Database (DB):

A database is a structured collection of data that is organized and stored in a way that allows for efficient data retrieval, management, and manipulation. Databases are used to store and manage various types of information, ranging from simple lists to complex business data. They serve as a central repository for data, enabling multiple users or applications to access and interact with the stored information. Databases are commonly used in software applications, business operations, and various other domains.

Key characteristics of databases:

- **Structured Data**: Databases store data in a structured format, often using tables or collections to organize related information.

- **Data Integrity**: Databases enforce data integrity constraints to maintain the accuracy and consistency of data.

- **Data Security**: Access to databases can be controlled through user authentication and authorization mechanisms.

- **Concurrent Access**: Databases support concurrent access by multiple users or applications, ensuring that data remains consistent.

- **Scalability**: Databases can scale to handle large volumes of data and high numbers of transactions.

- **Data Retrieval**: Users and applications can retrieve, query, and manipulate data from databases.

Database Management System (DBMS):

A Database Management System (DBMS) is software that provides tools and services for creating, managing, and interacting with databases. The primary purpose of a DBMS is to enable efficient and secure data storage, retrieval, and manipulation. It acts as an intermediary between users or applications and the physical database, handling tasks like data storage, indexing, transaction management, and security.

Common DBMS features:

- **Data Storage**: DBMS systems store data on disk or in memory, managing how data is organized and accessed.

- **Data Retrieval**: DBMS systems provide SQL-based interfaces for retrieving and querying data.

- **Data Security**: DBMS systems offer authentication, authorization, and encryption to protect data.

- **Data Integrity**: DBMS systems enforce constraints and rules to maintain data accuracy and consistency.

- **Concurrency Control**: DBMS systems manage concurrent access to the database to prevent data corruption.

- **Backup and Recovery**: DBMS systems offer backup and restore mechanisms to protect data in case of failure.

- **Indexing**: DBMS systems create and manage indexes to improve data retrieval performance.

- **Transactions**: DBMS systems support transactions, allowing a series of operations to be executed atomically.

Types of DBMS:

1. **Relational DBMS (RDBMS)**: Data is organized in tables with rows and columns, and relationships between data are established. Examples include MySQL, Oracle Database, and Microsoft SQL Server.

2. **NoSQL DBMS**: Designed for handling unstructured or semi-structured data. Types include document-oriented (e.g., MongoDB), key-value stores (e.g., Redis), and column-family stores (e.g., Apache Cassandra).

3. **Object-Oriented DBMS (OODBMS)**: Designed to work with object-oriented programming languages and data models.

4. **Graph DBMS**: Specialized for managing graph-like data structures.

Types of SQL Commands:

Structured Query Language (SQL) is a domain-specific language for managing and querying

relational databases. SQL commands are used to interact with a database, including retrieving, inserting, updating, and deleting data. Common types of SQL commands include:

1. **Data Query Language (DQL)**:
   - `SELECT`: Retrieves data from one or more tables.
   - `FROM`: Specifies the table(s) to query.
   - `WHERE`: Filters rows based on specified conditions.
   - `GROUP BY`: Groups rows by one or more columns.
   - `HAVING`: Filters groups based on specified conditions.
   - `ORDER BY`: Sorts rows based on specified columns.

2. **Data Manipulation Language (DML)**:
   - `INSERT INTO`: Adds new rows to a table.
   - `UPDATE`: Modifies existing data in a table.
   - `DELETE`: Removes rows from a table.

3. **Data Definition Language (DDL)**:
   - `CREATE TABLE`: Defines a new table and its structure.
   - `ALTER TABLE`: Modifies an existing table's structure.
   - `DROP TABLE`: Deletes a table and its data.
   - `CREATE INDEX`: Creates an index for improving data retrieval.
   - `DROP INDEX`: Removes an index.

4. **Data Control Language (DCL)**:
   - `GRANT`: Grants privileges or permissions to users or roles.
   - `REVOKE`: Revokes previously granted privileges.

5. **Transaction Control Language (TCL)**:
   - `COMMIT`: Confirms and persists changes made within a transaction.
   - `ROLLBACK`: Reverts changes made within a transaction.
   - `SAVEPOINT`: Sets a savepoint within a transaction for later rollback.

SQL commands allow users and applications to interact with relational databases, making it possible to store, retrieve, and manage data effectively.

```
/* employee table and user table */
emp_id, emp_company_name,            emp_name, emp_salary
1,          'Innova Solutions',   'Nani'      , '55000.2'
2,          'ACS Solutions',            'Pinky', '45000.2'
3,          'HCL',                          'Srinivas', '56000.2'
4,          'TECHERA Solutions',    'Ramu', '51000.2'
5,          'IBM',                          'Raju', '58000.2'
6,          'TCS',                          'Pavan', '65000.2'


CREATING TABLE
================
CREATE TABLE user (
id int NOT NULL AUTO_INCREMENT,
name varchar(60) not null,
age int not null,
occupation varchar(60),
PRIMARY KEY(id)
)

/** ALTER TABLE **/

/* add a column */
alter table user add dob varchar(60) not null after name;

/* add multiple columns */
alter table user add dob varchar(60) not null after name , add salary VARCHAR(60) not null
after occupation;

/* modify column is used to modify the definition of the COLUMN */
alter table user modify dob VARCHAR(40) ;

/* drop a column */
alter table user drop column dob;

/* change column name */
Syntax : ALTER TABLE table_name  CHANGE COLUMN old_name new_name
column_definition  [ FIRST | AFTER column_name ]
alter table user change name user_name after age

/* Rename table */
alter table rename to users

/* delete the row data from a table */
delete user where id in (4,6)
```

```
/* delete entire table data or TRUNCATE table data without deleting columns */
TRUNCATE table user;

/* drop table or delete table with columns */
drop table user;

/* drop table or delete table with columns */
drop table IF EXISTS user, employee, customer ;

/* show table STRUCTURE */
desc user;

/* show columns from table from any database */

syntax : SHOW COLUMNS FROM database_name.table_name;
```

MySQL CONSTRAINTS

==========

The constraint in MySQL is used to specify the rule that allows or restricts what values/data will be stored in the table.

They provide a suitable method to ensure data accuracy and integrity inside the table.

It also helps to limit the type of data that will be inserted inside the table. 4

If any interruption occurs between the constraint and data action, the action is failed.

Types of MySQL Constraints

Constraints in MySQL is classified into two types:

i) Column Level Constraints: These constraints are applied only to the single column that limits the type of particular column data.

ii) Table Level Constraints: These constraints are applied to the entire table that limits the type of data for the whole table.

1) NOT NULL Constraint

This constraint specifies that the column cannot have NULL or empty values. The below statement creates a table with NOT NULL constraint.

mysql> CREATE TABLE Student(Id INTEGER, LastName TEXT NOT NULL, FirstName TEXT NOT NULL, City VARCHAR(35));

2) UNIQUE Constraint

mysql> CREATE TABLE ShirtBrands(Id INTEGER, BrandName VARCHAR(40) UNIQUE,

Size VARCHAR(30));

mysql> INSERT INTO ShirtBrands(Id, BrandName, Size) VALUES(1, 'Pantaloons', 38), (2, 'Cantabil', 40);

mysql> INSERT INTO ShirtBrands(Id, BrandName, Size) VALUES(1, 'Raymond', 38), (2, 'Cantabil', 40); // gives an error in 'Cantabil' value(duplicate entry for key)

3) CHECK Constraint

It controls the value in a particular column. It ensures that the inserted value in a column must be satisfied with the given condition.

In other words, it determines whether the value associated with the column is valid or not with the given condition.

The CHECK constraint ensures that the inserted value in a column must be satisfied with the given condition means the Age of a person should be greater than or equal to 18.

```
mysql> CREATE TABLE Persons (
ID int NOT NULL,
Name varchar(45) NOT NULL,
Age int CHECK (Age>=18)
);
```

mysql> INSERT INTO Persons(Id, Name, Age) VALUES (1,'Robert', 28), (2, 'Joseph', 35), (3, 'Peter', 40);

mysql> INSERT INTO Persons(Id, Name, Age) VALUES (1,'Robert', 15); // gives an error check_constraint 'persons_chk_1' is violated (because of value '15')

4) DEFAULT Constraint

This constraint is used to set the default value for the particular column where we have not specified any value.

It means the column must contain a value, including NULL.

For example, the following statement creates a table "Persons" that contains DEFAULT constraint on the "City" column.

If we have not specified any value to the City column, it inserts the default value:

```
mysql> CREATE TABLE Persons (
ID int NOT NULL,
Name varchar(45) NOT NULL,
Age int,
City varchar(25) DEFAULT 'New York'
```

```
);
```

mysql> INSERT INTO Persons(Id, Name, Age, City) VALUES (1,'Robert', 15, 'Florida'), (2, 'Joseph', 35, 'California'),(3, 'Peter', 40, 'Alaska');

mysql> INSERT INTO Persons(Id, Name, Age) VALUES (1,'Brayan', 15); // after 15 city was not given, it will take 'New_York' as DEFAULT value.

5) AUTO_INCREMENT Constraint

This constraint automatically generates a unique number whenever we insert a new record into the table.

Generally, we use this constraint for the primary key field in a table.

```
mysql> CREATE TABLE Animals(
id int NOT NULL AUTO_INCREMENT,
name CHAR(30) NOT NULL,
PRIMARY KEY (id)
);
```

6) ENUM Constraint

The ENUM data type in MySQL is a string object. It allows us to limit the value chosen from a list of permitted values in the column specification at the time of table creation.

It is short for enumeration, which means that each column may have one of the specified possible values.

It uses numeric indexes (1, 2, 3â ¦) to represent string values.

```
mysql> CREATE TABLE Shirts (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(35),
  size ENUM('small', 'medium', 'large', 'x-large')
);
```

mysql> INSERT INTO Shirts(id, name, size) VALUES (1,'t-shirt', 'medium'),(2, 'casual-shirt', 'small'),(3, 'formal-shirt', 'large');

7) INDEX Constraint

This constraint allows us to create and retrieve values from the table very quickly and easily.

An index can be created using one or more than one column. It assigns a ROWID for each row in that way they were inserted into the table.

```
mysql> CREATE TABLE Shirts (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(35),
```

```
    size ENUM('small', 'medium', 'large', 'x-large')
        );
```

```
        mysql> INSERT INTO Shirts(id, name, size) VALUES (1,'t-shirt', 'medium'), (2, 'casual-shirt', 'small'), (3, 'formal-shirt', 'large');

        mysql> CREATE INDEX idx_name ON Shirts(name);  // creating index query
        mysql> SELECT * FROM Shirts USE INDEX(idx_name);
```

8) Foreign Key Constraint

This constraint is used to link two tables together. It is also known as the referencing key.

A foreign key column matches the primary key field of another table. It means a foreign key field in one table refers to the primary key field of another table.

Let us consider the structure of these tables: Persons and Orders.

persons table :

```
        CREATE TABLE Persons (
    Person_ID int NOT NULL PRIMARY KEY,
    Name varchar(45) NOT NULL,
    Age int,
    City varchar(25)
        );
```

Orders table :

```
        CREATE TABLE Orders (
    Order_ID int NOT NULL PRIMARY KEY,
    Order_Num int NOT NULL,
    Person_ID int,
    FOREIGN KEY (Person_ID) REFERENCES Persons(Person_ID)
        );
```

In the above table structures, we can see that the "Person_ID" field in the "Orders" table points to the "Person_ID" field in the "Persons" table.

The "Person_ID" is the PRIMARY KEY in the "Persons" table, while the "Person_ID" column of the "Orders" table is a FOREIGN KEY.

```
/* Copy data from one table to another table (coping from product row 1 data to product_info table row1 */
insert into product_info(id, expiry_date, product_name, product_price )  select id, expiry_date, product_name, product_price from product where id =1
```

MySQL CLAUSES
==========

1. WHERE - is used to filter the results. It specifies a specific position where you have to do the operation.

Ex: select * from student where name = "Nani";

2. DISTINCT - is used to remove duplicate records from table and fetch only the unique records. The DISTINCT clause is used only with SELECT statement.

Ex: select distinct product_price from product

3. ORDER BY - is used to sort the records in ascending or descending order.

Ex: SELECT * FROM product ORDER BY product_name ASC; // ascending order

SELECT * FROM product ORDER BY product_name DESC; // descnding order.

SELECT * FROM product ORDER BY product_name; // by default ascending order.

4. GROUP BY - The MYSQL GROUP BY Clause is used to collect data from multiple records and group the result by one or more column. It is generally used in a SELECT statement.

You can also use some aggregate functions like COUNT, SUM, MIN, MAX, AVG etc. on the grouped column.

Ex: select product_name , count(*) from product group by product_name // group by with COUNT function.

select emp_name , sum(work_hours) as total_work_hours from emp_work_hours group by emp_name //group by with SUM() function.

select emp_name ,MIN(work_hours) as minimum_work_hours from emp_work_hours group by emp_name // group by with MIN() function.

select emp_name ,MAX(work_hours) as maximum_work_hours from emp_work_hours group by emp_name // group by with MAX() function.

select emp_name ,AVG(work_hours) as avg_work_hours from emp_work_hours group by emp_name // group by with AVG() function.

5. HAVING - MySQL HAVING Clause is used with GROUP BY clause. It always returns the rows where condition is TRUE.

select emp_name , sum(work_hours) as total_work_hours from

emp_work_hours group by emp_name having sum(work_hours) > 13 // returns records whose sum of work_hours is above 13 work_hours.

MYSQL CONTROL FLOW :
==========

Syntax: SELECT IF(expression1(check condition) , expression 2(true), expression 3(false);

IF EXPRESSION:

```
 SELECT IF(30<20, 'Yes', 'No');
 SELECT IF(200=30, 'CORRECT' , 'WRONG');
 SELECT IF(STRCMP('WELCOME', 'UNWELCOME')=0, 'SAME',  'DIFFERENT'); // comparing the strings

 SELECT emp_name,emp_salary, IF(emp_salary >55000, 'ELIGIBLE FOR LOAN', 'NOT ELIGIBLE FOR LOAN') AS result from employee;

 SELECT IFNULL('Welcome', 'Unwelcome'); /* it returns Welcome because , first expression is not null.*/
 SELECT IFNULL(NULL, 'Unwelcome'); /* it returns Unwelcome because , first expression is null.*/
```

CASE EXPRESSION:

Syntax:

```
CASE value
      WHEN [compare_value] THEN result
      [WHEN [compare_value] THEN result ...]
      [ELSE result]
END
```

EX: 1

```
SELECT CASE 'getout'
      WHEN 'welcome' THEN 'THANK YOU'
   WHEN 'Unwelcome' THEN 'NO PROBLEM'
   WHEN 'getout' THEN 'SAD'   /* This will be returned */
   ELSE 'INVALID'
END
```

EX: 2

```
SELECT emp_id, emp_name ,
CASE
       WHEN emp_salary > 50000 THEN 'Eligible for loan'
   WHEN emp_salary  < 50000 THEN 'Not eligible for loan'
   WHEN emp_salary = 60000 THEN 'Interest will be less'
   ELSE 'INVALID'
END AS result from employee
```

MySQL CONDITIONS:
==========

1. AND : select * from employee where emp_name ='Nani' AND emp_salary = 55000.2;

2. OR : select * from employee where emp_name = 'nani' OR emp_name = 'ramu';

3. AND OR : select * from employee where (emp_name = 'nani' AND emp_name = 'ramu') OR emp_salary>=55000

4. Boolean : SELECT studentid, name, pass FROM student1 WHERE pass = TRUE;

5. LIKE   : LIKE condition is used to perform pattern matching to find the correct result. It is used in SELECT, INSERT, UPDATE and DELETE statement with the combination of WHERE clause.

```
        select * from employee where emp_name LIKE '%van'; /* search for name based on given ending characters('van' - 'Pavan') */
        select * from employee where emp_name LIKE '%an%'; /* search for name based on given middle characters('%an%' - 'Pavan') */
        select * from employee where emp_name LIKE 'Pav%'; /* search for name based on given first characters('Pav%' - 'Pavan') */

        select * from employee where emp_name LIKE 'P_v_n%'; // search for name based on given index characters('Pav%' - 'Pavan') - using underscore
```

6. IN   : The MySQL IN condition is used to reduce the use of multiple OR conditions in a SELECT, INSERT, UPDATE and DELETE statement.

```
        select * from employee where id IN (3,7,8); /* it will fetch the records whose id's are 3,7,8 */
```

select * from employee where emp_name IN ('nani', 'pavan', 'raju');

7. ANY : The ANY keyword is a MySQL operator that returns the Boolean value TRUE if the comparison is TRUE for ANY of the subquery condition. In other words, this keyword returns true if any of the subquery condition is fulfilled when the SQL query is executed. The ANY keyword must follow the comparison operator.

It is noted that ALL SQL operator works related to ANY operator, but it returns true when all the subquery values are satisfied by the condition in MySQL.

The ANY operator works like comparing the value of a table to each value in the result set provided by the subquery condition. And then, if it finds any value that matches at least one value/row of the subquery, it returns the TRUE result.

select emp_salary from employee where emp_salary > ANY (select product_price from product) /** it returns true that mean employee salaries will be printed because salaries are greater than product_prices */

select product_price from product where product_price > ANY(select emp_salary from employee) /** it returns false(empty table) because product_prices are not greater than employee salaries */

8. NOT : select * from employee where emp_name NOT IN ('nani', 'pavan', 'raju');
select * from employee where id IN (3, 7);

9. NOT EQUAL : select * from employee where emp_id <> 6; /* NOT EQUAL */
select * from employee where emp_id != 6; /* NOT EQUAL : Above and this query will give same results */

10. NULL/NOT NULL : select * from employee where emp_name IS NULL; /* returns records whose names are having null value */
: select * from employee where emp_name IS NOT NULL; /* returns records whose names are not having null value */

MySQL JOINS :
==========

MySQL JOINS are used with SELECT statement. It is used to retrieve data from multiple tables.

It is performed whenever you need to fetch records from two or more tables.

1. INNER JOIN :
==========
The MySQL Inner Join is used to returns only those results from the tables that match the specified condition and hides other rows and columns.
MySQL assumes it as a default Join, so it is optional to use the Inner Join keyword with the query.

Syntax :SELECT columns
FROM table1
INNER JOIN table2 ON condition1
INNER JOIN table3 ON condition2
...;

ex: select e.emp_id, ew.working_date,e.emp_name, work_hours, emp_salary from employee e INNER JOIN emp_work_hours ew ON e.emp_id = ew.emp_id;

MySQL Inner Join with GROUP BY CLAUSE
---

select ew.day_id , e.emp_name, e.emp_id, e.emp_salary , ew.working_date from emp_work_hours ew LEFT JOIN employee e ON e.emp_id = ew.emp_id;

MySQL Inner Join with USING clause:

Sometimes, the name of the columns is the same in both the tables. In that case, we can use a USING keyword to access the records. The following query explains it more clearly:

select employee.emp_id, emp_work_hours.working_date,employee.emp_name, work_hours, emp_salary from employee INNER JOIN emp_work_hours USING(emp_id);

2. LEFT JOIN
==========

The Left Join in MySQL is used to query records from multiple tables.
This clause is similar to the Inner Join clause that can be used with a SELECT statement immediately after the FROM keyword.
When we use the Left Join clause, it will return all the records from the first (left-side) table, even no matching records found from the second (right side) table.

If it will not find any matches record from the right side table, then returns null.

Syntax : SELECT columns
FROM table1
LEFT JOIN table2
ON Join_Condition;

example :

```
select   ew.day_id , e.emp_name, e.emp_id, e.emp_salary , ew.working_date from
emp_work_hours  ew LEFT JOIN   employee e  ON  e.emp_id = ew.emp_id;
```

MySQL LEFT JOIN with GROUP BY CLAUSE

```
select   ew.day_id , e.emp_name, e.emp_id, e.emp_salary , ew.working_date from
emp_work_hours  ew LEFT JOIN   employee e  ON  e.emp_id = ew.emp_id GROUP BY
e.emp_company_name
```

3. RIGHT-JOIN

==========

The Right Join is used to joins two or more tables and returns all rows from the right-hand table, and only those results from the other table that fulfilled the join condition.
If it finds unmatched records from the left side table, it returns Null value.
It is similar to the Left Join, except it gives the reverse result of the join tables.
It is also known as Right Outer Join. So, Outer is the optional clause used with the Right Join.

Syntax: SELECT column_list
FROM Table1
RIGHT JOIN Table2
ON join_condition;

Example : SELECT emp_name,c.id, customer_name,state from customer c
RIGHT JOIN employee e ON c.id = e.emp_id
RIGHT JOIN product p ON c.id = p.id

4. CROSS-JOIN

==========

MySQL CROSS JOIN is used to combine all possibilities of the two or more tables and returns the result that contains every row from all contributing tables.
The CROSS JOIN is also known as CARTESIAN JOIN, which provides the Cartesian product of all associated tables.
The Cartesian product can be explained as all rows present in the first table multiplied by all rows present in the second table.
It is similar to the Inner Join, where the join condition is not available with this clause.

Syntax : SELECT *
FROM customers
CROSS JOIN contacts;

Example : SELECT * FROM employee CROSS JOIN emp_work_hours

5. SELF-JOIN

==========

A SELF JOIN is a join that is used to join a table with itself.
In the previous sections, we have learned about the joining of the table with the other tables using different JOINS, such as INNER, LEFT, RIGHT, and CROSS JOIN.
However, there is a need to combine data with other data in the same table itself. In that case, we use Self Join.

We can perform Self Join using table aliases. The table aliases allow us not to use the same table name twice with a single statement.
If we use the same table name more than one time in a single query without table aliases, it will throw an error.

The table aliases enable us to use the temporary name of the table that we are going to use in the query.
Let us understand the table aliases with the following explanation.

Suppose we have a table named "student" that is going to use twice in the single query. To aliases the student table, we can write it as:

Select â ¦ FROM student AS S1
INNER JOIN student AS S2;

Syntax : SELECT s1.col_name, s2.col_name...
FROM table1 s1, table1 s2
WHERE s1.common_col_name = s2.common_col_name;

example : SELECT s1.student_id, s1.name
FROM student AS s1, student s2
WHERE s1.student_id=s2.student_id
AND s1.course_id<>s2.course_id;

i. SELF JOIN using INNER JOIN clause

```
SELECT  s1.student_id, s1.name
FROM student s1
INNER JOIN student s2
ON s1.student_id=s2.student_id
AND s1.course_id<>s2.course_id
GROUP BY student_id;
```

ii. SELF-JOIN using LEFT JOIN clause

```
SELECT (CONCAT(s1.stud_lname, ' ', s2.stud_fname)) AS 'Monitor',
s1.city
FROM students s1
LEFT JOIN students s2 ON s1.student_id=s2.student_id
ORDER BY s1.city DESC;
```