What is a database management system?

A database management system (DBMS) is software that allows users to create, organize, store, retrieve, manipulate, and manage data efficiently. It provides an interface for interacting with databases and enables users to perform various operations on the data.

A DBMS serves as an intermediary between the users and the database. It ensures data integrity, security, and consistency while handling multiple concurrent user requests. Some common features of a DBMS include:

Data definition: The DBMS allows users to define the structure of the database, specifying data types, relationships between entities, and constraints.

Data manipulation: It enables users to insert, update, delete, and retrieve data from the database using structured query languages (SQL) or other programming interfaces.

Data storage and organization: The DBMS manages the physical storage of data on disks or other storage media, determining how data is stored, indexed, and accessed for optimal performance.

Data security: DBMS provides mechanisms to control access to the database, enforcing user authentication, authorization, and data encryption to protect sensitive information.

Data concurrency and recovery: It handles concurrent access to the database by multiple users, ensuring that transactions are processed reliably and in a controlled manner. It also provides mechanisms for data backup and recovery in case of system failures or errors.

Data integrity and consistency: The DBMS enforces data integrity constraints, such as primary key and foreign key constraints, to maintain the consistency and accuracy of the stored data.

Data scalability and performance optimization: A DBMS allows for scaling the database as data volume increases, and it includes features to optimize query execution and enhance system performance.

Examples of popular DBMSs include Oracle Database, Microsoft SQL Server, MySQL, PostgreSQL, MongoDB, and SQLite. Each DBMS may have its own strengths, limitations, and specialized features, catering to different application requirement

Basics of MySQL:

1.Creating a database- To create a database use the following command:

```
CREATE DATABASE database_name;
```

2.Constructing tables under the database:

Use the CREATE TABLE statement to define the structure of your table. Specify the table name and define the columns along with their data types and any additional constraints.

```
CREATE TABLE employees (
        id INT PRIMARY KEY AUTO_INCREMENT,
        first_name VARCHAR(50),
        last_name VARCHAR(50),
        email VARCHAR(100),
        hire_date DATE
);
```

In the above example, we created a table with columns for id, first_name, last_name, email, and hire_date. The id column is defined as the primary key with the AUTO_INCREMENT attribute, which automatically generates a unique value for each new row.

3.Specify constraints (optional): You can add constraints to enforce data integrity. For example, to specify a column as the primary key, use the PRIMARY KEY constraint. You can also define NOT NULL constraints to ensure that certain columns always have a value.

4. Inserting data in the tables:

There are two ways to insert data into a table. Let's say we are familiar with the serial of the attributes that we have added in the table. In that case we can use:

```
INSERT INTO employees  VALUES ('val1','val2'.....)
```

However, in reality, we do not always remember the exact order at which we put the attributes. So using this command is easier:

```
INSERT INTO employees(id, first_name,last_name,email,hire_date) VALUES ('001','Stephen', 'Davis', 'sth@gmail.com', '20/1/12')
```

Note: You can also choose the attributes which you want to populate, so just select those attributes that you want to insert into the table!

5. Deleting an attribute from the table which meets a certain condition:

```
DELETE FROM employees

WHERE id= 001
```

Deleting an attribute(email for this example) from the table:

```
ALTER TABLE employees

DROP COLUMN email;
```

6. Query Design:

The basic query design is:

```
SELECT attribute_name
FROM table_name
WHERE condition (if there is some condition)
```

The query above is manipulated as per need.

1.Let's say we want to see all the employees' IDs.

```
SELECT id
FROM employees
```

2. If we want to view all rows we have to use the * command which is:

```
SELECT *
FROM employees
```

This query fetches the whole table employees.

3. If we want to view the last_name of the employee with ID=001, this command is used:

```
SELECT last_name
FROM employees
```

```
WHERE id= 001
```

Now, we also have to remember what the data type of attribute is, if it was inserted as text, then we have to write the above command like this:

```
 SELECT last_name
FROM employees
WHERE id= '001'
```

4. Multiple Conditions:

In case of multiple conditions, we have to understand the query first (i.e. does it require that both conditions are met or only one of them?)

a. Query: Find the employee ID of those employees whose last name is Rahman and were hired on 5/5/16. *(In this query, we have to fulfill both conditions so we should use AND)*

```
 SELECT id
FROM employees
WHERE last_name= 'Rahman' AND hire_date= '05/05/16'
```

b. Query: Find the employee ID of those employees whose last name is Rahman or Ahmed *(In this query, we have to fulfill only one condition so we should use OR)*

```
SELECT id
FROM employees
WHERE last_name= 'Rahman' OR last_name= 'Ahmed'
```

5. Some common functions:

To retrieve data sometimes you need to use some functions as per the query.
For example, we want to count the number of employees,the command will be:

```
SELECT COUNT(id)
FROM employees
```

There are other functions we can use as well such as AVG, SUM, MAX, MIN, etc.

6. Operator Meaning

- = Equal to

- > Greater than
- >= Greater than or equal to
- < Less than
- <= Less than or equal to
- <> Not equal to
- != Not equal to

Example: Find the employee id of those whose last name is not Rahman

```
SELECT id
FROM employees
WHERE last_name != 'Rahman'
```

7. Ordering:

To order the data in ascending order or descending order, we use ASC/DESC functions.

Example: Find the employee id and sort them in descending order.

```
SELECT id
FROM employees
ORDER BY id DESC
```

Note: By default, mySQL sorts data in ascending order so you do not necessarily need to add the ASC function.

7. Grouping:

If we want to group the data we want to retrieve, we have to use "GROUP BY" in the SQL command.

Example: Count the number of employees and group them as per their hire date.

```
SELECT COUNT(id)
FROM employees
GROUP BY hire_date
```