**Introduction to DBMS and RDBMS**

**A Database Management System (DBMS)** is a software system that is designed to manage and organize data in a structured manner. It allows users to create, modify, and query a database, as well as manage the security and access controls for that database.

**What is Database?**

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

For example: The college Database organizes the data about the admin, staff, students and faculty etc. Using the database, you can easily retrieve, insert, and delete the information.

**DBMS allows users the following tasks:** **Data Definition:** It is used for creation, modification, and removal of definition that defines the organization of data in the database. **Data Updation:** It is used for the insertion, modification, and deletion of the actual data in the database. **Data Retrieval:** It is used to retrieve the data from the database which can be used by applications for various purposes. **User Administration:** It is used for registering and monitoring users, maintain data integrity, enforcing data security, dealing with concurrency control, monitoring performance and recovering information corrupted by unexpected failure.

**Types of DBMS**

There are four main types of DBMS: relational, object-oriented, graph-based, and NoSQL.

* **Relational DBMSs** are the most common and use a tabular structure to store data.
* **Object-oriented DBMSs** use an object-oriented model to store data.
* **Graph-based DBMSs** use a graph structure to store data.
* **NoSQL DBMSs** are a newer DBMS type that uses a non-tabular structure to store data.

## Advantages of DBMS

## Data Integrity:

## Definition: Data integrity ensures the accuracy and reliability of data stored in the database. Example: Scenario: In a university database, the student's ID is set as the primary key. Advantage: Ensures that each student is uniquely identified, preventing duplicate records

## Data Security: Definition: Data security involves protecting data from unauthorized access and ensuring confidentiality. Example: Scenario: Access controls are implemented in a healthcare database to restrict nurses from accessing patient billing information. Advantage: Ensures that only authorized personnel can view or modify sensitive patient data.

1. **Data Organization: Definition:** Data organization refers to the structured arrangement of data elements within a database. **Example:** Scenario: In a library database, books are organized into tables with attributes such as title, author, and publication date. Advantage: Facilitates efficient data retrieval and management through a well-organized structure.
2. **Data Sharing: Definition:** Data sharing allows multiple users or applications to access and use the same data. **Example:** Scenario: An inventory management system allows both the sales and warehouse departments to access real-time stock levels. Advantage: Enables collaboration and ensures that all departments have access to up-to-date information.
3. **Data Redundancy: Definition:** Data redundancy refers to the unnecessary repetition of data within a database. **Example:** Scenario: In a customer database, customer addresses are stored separately from order records to avoid duplication. Advantage: Reduces storage space and minimizes the risk of inconsistent or conflicting information.
4. **Data Consistency: Definition:** Data consistency ensures that data remains accurate and unaltered throughout the database. **Example:** Scenario: In a banking system, when transferring money between accounts, the transaction deducts the correct amount from the sender and credits the same amount to the recipient. Advantage: Maintains accuracy and reliability of financial data across the entire system.
5. **Cost Reduction: Definition:** Cost reduction involves minimizing expenses related to data storage, maintenance, and access. **Example:** Scenario: An e-commerce company uses a cloud-based database, allowing them to scale storage dynamically based on demand. Advantage: Reduces the need for large upfront investments in physical infrastructure and provides cost-effective scalability.

**Disadvantages of DBMS**

**1. Complexity: Definition:** Complexity refers to the intricacy and difficulty associated with the installation, configuration, and maintenance of a Database Management System (DBMS).

**Example:** Scenario*:* Configuring a complex enterprise-level DBMS with multiple instances, replication, and distributed databases.

Disadvantage*:* Requires highly skilled personnel and extensive training, increasing the overall complexity.

**2. Performance Overhead:**

**Definition:** Performance overhead refers to the additional processing time and system resources required to run and manage a DBMS.

**Example:**

Scenario*:* Running complex queries involving multiple joins and aggregations on a large database.

Disadvantage*:* Intensive queries may impact overall system performance, leading to slower response times.

**3. Scalability:**

**Definition:** Scalability refers to the system's ability to handle increased data volumes, user loads, or transaction rates.

**Example:**

Scenario*:* A rapidly growing e-commerce platform experiences increased traffic and data storage needs.

Disadvantage*:* Scaling a database system to handle the growing demands may be challenging and require additional resources.

**4. Cost:**

**Definition:** Cost involves the expenses associated with acquiring, implementing, and maintaining a Database Management System.

**Example:**

Scenario*:* Purchasing licenses for a commercial DBMS like Oracle or Microsoft SQL Server.

Disadvantage*:* Incurs substantial upfront and ongoing costs, which might be a significant factor for small businesses.

**5. Limited Use Cases:**

**Definition:** Limited use cases imply that certain types of applications or data structures might not be well-suited for a particular DBMS.

**Example:**

Scenario*:* Choosing a relational database for a project with highly interconnected and hierarchical data.

Disadvantage*:* Non-relational databases might be more suitable for specific use cases, such as graph databases for highly interconnected data.

**6. Damaged Part:**

**Definition:** Refers to the potential vulnerability or weakness in a DBMS that might lead to system failures, data corruption, or security breaches.

**Example:**

Scenario*:* A database server experiencing hardware failures, resulting in data loss or system downtime.

Disadvantage*:* Vulnerabilities or weaknesses may expose the system to various risks, affecting data integrity and availability.

**RDBMS(**Relational Database Management System**)**

**Relational Database Management System** (**RDBMS)** is a type of  Database Management System (DBMS) that stores data in a row-based table structure which connects related data elements. An RDBMS includes functions that maintain the security, accuracy, integrity and consistency of the data

RDBMS is the basis for all modern database systems such as MySQL, Microsoft SQL Server, Oracle, and Microsoft Access. It uses SQL queries to access the data in the database. It contains several tables, and each table has its primary key. Due to a collection of an organized set of tables, data can be accessed easily in RDBMS.

**Key RDBMS Concepts:**

* **Tables:** RDBMSs store data in tables, which are like spreadsheets with rows and columns. Each row represents a record, while each column represents an attribute (characteristic) of that record.
* **Columns (Attributes):** These define the specific characteristics associated with each record in a table. For example, in a customer table, columns might include "Customer ID," "Name," "Address," "Email," and "Phone Number."
* **Rows (Records):** These represent individual instances of data within a table, corresponding to specific entities. For example, each row in a customer table might represent a particular customer, containing their unique ID and other details.
* **Keys:**
  + **Primary Key:** A unique identifier that distinguishes each record within a table. It ensures no duplicate records exist.
  + **Foreign Key:** A column in one table that references the primary key of another table, establishing a link between them. This enables the creation of relationships between tables.

**Normalization Techniques:**

Normalization is a process of organizing data in tables to minimize data redundancy and improve data integrity. By following normalization rules, you can reduce the risk of data inconsistency and simplify data manipulation.

Here are some key normalization forms:

* **First Normal Form (1NF):**
  + Eliminates duplicate rows within a table.
  + Ensures each column value represents a single atomic (indivisible) attribute.
* **Second Normal Form (2NF):**
  + Satisfies all 1NF requirements.
  + Eliminates partial dependencies. A partial dependency occurs when a non-key attribute depends only on a part of the primary key, not the entire key.
* **Third Normal Form (3NF):**
  + Satisfies all 2NF requirements.
  + Eliminates transitive dependencies. A transitive dependency occurs when a non-key attribute depends on another non-key attribute, which in turn depends on the primary key.
* **Boyce-Codd Normal Form (BCNF):**
  + Satisfies all 3NF requirements.
  + Eliminates determinant dependencies. A determinant dependency occurs when a non-key attribute determines another non-key attribute, even though it may not directly depend on the primary key.

**Choosing the Right Normalization Form:**

The appropriate level of normalization depends on the specific needs and complexity of your database. While higher forms offer greater data integrity, they can also increase database complexity. Consider striking a balance between data integrity, performance, and maintainability when choosing a normalization form.

**Advantages of RDBMS:**

1. **Data Integrity:** **Example:** In a student database, the use of primary keys ensures that each student record is unique, maintaining data integrity.
2. **Structured Data Storage:** **Example:** Organizing data into tables with predefined relationships facilitates efficient storage and retrieval.
3. **Data Independence:** **Example:** Modifying the database schema does not impact applications, promoting data independence.
4. **Efficient Query Processing:** **Example:** RDBMS uses query optimization techniques for faster and more efficient data retrieval.
5. **Relationships Between Tables:** **Example:** Establishing relationships between tables supports complex queries and normalization.
6. **Multi-User Support:** **Example:** Concurrent access to a banking database allows multiple users to perform transactions simultaneously.
7. **Security Features: Example:** User authentication and authorization mechanisms secure the database against unauthorized access.
8. **ACID Properties:** **Example:** Ensures atomicity, consistency, isolation, and durability in transactions, maintaining data integrity.
9. **Scalability:** **Example:** RDBMS systems can scale horizontally or vertically to handle growing data volumes.
10. **Standard Query Language (SQL):** **Example:** The standardized SQL language simplifies data retrieval and manipulation.

**Disadvantages of RDBMS:**

1. **Complexity in Design:** **Example:** Designing a normalized database with complex relationships can be challenging and requires expertise.
2. **Performance Overhead:** **Example:** Joining multiple tables in complex queries can lead to performance issues, especially in large databases.
3. **Scalability Challenges:** **Example:** Scaling horizontally (adding more servers) can be complex, and not all RDBMS solutions scale equally well.
4. **Cost:** **Example:** Acquiring licenses for commercial RDBMS solutions may incur significant costs for organizations.
5. **Limited Performance for Certain Types of Applications:** **Example:** For certain types of applications (e.g., graph databases, hierarchical data structures), RDBMS may not be the most performant solution.
6. **Concurrency Control Overhead: Example:** Implementing concurrency control mechanisms to manage simultaneous user access can add overhead to the system.
7. **Fixed Schema: Example:** RDBMS requires a predefined schema, making it less flexible for applications with evolving data structures.
8. **High Overhead: Example:** They typically have more overhead compared to some NoSQL databases. The need for data normalization, transactions, and referential integrity enforcement can result in increased storage requirements and slower performance.

**DBMS vs RDBMS: Difference**

|  |  |
| --- | --- |
| **DBMS** | **RDBMS** |
| Data is stored in a database management system (DBMS) as a file | Tables are used to store information |
| Data is stored in a database management system (DBMS) in either a navigational or hierarchical format | RDBMS employs a tabular format, with column names as headers and associated data as rows |
| Only a single user is supported by the DBMS | It may be used by numerous people |
| The data in a typical database may not be stored according to the ACID model | Relational databases are more difficult to create, but they are more consistent and organised  They follow the rules of ACID (Atomicity, Consistency, Isolation, Durability) |
| It is an application that is used to manage databases over computer networks as well as the system hard drives | The database systems are used to keep track of the relationships between the tables |
| Normalization is not supported by DBMS. | A relational database management system (RDBMS) can be normalised. |
| The DBMS system is mostly used to manage tiny amounts of data | The RDBMS database is built to manage a vast volume of data |
| Client-server architecture is not supported by DBMS | Client-server architecture is supported by RDBMS |
| For complicated and vast amounts of data, data retrieval takes longer | Because of its relational methodology, data retrieval is quick |
| There is no correlation between the data | Data is kept in the form of tables that are linked together via foreign keys |
| There is no sense of safety | Multiple security levels are available. At the OS, command, and object levels, log files are produced |
| Individual data items must be accessed | SQL queries make it simple to retrieve data  At the same time, many data items can be accessed |
| A file system, XML, the Windows Registry, and other DBMS are examples | MySQL, Oracle, SQL Server, and other RDBMS are examples |

**SQL (Structured Query Language)**

## SQL stands for Structured Query Language. It is used for storing and managing data in Relational Database Management System (RDBMS).

## It is a standard language for Relational Database System. It enables a user to create, read, update and delete relational databases and tables.

## All the RDBMS like MySQL, Informix, Oracle, MS Access and SQL Server use SQL as their standard database language.

## SQL follows the following rules:

## Structure query language is not case sensitive. Generally, keywords of SQL are written in uppercase.

## Statements of SQL are dependent on text lines. We can use a single SQL statement on one or multiple text line.

## Using the SQL statements, you can perform most of the actions in a database. SQL depends on tuple relational calculus and relational algebra.

## Advantages of SQL

## • High speed

## • No coding needed

## • Well defined standards

## • Portability

## • Interactive language

## • Multiple data view

## SQL Commands

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

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

## Types of SQL Commands

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

## Data Definition Language (DDL) Commands:

## 1. CREATE:

## The CREATE TABLE statement is used to create a new table in a database. Syntax :

## CREATE TABLE table\_name (

## column1 datatype,

## column2 datatype,

## column3 datatype,

## ....);

## Example:

## CREATE TABLE employee (

## PersonID int,

## LastName varchar(255),

## FirstName varchar(255),

## Email varchar(255),

## City varchar(255)

## );

## 2. ALTER:

## Modifies the structure of existing database objects.

## Examples:

## Add Column to Table:

## ALTER TABLE Persons ADD COLUMN emp\_department VARCHAR(50);

## Modify Column Data Type:

## ALTER TABLE employees MODIFY COLUMN emp\_salary DECIMAL(12, 2);

## 3. DROP:

## Deletes existing database objects like tables, views, indexes, etc.

## Examples:

## DROP TABLE employee;

## 4. TRUNCATE:

## Removes all records from a table, but retains the table structure for future use.

## Example:

## TRUNCATE TABLE employees;

## Data Manipulation Language (DDM) Commands:

## DML commands are used to modify the database. It is responsible for all form of CHANGES in the database.

## The command of DML is not auto-committed that means it can't permanently save all the changes in the database.

## Here are some commands that come under DML:

## INSERT

## UPDATE

## DELETE

## INSERT

## The INSERT statement is a SQL query. It is used to insert data into the row of a table.

## Syntax:

## INSERT INTO TABLE\_NAME VALUES (value1, value2, value3, .... valueN);

## Example:

## INSERT INTO employee (FirstName, LastName, email, City) VALUES ('saleeja', 's', 'saleej@gmail', 'kollam');

## UPDATE

## This command is used to update or modify the value of a column in the table.

## Syntax:

## UPDATE table\_name SET [column\_name1= value1,...column\_nameN = valueN] [WHERE CONDITION]

## Example:

## UPDATE employee SET User\_Name ='saleeja' WHERE employee\_Id = '3'

## Data Query Language(DQL) Commands

## DQL is used to fetch the data from the database.

## It uses only one command:

## SELECT

## It is used to select the attribute based on the condition described by WHERE clause.

## Syntax:

## SELECT expressions FROM TABLES WHERE conditions;

## Example:

## SELECT emp\_name FROM employee WHERE age > 20;

## SQL Logical Operators

|  |  |
| --- | --- |
| **Operator** | **Description** |
| ALL | TRUE if all of the subquery values meet the condition |
| AND | TRUE if all the conditions separated by AND is TRUE |
| ANY | TRUE if any of the subquery values meet the condition |
| BETWEEN | TRUE if the operand is within the range of comparisons |
| EXISTS | TRUE if the subquery returns one or more records |
| IN | TRUE if the operand is equal to one of a list of expressions |
| LIKE | TRUE if the operand matches a pattern |
| NOT | Displays a record if the condition(s) is NOT TRUE |
| OR | TRUE if any of the conditions separated by OR is TRUE |
| SOME | TRUE if any of the subquery values meet the condition |

## AVG FUNCTION

## The AVG function is used to calculate the average value of the numeric type. AVG function returns the average of all non-Null values.

## Syntax:

## AVG() or AVG( [ALL|DISTINCT] expression )

## Example

## SELECT AVG(COST) FROM PRODUCT\_MAST;

## MAX FUNCTION

## MAX function is used to find the maximum value of a certain column. This

## function determines the largest value of all selected values of a column.

## Syntax:

## MAX() or MAX( [ALL|DISTINCT] expression )

## Example:

## SELECT MAX(RATE) FROM PRODUCT\_MAST;

## MIN FUNCTION

## MIN function is used to find the minimum value of a certain column. This

## function determines the smallest value of all selected values of a column

## Syntax:

## MIN() or MIN( [ALL|DISTINCT] expression )

## Example:

## SELECT MIN(RATE) FROM PRODUCT\_MAST;

## SQL JOIN

## SQL, JOIN means "to combine two or more tables". In SQL, JOIN clause is used to combine the records from two or more tables in a database.

## Types of SQL JOIN

## INNER JOIN

## LEFT JOIN

## RIGHT JOIN

## FULL JOIN

## INNER JOIN

## In SQL, INNER JOIN selects records that have matching values in both tables as long as the condition is satisfied. It returns the combination of all rows from both the tables where the condition satisfies.

## Syntax:

## SELECT table1.column1, table1.column2, table2.column1,....

## FROM table1

## INNER JOIN table2

## ON table1.matching\_column = table2.matching\_column;

## Example:

## SELECT EMPLOYEE.EMP\_NAME, PROJECT.DEPARTMENT

## FROM EMPLOYEE

## INNER JOIN PROJECT

## ON PROJECT.EMP\_ID = EMPLOYEE.EMP\_ID;

## LEFT JOIN

## The SQL left join returns all the values from left table and the matching values

## from the right table. If there is no matching join value, it will return NULL. Syntax:

## SELECT table1.column1, table1.column2, table2.column1,....

## FROM table1

## LEFT JOIN table2

## ON table1.matching\_column = table2.matching\_column;

## Example:

## SELECT EMPLOYEE.EMP\_NAME, PROJECT.DEPARTMENT

## FROM EMPLOYEE

## LEFT JOIN PROJECT

## ON PROJECT.EMP\_ID = EMPLOYEE.EMP\_ID;

## RIGHT JOIN

## In SQL, RIGHT JOIN returns all the values from the values from the rows of right

## table and the matched values from the left table. If there is no matching in both tables, it will return NULL.

## Syntax:

## SELECT table1.column1, table1.column2, table2.column1,....

## FROM table1

## RIGHT JOIN table2

## ON table1.matching\_column = table2.matching\_column;

## Example:

## SELECT EMPLOYEE.EMP\_NAME, PROJECT.DEPARTMENT

## FROM EMPLOYEE

## RIGHT JOIN PROJECT

## ON PROJECT.EMP\_ID = EMPLOYEE.EMP\_ID;

## FULL JOIN

## In SQL, FULL JOIN is the result of a combination of both left and right outer join.

## Join tables have all the records from both tables. It puts NULL on the place of

## matches not found.

## Syntax:

## SELECT table1.column1, table1.column2, table2.column1,....

## FROM table1

## FULL JOIN table2

## ON table1.matching\_column = table2.matching\_column;

## Example:

## SELECT EMPLOYEE.EMP\_NAME, PROJECT.DEPARTMENT

## FROM EMPLOYEE

## FULL JOIN PROJECT

## ON PROJECT.EMP\_ID = EMPLOYEE.EMP\_ID;