:- E-Commerce Product Catalog

Structured Data

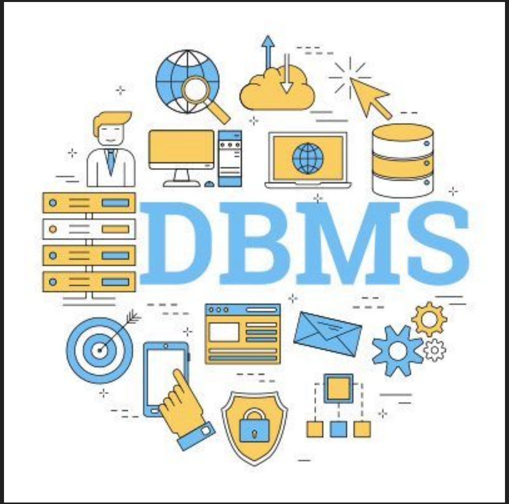
- Structured data is organized in a predefined manner, typically in rows and columns (like a table in a database)
- This data is highly organized and easily searchable
- Eg:- Student records in a university database

Database Management System (DBMS)

- Imagine a university keeps student records in paper files
- As student numbers grow, managing, retrieving, and updating records becomes challenging!
  - How a student's marks need to be updated?
  - What if the university wants to find the top scorer in a subject?
  - How multiple departments can simultaneous access to data?

Database Management System (DBMS)

A Database Management System (DBMS) solves these challenges by providing an efficient way to store, manage, retrieve, and manipulate data digitally

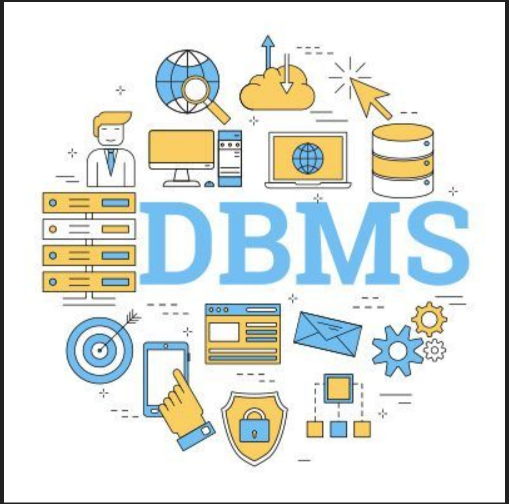


Database Management System (DBMS)

- A Database Management System (DBMS) is software that allows users to store, organize, manage, and retrieve data efficiently in a structured manner

- It acts as an interface between the database and the users or applications accessing the data

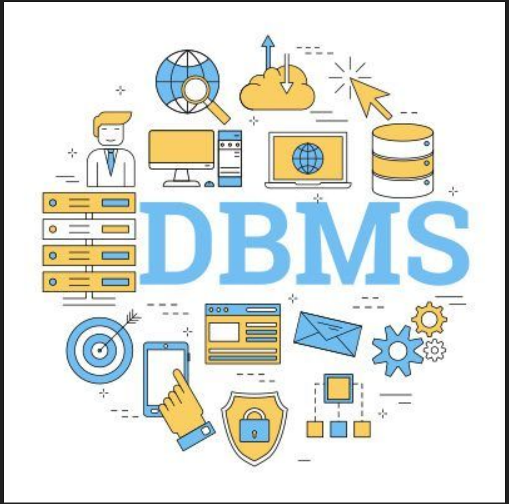
- Eg:- MySQL, Oracle, PostgreSQL, Microsoft SQL Server



Database Management System (DBMS)

Key Features of DBMS

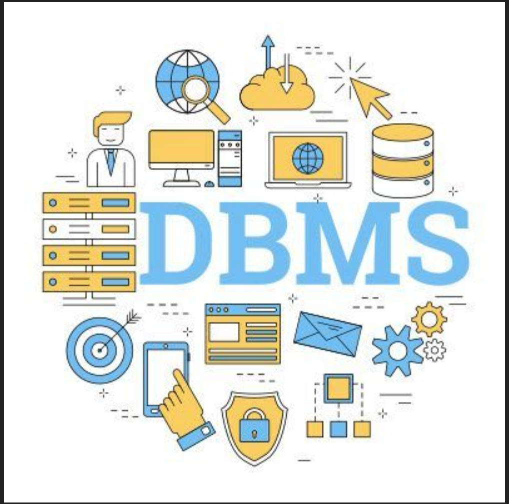
- Data Organization: Stores data in tables, making it structured and easy to manage
- Efficient Retrieval: Uses queries (SQL) to retrieve specific data quickly
- Multi-User Access: Allows multiple users to access and modify data simultaneously
- Data Integrity & Security: Ensures data accuracy and prevents unauthorized access



Database Management System (DBMS)

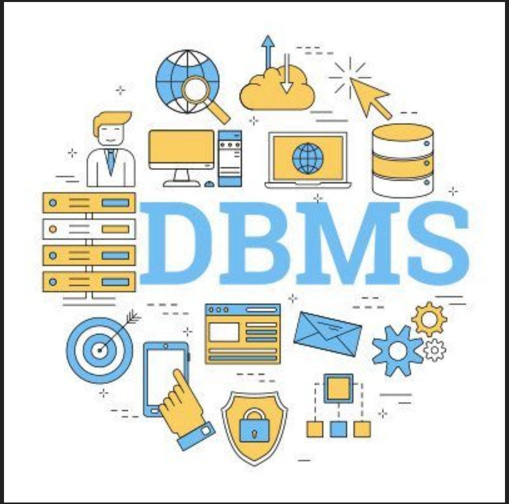
- Without a DBMS, handling large amounts of data manually would be inefficient and error-prone

- Modern applications, websites, and businesses rely on DBMS to manage their data efficiently



Database Management System (DBMS)

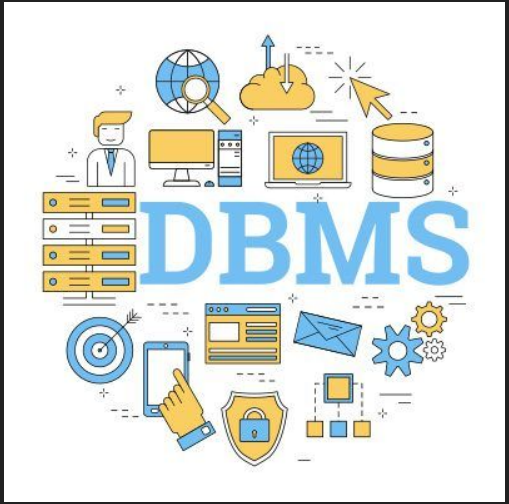
- Without a DBMS, handling large amounts of data manually would be inefficient and error-prone
- Modern applications, websites, and businesses rely on DBMS to manage their data efficiently



Characteristics of DBMS

Data Abstraction

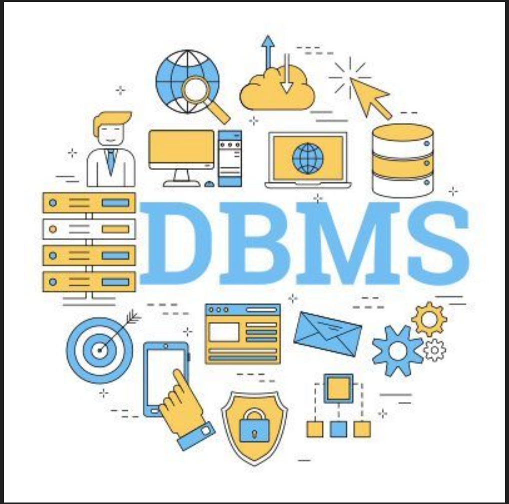
- Hides the complexity of data storage from users
- Users interact with logical structures (tables, views) rather than physical storage details



Characteristics of DBMS

Data Independence

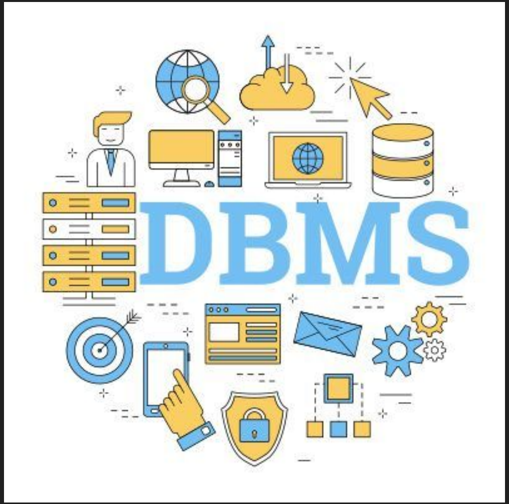
- Changes in storage structure (physical level) do not affect how data is accessed at the logical level
- Two types:
  - Logical Data Independence: Schema changes (adding columns) don’t affect applications
  - Physical Data Independence: Changing storage mechanisms does not affect database structure



Characteristics of DBMS

Data Integrity

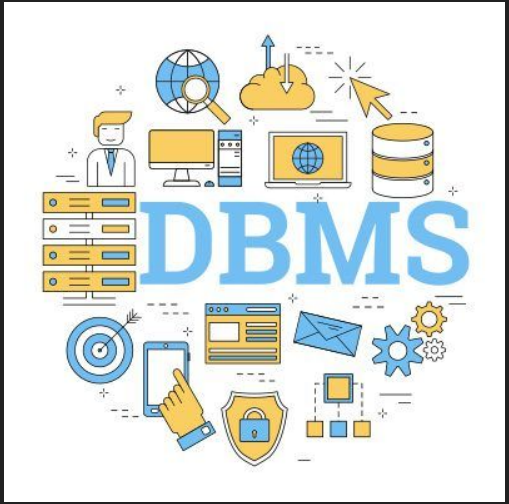
- Ensures data integrity through ACID properties
    - Atomicity: A transaction is all or nothing (e.g., money transfer)
    - Consistency: Database remains in a valid state before & after transactions
    - Isolation: Concurrent transactions don't interfere with each other
    - Durability: Once committed, transactions are permanent, even after system failure



Characteristics of DBMS

Data Security & Authorization

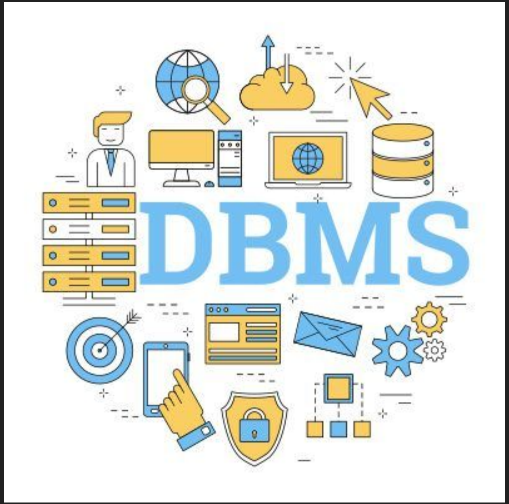
- Controls who can access what data using authentication and authorization
- Uses roles, privileges, and encryption for security.
- Example: A student can see grades, but only an admin can modify them



Characteristics of DBMS

Multi-User Access & Concurrency

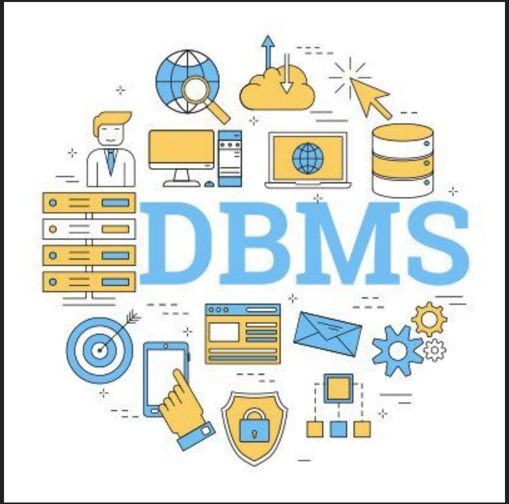
- Multiple users can access and modify data simultaneously without conflicts
- Uses locking mechanisms to prevent data inconsistencies
- Example: Two customers booking the last flight ticket won't create duplicate bookings



Characteristics of DBMS

Data Redundancy Control & Normalization

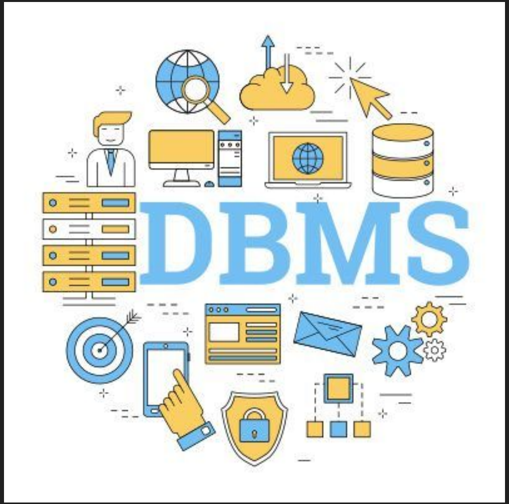
- Reduces duplicate data using normalization techniques (e.g., breaking tables into smaller relations)
- Improves storage efficiency and avoids data inconsistency
- Example: Instead of storing a department name multiple times, a separate Departments table is used



Characteristics of DBMS

Backup & Recovery

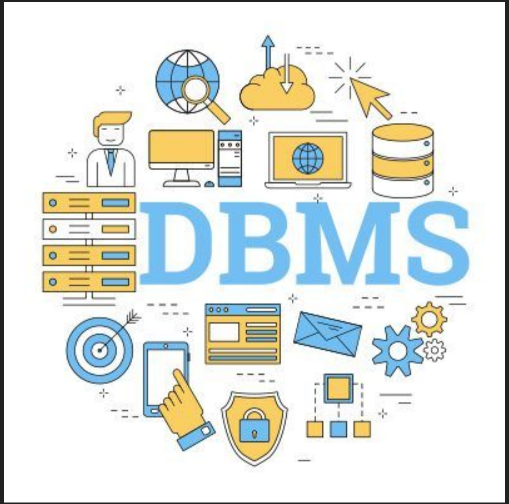
- Ensures data is not lost due to failures
- Supports automatic backups and restore mechanisms
- Example: If a system crashes, the transaction log helps recover lost data



Characteristics of DBMS

Query Processing & Optimization

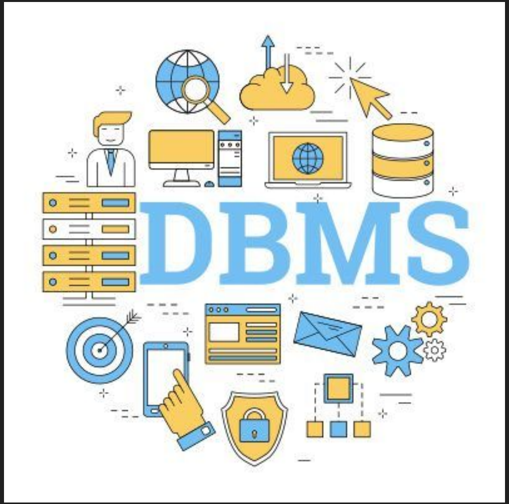
- Uses SQL queries to fetch and manipulate data efficiently
- Query optimization improves performance by choosing the best execution plan



Characteristics of DBMS

Data Integrity & Consistency

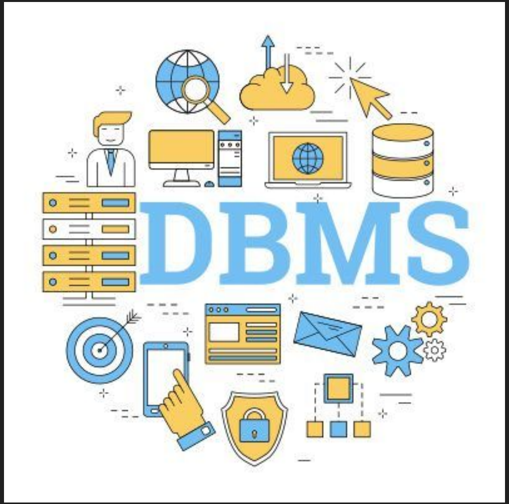
- Ensures correctness and validity of stored data using constraints
- Example:- Prevention of blank and duplicate entries in a table



Characteristics of DBMS

Scalability & Performance

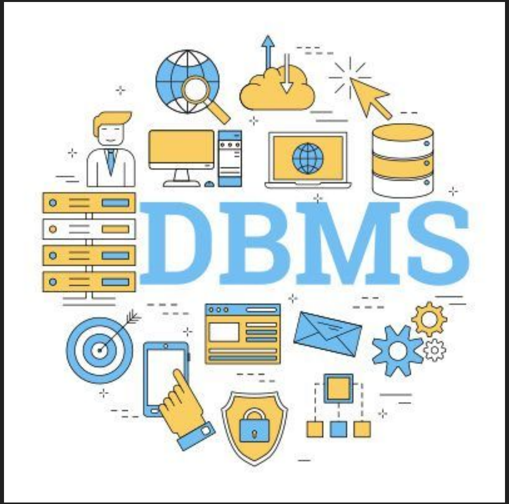
- Supports handling large datasets efficiently.
- Scales vertically (better hardware) or horizontally (distributed databases).



Advantages of DBMS

Data Integrity & Accuracy

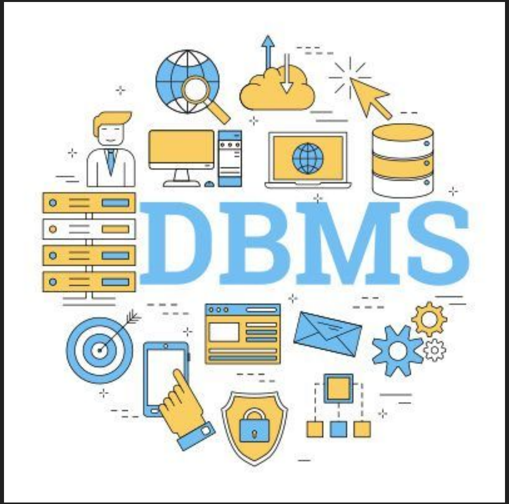
- Ensures correct and consistent data using constraints (Primary Key, Foreign Key)
- Prevents duplicate, missing, or incorrect entries
- Example: In a student database, each student has a unique ID, preventing duplicate records



Advantages of DBMS

Data Security & Controlled Access

- Provides user authentication and role-based access control
- Prevents unauthorized access and ensures data privacy
- Example: In a bank database, customers can view their accounts, but only bank staff can update balances

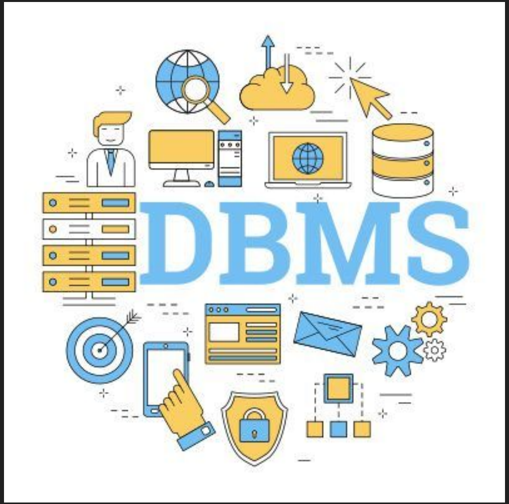


Advantages of DBMS

Efficient Data Retrieval & Query Processing

- Uses SQL queries to fetch data quickly
- Indexes and optimization speed up search operations

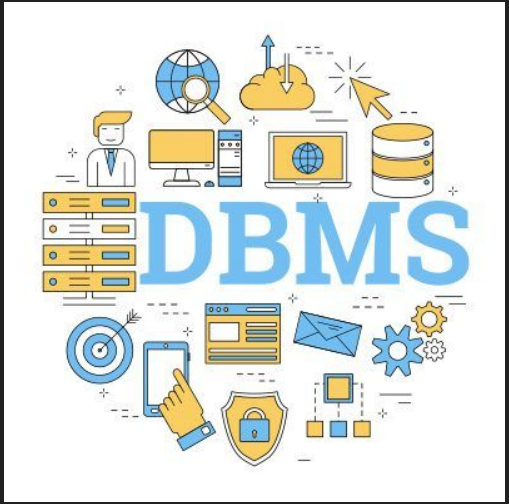
- Example: A SELECT query retrieves a customer’s order history in seconds, unlike searching through multiple file



Advantages of DBMS

Data Redundancy Reduction

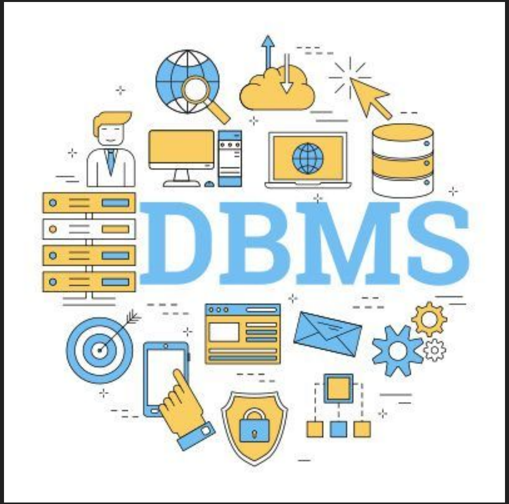
- Eliminates duplicate data using normalization techniques
- Saves storage space and ensures data consistency
- Example: Instead of storing a customer's address multiple times, a separate Customer Details table is used



Advantages of DBMS

Multi-User Access & Concurrency

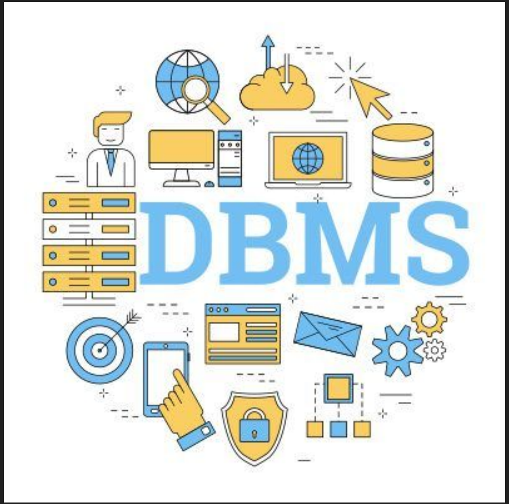
- Allows multiple users to access and modify data simultaneously
- Uses locking mechanisms to prevent conflicts
- Example: An e-commerce platform allows thousands of customers to browse and purchase items at the same time.



Advantages of DBMS

Backup & Recovery

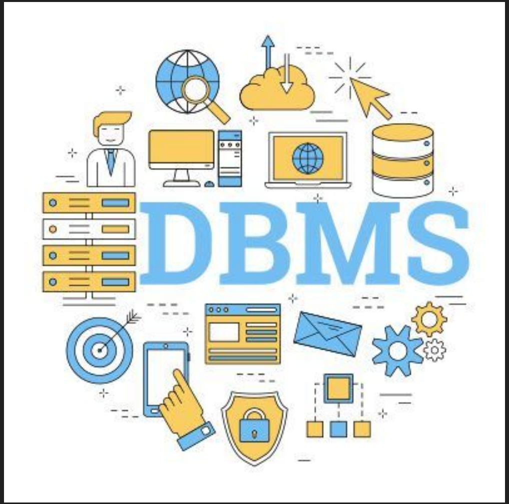
- Automated backups protect against data loss
- Recovery mechanisms restore data after system failures



Advantages of DBMS

Data Independence

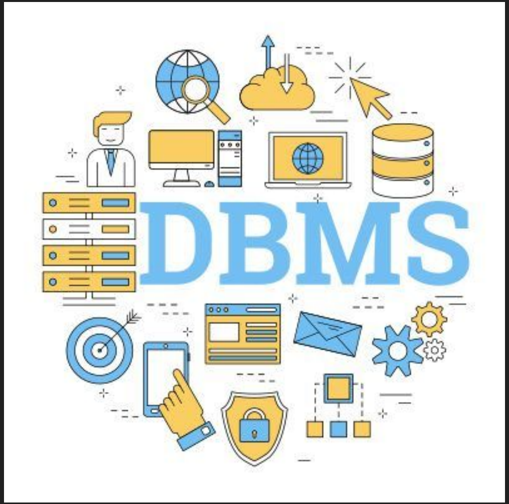
- Changes in physical storage do not affect how data is accessed
- Logical structure changes without affecting the application
- Example: A company can upgrade from HDD to SSD storage without changing its database queries



Advantages of DBMS

Scalability & Performance

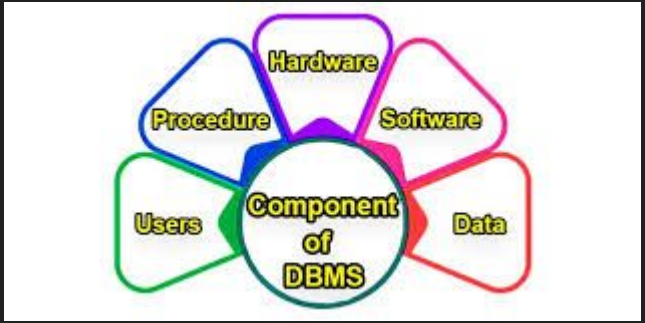
- Supports large datasets and high user traffic
- Scales horizontally (adding more servers) or vertically (upgrading hardware)
- Example: Social media platforms like Facebook use distributed DBMS to handle millions of active users



Advantages of DBMS

Scalability & Performance

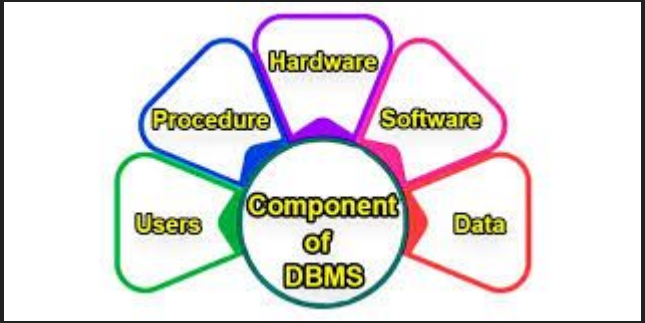
- Supports large datasets and high user traffic
- Scales horizontally (adding more servers) or vertically (upgrading hardware)
- Example: Social media platforms like Facebook use distributed DBMS to handle millions of active users



Components of DBMS

Hardware

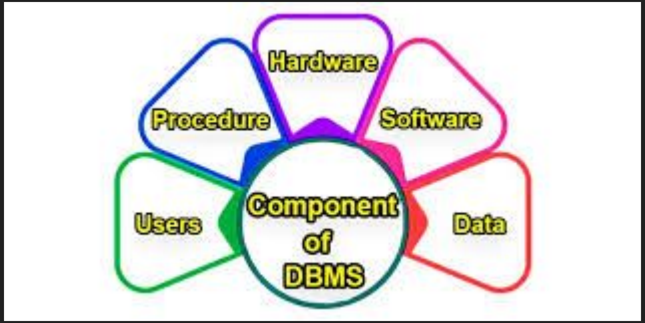
- Provides the physical infrastructure for data storage and processing
- Includes servers, storage devices, network devices, and client computers
- Determines performance, availability, and scalability of the database system



Components of DBMS

Software

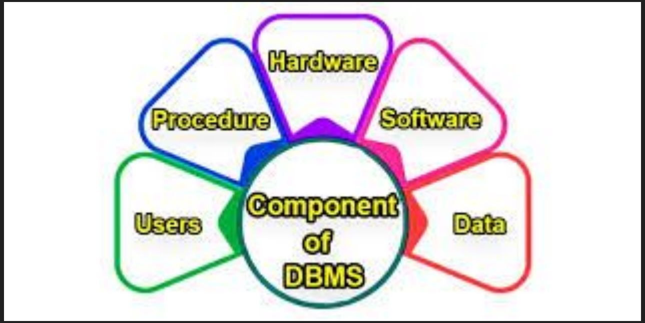
- The actual DBMS software that manages the database
- Consists of:
  - DBMS Software (e.g., MySQL, Oracle, PostgreSQL, MongoDB)
  - Operating System (Windows, Linux) that supports the DBMS
  - Application Programs (banking software, e-commerce apps) that interact with the DBMS



Components of DBMS

Data

- The most important component—actual stored information
- Includes structured, semi-structured, and unstructured data
- Organized in tables, documents, or files within the DBMS
- Example: A university database contains student records, courses, and faculty details

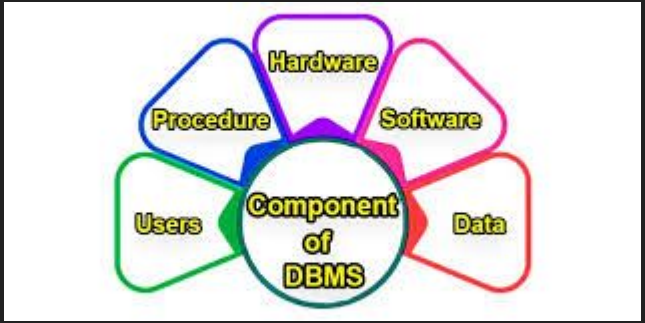


Components of DBMS

Database Access Language (SQL)

- Allows users to interact with the database via queries
- SQL (Structured Query Language) is the most common database language

Components of DBMS



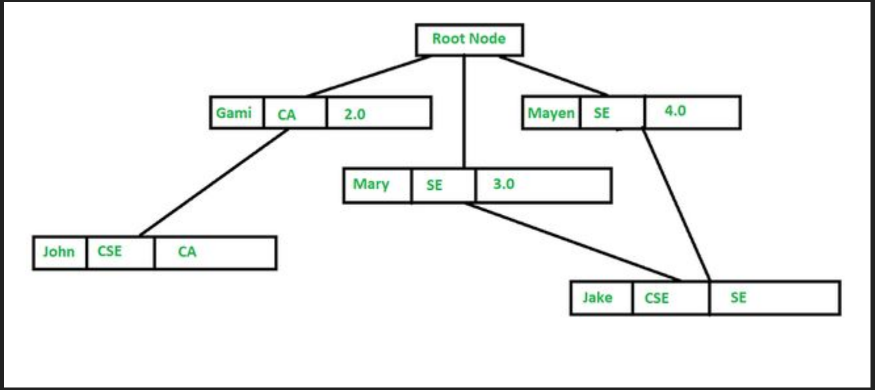
Users

- Different users interact with the DBMS based on their roles
- Database Administrators (DBAs) – Manage security, backups, and performance
- Developers – Write queries, design schemas, and optimize data retrieval
- End Users – Use applications (e.g., customers of an online shopping site)
- Example: A DBA in a hospital database ensures that only doctors access patient records

Data Models

- A Data Model defines how data is structured, stored, and manipulated in a database

- It provides a logical framework for organizing data



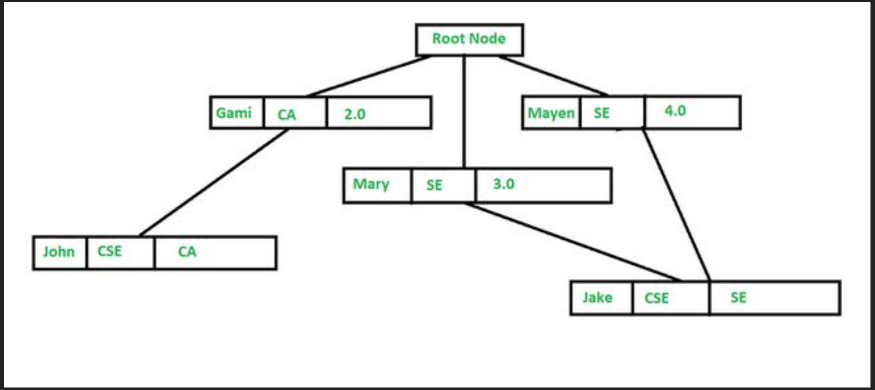
Data Models

hierarchical Data Model

- Organizes data in a tree-like hierarchy where each record has a single parent and many children

- Suited for applications with a clear hierarchical relationship, such as organizational structures

hierarchical Data Model



Data Models

- Refer the Figure
- There are students and courses
- Each student can enroll for multiple courses - the relationship between them is one to many

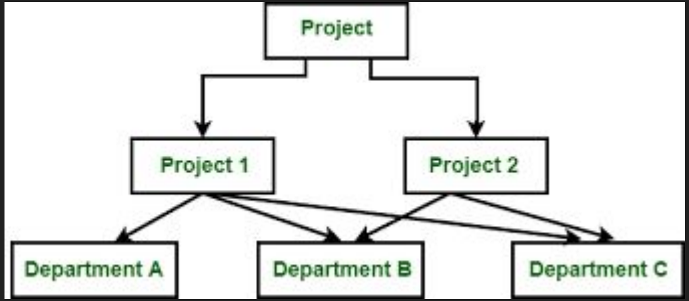
hierarchical Data Model - Advantages

- The tree-like structure of the hierarchical model is easy to understand, which simplifies the design and navigation of databases
- Inherently maintains data integrity. Each child record has only one parent, which helps prevent redundancy and preserves the consistency of data across the database
- The model allows for fast and efficient retrieval of data
- Provides enhanced security - Access to data can be controlled by restricting users to certain segments of the tree
-

hierarchical Data Model - Disadvantages

- Making changes to the database structure is difficult
- Many to many relationship is not supported
- If a child has multiple parents, each parent child relationship is to be represented separately - This redundancy can consume additional storage and complicate data management

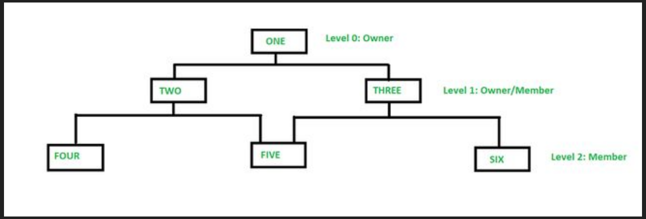
Network Data Model



Data Models

- Utilizes a graph structure
- Allows records to have multiple parent and child relationships, supporting many-to-many relationships
- Ideal for complex relationships like social networks or transportation routes

Network Data Model

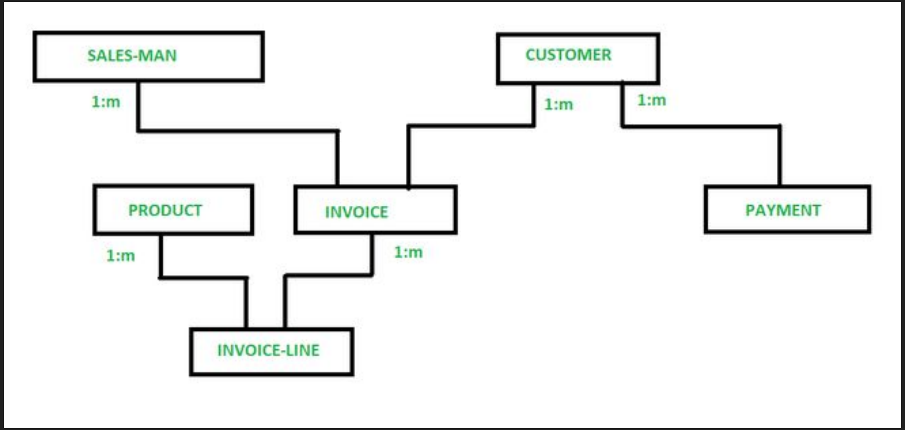


Data Models

- In the Figure you can see that member TWO has only one parent ONE whereas member FIVE has two parent

Network Data Model

- Example



Data Models

Network Data Model - Advantages

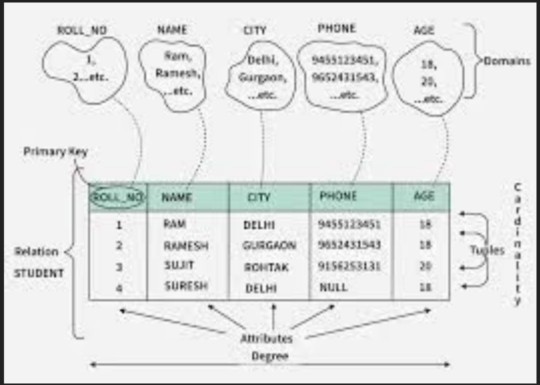
- Capable of handling multiple types of relationships
- This network does not allow a member to exist without an owner which leads to the concept of Data integrity
- This model allows to represent multi parent relationships

Data Models

Network Data Model - Disadvantages

- The schema or the structure is very complex
- The design or the structure of this model is not user-friendly

Data Models

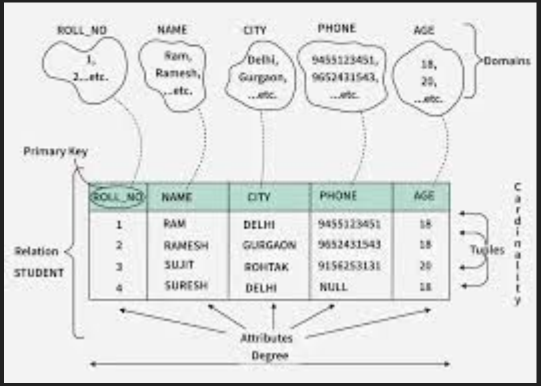


ROLL_NO	NAME	CITY	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI	NULL	18

Relational Data Model

- Represents data in tables (relations) consisting of rows and columns
- Widely used due to its simplicity and flexibility, suitable for various applications
- A customer database with tables for customer information, orders, and products

Relational Data Model



ROLL_NO	NAME	CITY	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI	NULL	18

Data Models=

- Represents data in tables (relations) consisting of rows and columns
- Widely used due to its simplicity and flexibility, suitable for various applications
- A customer database with tables for customer information, orders, and products

Relational Data Model

- Let us see how relational model can be created
- See a description of a University department: "In a university, there are several departments. Each department has its own teachers. Each department offers various programmes. Within these programmes, teachers offer courses"

Relational Data Model

- Let us see how relational model can be created

- See a description of a University department: "In a university, there are several departments. Each department has its own teachers. Each department offers various programmes. Within these programmes, teachers offer courses"

- We can see that there are four tables here:
  - Department
  - Teacher
  - Programmes
  - Courses

Relational Data Model

- Now we have to identify the columns in these tables - the kind of data that will be stored in the tables concerned
- Take the department table. Columns are
  - dept_id (Department ID)
  - dept_name (Department Name)
- Sample data in the department table

dept_id	dept_name
001	Information Technology
002	Mathematics
003	Statistics

Relational Data Model

- Now we have to identify the columns in these tables - the kind of data that will be stored in the tables concerned

- Take the department table. Columns are
  - dept_id (Department ID)
  - dept_name (Department Name)

Relational Data Model

- Now we have to identify the columns in these tables - the kind of data that will be stored in the tables concerned

- Take the department table. Columns are
  - dept_id (Department ID)
  - dept_name (Department Name)

- Sample data in the department table
- This table can many rows (actual data)
- We can see three rows (records)

dept_id	dept_name
T001	Information Technology
T002	Mathematics
T003	Statistics

Relational Data Model

dept_id	dept_name
T001	Information Technology
T002	Mathematics
T003	Statistics

- To distinguish between data in each rows a unique identifier must be identified for each record in a table
- dept_id can be chosen as the unique identifier for each record
- This unique identifier is called as the primary key of a table! Hence the table structure can be rewritten as
  - dept_id (Primary Key)
  - dept_name (Unique)

Relational Data Model

- Now we have to identify the columns in these tables - the kind of data that will be stored in the tables concerned

- Take the department table. Columns are
  - dept_id (Department ID)
  - dept_name (Department Name)

- Sample data in the department table
- This table can many rows (actual data)
- We can see three rows (records)
- To distinguish between data in each rows
- a unique identifier must be identified for each record in a table.

dept_id	dept_name
T001	Information Technology
T002	Mathematics
T003	Statistics

Relational Data Model

- Let us identify the columns in Teacher table
  - teach_id (Teacher ID) Primary Key
  - teach_name (Teacher Name)
  - dept_id (Department Name)

- Sample data in the department table

Relational Data Model

- Let us identify the columns in Teacher table
  - teach_id (Teacher ID) Primary Key
  - teach_name (Teacher Name)
  - dept_name (Department Name)

- See the sample data
- Since we have kept department name here, it leads to several inconveniences
  - Department names are to be stored repetitively in each row, if there are many teachers in a department!

teach_id	teach_name	dept_name
T001	Mohammed	IT
T002	James	Mathematics
T003	Ramkumar	Statistics
T004	Vivek	Information Technology

Relational Data Model

- Let us identify the columns in Teacher table
  - teach_id (Teacher ID) Primary Key
  - teach_name (Teacher Name)
  - dept_name (Department Name)

- See the sample data
- Since we have kept department name here, it leads to several inconveniences
  - Same Department name can be specified in multiple ways such as IT, Information Technology -
  - This leads to inconsistencies!

teach_id	teach_name	dept_name
T001	Mohammed	IT
T002	James	Mathematics
T003	Ramkumar	Statistics
T004	Vivek	Information Technology

Relational Data Model

- To solve this problem, we have to replace the column dept_name with dept_id
- Let us identify the columns in Teacher table
  - teach_id (Teacher ID) Primary Key
  - teach_name (Teacher Name)
  - dept_id (Department ID)

- See the sample data
- Even here we have few issues!

teach_id	teach_name	dept_id
T001	Mohammed	D001
T002	James	D002
T003	Ramkumar	D003
T004	Vivek	D001

Relational Data Model

- To solve this problem, we have to replace the column dept_name with dept_id
- Let us identify the columns in Teacher table
  - teach_id (Teacher ID) Primary Key
  - teach_name (Teacher Name)
  - dept_id (Department ID)

- See the sample data
- How to get the department name?!
- It is not stored in this table! Its stored in the department table!
- It is possible to provide a code

that is not available in the department table!

teach_id	teach_name	dept_id
T001	Mohammed	D001
T002	James	D002
T003	Ramkumar	D003
T004	Vivek	D001

Relational Data Model

- Let us take these two tables: department and teachers
  - Every teacher belongs to a department!
  - Department is specified as a code in the teacher table
  - This code is already stored in the department table

dept_id	dept_name
D001	Information Technology
D002	Mathematics
D003	Statistics

teach_id	teach_name	dept_id
T001	Mohammed	D001
T002	James	D002
T003	Ramkumar	D003
T004	Vivek	D001

Relational Data Model

- Let us take these two tables: department and teachers
  - Every teacher belongs to a department!
  - If a teacher's department code doesn't match any code in the department table, it would be like saying a teacher belongs to a department that doesn't exist—that's not allowed!

dept_id	dept_name
D001	Information Technology
D002	Mathematics
D003	Statistics

teach_id	teach_name	dept_id
T001	Mohammed	D001
T002	James	D002
T003	Ramkumar	D003
T004	Vivek	D001

Relational Data Model

- Let us take these two tables: department and teachers
  - A relation is maintained between the department and teachers
  - Relation is based on a column (dept_id) in the teachers table - this column refers to the dept_id column in the department table

department

dept_id	dept_name
D001	Information Technology
D002	Mathematics
D003	Statistics

teacher

teach_id	teach_name	dept_id
T001	Mohammed	D001
T002	James	D002
T003	Ramkumar	D003
T004	Vivek	D001

Relational Data Model

- Let us take these two tables: department and teachers
  - dept_id column in the teachers table is called as the foreign key!
  - Once this relationship is maintained, we can not add a row in the teachers table with a value for dept_id that does not exist in the table department!

department

dept_id	**dept_name**
D001	Information Technology
D002	Mathematics
D003	Statistics

teacher

teach_id	**teach_name**	**dept_id**
T001	Mohammed	D001
T002	James	D002
T003	Ramkumar	D003
T004	Vivek	D001

Relational Data Model

- Primary Key:- A unique identifier for each record in a table
- Foreign Key:- A column in one table that refers to the primary key of another table

Schema

- A schema is essentially the blueprint or structure of a database. It defines how data is organized, specifying the tables, the columns within those tables, the data types for each column, and the relationships between the tables through keys and constraints

- Remember the tables and their structures that we had discussed in the case of a University - it was the schema for the University database

Relational Data Model

- Let us take these two tables: department and teachers
  - Hence we say that there is a relationship between department and teacher tables!
  - In the Teacher table, the DepartmentID column contains such as D001 and D002. These values must match one of the DepartmentID values in the Department table. This ensures that every teacher is assigned to an existing department!

dept_id	dept_name
D001	Information Technology
D002	Mathematics
D003	Statistics

teach_id	teach_name	dept_id
T001	Mohammed	D001
T002	James	D002
T003	Ramkumar	D003
T004	Vivek	D001

Relational Data Model

- Let us take these two tables: department and teachers
  - Hence we say that there is a relationship between department and teacher tables!
  - Note that dept_id in teacher table refers to the primary key of employee table
  - Hence dept_id in Teacher table is a foreign key!

dept_id	dept_name
D001	Information Technology
D002	Mathematics
D003	Statistics

teach_id	teach_name	dept_id
T001	Mohammed	D001
T002	James	D002
T003	Ramkumar	D003
T004	Vivek	D001

Relational Data Model=

- Let us identify the columns in Programme table
  - prog_id (Teacher ID) Primary Key
  - prog_name (Teacher Name)
  - dept_id (Department ID) Foreign Key

prog_id	teach_name	dept_id
P001	FYIMP	D001
P002	MSc Mathematics	D002
P003	MSc Statistics	D003
P004	MCA	D001

Relational Data Model=

- What are the columns in the Course table?

Relational Data Model=

- What are the columns in the Course table?
    - c_id (Course ID) Primary Key
    - c_name (Course Name)
    - prog_id (programme ID) Foreign Key
    - t_id (Teacher ID) Foreign Key

c_id	c_name	prog_id	t_id
P001	FCS	P001	T001
P002	Set Theory	P002	T002

Schema

Schema

- A database schema defines the structure and organization of data within a database
- It outlines how data is logically stored, including the relationships between different tables and other database objects

Three Schema Architecture

- The three schema architecture is a framework designed to separate the database system into three distinct layers, each with its own role
- This separation helps achieve data independence and simplifies the design, implementation, and maintenance of a database
    - External Schema (View Level)
    - Conceptual Schema (Logical Level)
    - Internal Schema (Physical Level)

Three Schema Architecture - External Schema (View Level)

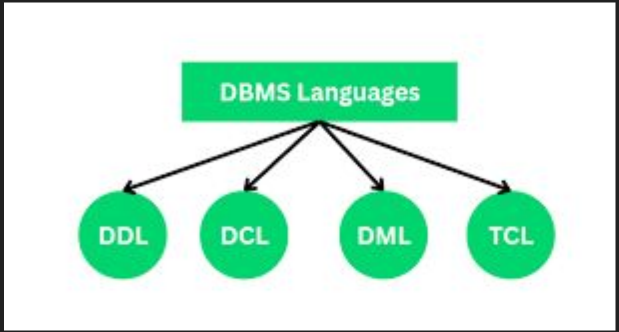
- The user view of the data
- It defines how users can access and see the data
- Each user or group may have a different view, displaying only the data relevant to them
- Sensitive data can be hidden, and data can be presented in a way that makes sense to the user
- Eg:- In a university database, students might see their grades and courses, while staff concerned will enjoy the privilege to add/edit the grade

Three Schema Architecture - Conceptual Schema (Logical Level)

- The logical view of the data. It defines what data is stored in the database and how it relates to other data
- It abstracts away the details of physical storage, focusing solely on the logical relationships among data
- Changes in the physical storage of data do not affect the conceptual schema, providing logical data independence
- Eg:- For the university, the conceptual schema defines the structure of departments, teachers, programmes, and courses, along with the relationships among them

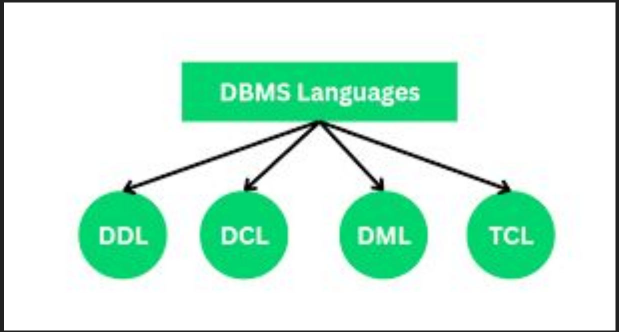
Three Schema Architecture - Internal Schema (Physical Level)

- The physical view of the data. It defines how data is stored and organized on the computer hardware
- Changes at the physical level (like new storage devices or indexing strategies) don’t affect the conceptual schema
- Eg:- The university database might store records in B-trees or hash tables, but these details are hidden from the user and the conceptual schema



Database Languages

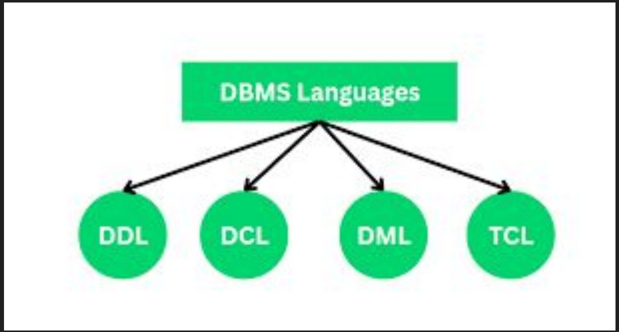
- Database languages help users manage and access databases
- The most widely used database language is SQL (Structured Query Language)
- Include Data Definition Language (DDL), Data Manipulation Language (DML), Data Control Language (DCL), and Transaction Control Language (TCL)



Database Languages

Data Definition Language (DDL)

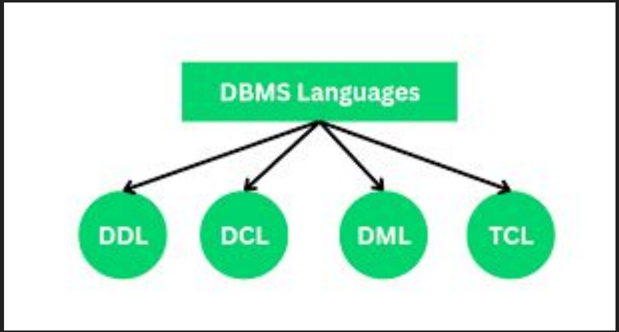
- These commands define and modify the structure (schema) of a database
- This includes creating, altering, and deleting tables
- Eg:- CREATE, ALTER, DROP



Database Languages

Data Manipulation Language (DML)

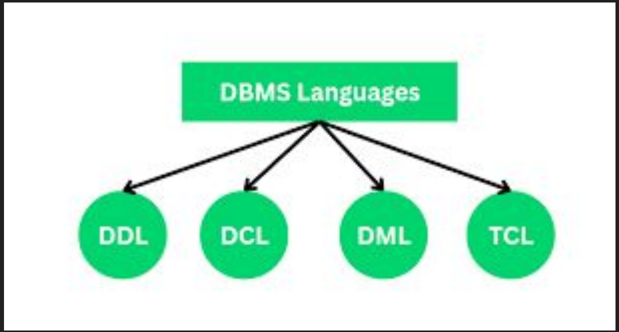
- Used to manage data in database tables
- Used to retrieve and modify data
- Key commands include SELECT, INSERT, UPDATE, and DELETE



Database Languages

Data Control Language (DCL)

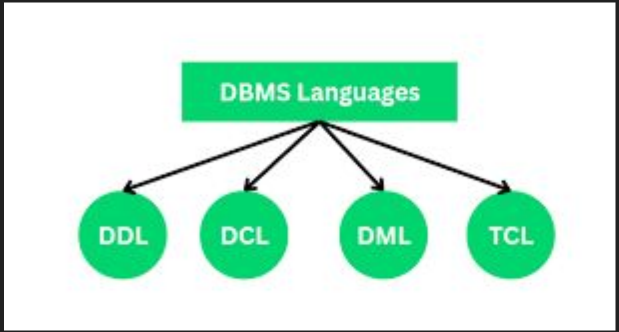
- These command are used to manage access and permissions to the database objects
- They help maintain data security by controlling who can access or modify data
- Eg:- GRANT, REVOKE



Database Languages

Transaction Control Language (TCL)

- TCL commands manage the transactions in a database to ensure data integrity and consistency
- Transactions are sequences of operations that are treated as a single logical unit
- Eg:- COMMIT, ROLLBACK, SAVEPOINT



Database Languages

Transaction Control Language (TCL)

- TCL commands manage the transactions in a database to ensure data integrity and consistency
- Transactions are sequences of operations that are treated as a single logical unit
- Eg:- COMMIT, ROLLBACK, SAVEPOINT

PostgreSQL

- PostgreSQL is an open source relational database system
- It uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads
- PostgreSQL evolved from the Ingres project at the University of California, Berkeley
- It is supported on all major operating systems, including Windows, Linux, macOS, FreeBSD, and OpenBSD

PostgreSQL

- PostgreSQL is an open source relational database system
- It uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads
- PostgreSQL evolved from the Ingres project at the University of California, Berkeley
- It is supported on all major operating systems, including Windows, Linux, macOS, FreeBSD, and OpenBSD

PostgreSQL

PostgreSQL supports the following data types

- Primitives: Integer, Numeric, String, Boolean
- Structured: Date/Time, Array, Range / Multirange, UUID
- Document: JSON/JSONB, XML, Key-value (Hstore)
- Geometry: Point, Line, Circle, Polygon
- Customizations: Composite, Custom Types

PostgreSQL

PostgreSQL Numeric data types

- Integer
  - smallint (2 bytes): For small-range integers
  - integer (4 bytes): For typical integer values.
  - bigint (8 bytes): For very large integers

PostgreSQL

PostgreSQL Numeric data types

- Floating-Point
    - real (4 bytes): Single precision floating point
    - double precision (8 bytes): Double precision floating point

PostgreSQL

PostgreSQL Character data types

- Fixed-Length:
    - char(n): Fixed-length character string, padded with spaces if necessary
- Variable-Length:
    - varchar(n): Variable-length string with a limit
    - text: Variable-length string with no specific length limit, ideal for storing large text data

PostgreSQL

PostgreSQL Character data types

- Date/Time:
    - date: Stores a calendar date (year, month, day).
    - time [(p)]: Stores a time of day (without time zone).
    - timestamp [(p)] [without time zone]: Stores a date and time.
    - timestamp with time zone: Stores a date and time with time zone information

PostgreSQL

PostgreSQL Character data types

- Boolean:
  - boolean: Represents logical values (true, false, and NULL)

To use postgreSQL on your ubuntu system

1. Install posgreSQL application (if it is not already installed)
2. Open terminal application
3. Type psql
    a. Runs psql as the current system user
4. Type /l
    a. To list all databases available
5. Type CREATE DATABASE testdb
6. /c testdb

Listing Tables in Database

- /dt
- All tables in the current database will be listed

Creating Tables

-
  ```
  CREATE TABLE table_name(
     column1 datatype,
     column2 datatype,
     column3 datatype,
     .....
     columnN datatype,
     PRIMARY KEY( one or more columns )
  );
  ```

Creating Tables - Example

```
CREATE TABLE  department (
    dept_id char(4) NOT NULL,
    dept_name varchar(30),
    PRIMARY KEY( dept_id)
);
```

dept_id	dept_name
D001	Information Technology
D002	Mathematics
D003	Statistics

View the Structure of Table

- \d department

Inserting Rows into Table

- INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)
  VALUES (value1, value2, value3,...valueN);

Inserting Rows into Table - Example

- INSERT INTO department (dept_id, dept_name) VALUES ('D001', 'Information Technology');

- Note that the we can insert even empty data for
- dept_name! This is due to the fact that we did not
- specify that dept_name can not be blank!
- Such restrictions are called as constraints
- Constraints can be added at the time of creating
  a table or later

dept_id	dept_name
D001	Information Technology
D002	Mathematics
D003	Statistics

Constraints

- Constraints are rules applied to table columns in PostgreSQL to ensure data integrity and consistency
- They restrict the types of data that can be inserted into a table, ensuring that the stored data adheres to business rules and relational integrity

Common Types of Constraints in PostgreSQL

- NOT NULL Constraint:
  - Purpose: Ensures that a column cannot have a NULL value
- UNIQUE Constraint:
  - Purpose: Ensures that all values in a column (or a group of columns) are distinct
- PRIMARY KEY Constraint:
  - Purpose: Uniquely identifies each record in a table. It inherently combines a NOT NULL and UNIQUE constraint

Common Types of Constraints in PostgreSQL

- FOREIGN KEY Constraint:
  - Purpose: Enforces a link between data in two tables by referencing the primary key of another table. This maintains referential integrity

Adding Constraints Along with Table Creation

```
CREATE TABLE departments (
    department_id PRIMARY KEY,
    department_name VARCHAR(30) NOT NULL UNIQUE);
```

- Constraints are added along with column definitions
- Here we have added two constraints - every row should have a unique not null value for the department name!
- Since we have already defined the table, we have to use the ALTER TABLE command to add the constraints!

Adding Constraints for Existing Tables

- Add the NOT NULL Constraint
  - ALTER TABLE tablename MODIFY coulmn_name NOT NULL;
  - ALTER TABLE department MODIFY dept_name NOT NULL; (Example)

- Add the UNIQUE constraint
  - ALTER TABLE tablename ADD CONSTRAINT nameofconstraint UNIQUE (column name)

  - ALTER TABLE Department ADD CONSTRAINT unique_department_name UNIQUE (dept_name)(Example)

Retrieving Data from Tables

- Get a list of all rows in the table, listing all columns
  - SELECT * FROM table_name;

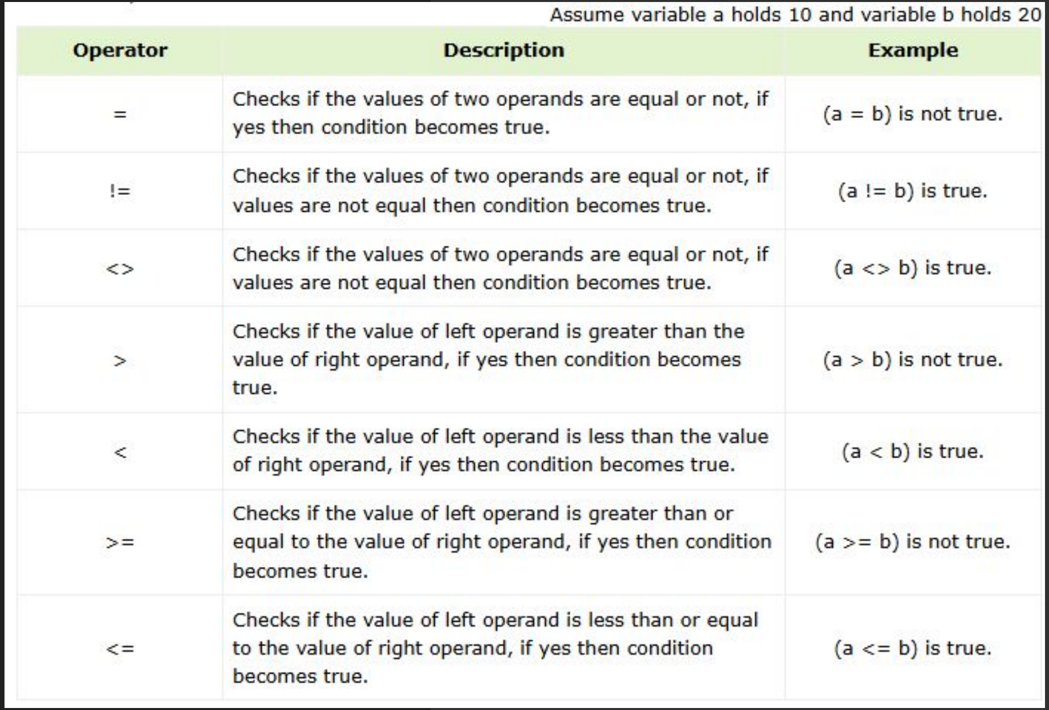
- Get a list of all rows in the table, listing only the specified columns
  - SELECT column1, column2, columnN FROM table_name;

- Get a list of all rows in the table, listing in a sorted order
  - SELECT column1, column2, columnN FROM table_name ORDER By column1;

Retrieving Data from Tables

- Get a list of rows in the table, matching some specified condition
  - SELECT column1, column2, columnN FROM table_name WHERE expression
  - An expression is formed using comparison and logical operators
  - Expression is evaluated as a boolean value - True or False
  - Only rows for which the Boolean expression is true are returned

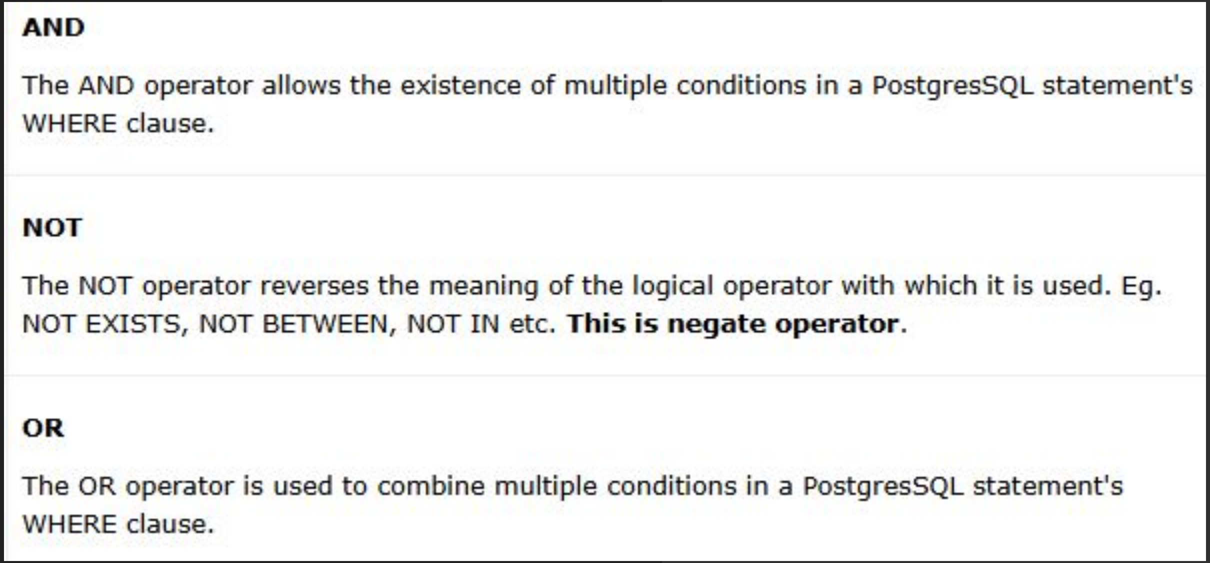
Retrieving Data from Tables - Comparison Operators



Assume variable a holds 10 and variable b holds 20

Operator	Description	Example
=	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(a = b) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a != b) is true.
<>	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a <> b) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(a >= b) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(a <= b) is true.

Retrieving Data from Tables - logical Operators



AND

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

NOT

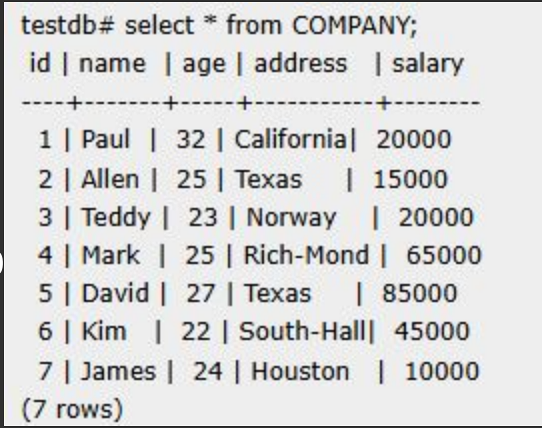
The NOT operator reverses the meaning of the logical operator with which it is used. Eg. NOT EXISTS, NOT BETWEEN, NOT IN etc. **This is negate operator**.

OR

The OR operator is used to combine multiple conditions in a PostgresSQL statement's WHERE clause.

Retrieving Data from Tables - WHERE clause - Examples

- Consider the table COMPANY with the following rows
- To list all the records where AGE is greater than or equal to 25 and salary is greater than or equal to 65000.00
    - SELECT * FROM COMPANY WHERE AGE >= 25 AND SALARY >= 65000;
- To list all the records where AGE is greater than or equal to 25 OR salary is greater than or equal to 65000.00
    - SELECT * FROM COMPANY WHERE AGE >= 25 OR SALARY >= 65000
- To list all the records where AGE is not NULL,
    - SELECT * FROM COMPANY WHERE AGE IS NOT NULL



```
testdb# select * from COMPANY;
 id | name  | age | address    | salary
----+-------+-----+------------+--------
  1 | Paul  |  32 | California|  20000
  2 | Allen |  25 | Texas      |  15000
  3 | Teddy |  23 | Norway     |  20000
  4 | Mark  |  25 | Rich-Mond  |  65000
  5 | David |  27 | Texas      |  85000
  6 | Kim   |  22 | South-Hall|  45000
  7 | James |  24 | Houston    |  10000
(7 rows)
```

Retrieving Data from Tables - WHERE clause - LIKE operator

- LIKE operator is used to match text values against a pattern using wildcards
- There are two wildcards used in conjunction with the LIKE operator –
  - The percent sign (%)
    - represents zero, one, or multiple numbers or characters
  - The underscore (_)
    - represents a single number or character
  - If either of these two signs is not used in conjunction with the LIKE clause, then the LIKE acts like the equals operator

- Example
  - SELECT dept_name FROM department WHERE dept_name LIKE 'Info%'
    - List all department names who’s name starts with Info
  - SELECT dept_name FROM department WHERE dept_name LIKE '__t%’
    - List all department names who’s name having its third letter as t
  - SELECT dept_id FROM department WHERE dept_id LIKE 'T___’
    - List all departmentIDs having its first letter as T
  - WHERE SALARY::text LIKE '2_%_%'
    - Finds any values that start with 2 and are at least 3 characters in length. Here salary is a numeric data; it should be typecasted to text as shown above (using ::text operator)

Creating Tables with Relations

-
  ```
  CREATE TABLE table_name(
     column1 datatype,
     column2 datatype,
     column3 datatype references table2(primary key of table2)
     .....
     columnN datatype,
     PRIMARY KEY( one or more columns )
  );
  ```

Joining Tables

- Queries can access multiple tables at once, or access the same table in such a way that multiple rows of the table are being processed at the same time
- Queries that access multiple tables (or multiple instances of the same table) at one time are called join queries
- They combine rows from one table with rows from a second table, with an expression specifying which rows are to be paired
- For example, to return all the weather records together with the location of the associated city, the database needs to compare the city column of each row of the weather table with the name column of all rows in the cities table, and select the pairs of rows where these values match

Joining Tables

- Queries can access multiple tables at once, or access the same table in such a way that multiple rows of the table are being processed at the same time
- Queries that access multiple tables (or multiple instances of the same table) at one time are called join queries
- They combine rows from one table with rows from a second table, with an expression specifying which rows are to be paired
- For example, to return all the rows in the teachers table with the names of the department belong to them, the database needs to compare the dept_id column of each row of the teacher table with the dept_id column of all rows in the department table, and select the pairs of rows where these values match

Joining Tables - Examples

- Retrieve all rows from courses table listing the department names of each faculty member
  - SELECT teachers.teach_id, teachers.teach_name, teachers.dept_id, department.dept_name FROM teachers, department WHERE teachers.dept_id = department.dept_id

- Retrieve all rows from courses table listing the department names of each faculty member, sorted by the department name
  - SELECT teachers.teach_id, teachers.teach_name, teachers.dept_id, department.dept_name FROM teachers, department WHERE teachers.dept_id = department.dept_id ORDER BY department.dept_name

Joining Tables - Examples

- Retrieve all rows from teachers table listing the department names of each faculty member, sorted by the department name
  - SELECT teachers.teach_id, teachers.teach_name, teachers.dept_id, department.dept_name FROM teachers, department WHERE teachers.dept_id = department.dept_id ORDER BY department.dept_name

- Retrieve all rows from teachers table listing the department names of each faculty member, listing only teachers who’s name starts with F
  - SELECT teachers.teach_id, teachers.teach_name, teachers.dept_id, department.dept_name FROM teachers, department WHERE teachers.dept_id = department.dept_id AND teachers.teach_name LIKE 'f%'

DELETING ROWS FROM TABLE

- All rows or only rows that match certain condition can be deleted
- DELETE command is used for this purpose

- To delete all rows in a table,
  - DELETE FROM table
    - Even if all rows are deleted, table will remain in the database without any rows!
    - Example:- DELETE FROM teachers
- To selectively delete rows in a table,
  - DELETE FROM table WHERE exp

DELETING ROWS FROM TABLE

- When we delete rows from tables involved in a primary key foreign key relationship, we can not delete rows from the parent table that are referenced by a child table
- Additional options are to be used to achieve this in a single attempt

- Example:- Consider department and teachers tables. Teachers is referring department

dept_id	dept_name
D001	Information Technology
D002	Mathematics
D003	Statistics

teach_id	teach_name	dept_id
T001	Mohammed	D001
T002	James	D002
T003	Ramkumar	D003
T004	Vivek	D001

dept_id	dept_name
D001	Information Technology
D002	Mathematics
D003	Statistics

teach_id	teach_name	dept_id
T001	Mohammed	D001
T002	James	D002
T003	Ramkumar	D003
T004	Vivek	D001

DELETING ROWS FROM TABLE

- Example:- Consider department and teachers tables. Teachers is referring department
- If try to delete a row with dept_id as D001 from the department table, it will not be done! Because there is at least one row in teachers table that refers the row with dept_id as D001 in the department table

DELETE TABLES - DROP TABLE

- Removes tables from the database
- DROP TABLE table1, table2,...
- In this case also, attempting to drop a table that is referenced by foreign key constraints in other tables is restricted (Similar to the DELETE command)

UPDATING COLUMN VALUES ROWS

- UPDATE table SET column = value, column2 = value2 WHERE exp;
    - First, specify the name of the table that you want to update data after the UPDATE keyword
    - Second, specify columns and their new values after SET keyword. The columns that do not appear in the SET clause retain their original values
    - Third, determine which rows to update in the condition of the WHERE clause
    - The WHERE clause is optional. If you omit the WHERE clause, the UPDATE statement will update all rows in the table

UPDATING COLUMN VALUES ROWS - Example

- Consider the following Table
- CREATE TABLE courses(

  ```
  course_id serial PRIMARY KEY,
  course_name VARCHAR(255) NOT NULL,
  price DECIMAL(10,2) NOT NULL,
  description VARCHAR(500),
  published_date date);
  ```
- To update the course with id 3 by changing the published_date to '2020-08-01'
  - UPDATE courses SET published_date = '2020-08-01' WHERE course_id = 3;

UPDATING COLUMN VALUES ROWS - Example

- To update the course with id 3 by changing the published_date to '2020-08-01'
  - UPDATE courses SET published_date = '2020-08-01' WHERE course_id = 3;
- To increase the price of all the courses 5%
  - UPDATE courses SET price = price * 1.05;
  - As you see, the expression for the new value can refer to the existing value(s) in the row
  - There is no WHERE clause! If it is omitted, it means that all rows in the table are updated
- To update more than one column in an UPDATE command
  - UPDATE mytable SET a = 5, b = 3, c = 1 WHERE a > 0;

====