MYSQL

1. Data:

Data is defined as facts or figures, or information that's stored in or used by a computer. Ex: Information collected for a research paper, an Email,etc.

2. **Database:**

Database is nothing but an organized form of data for easy access, storing, retrieval and managing of data. This is also known as structured form of data which can be accessed in many ways.

Example: School Management Database, Bank Management Database.

3. **DBMS:**

A Database Management System (DBMS) is a program that controls creation, maintenance and use of a database. DBMS can be termed as File Manager that manages data in a database rather than saving it in file systems.

4.) RDBMS:

RDBMS stands for Relational Database Management System. RDBMS store the data into the collection of tables, which is related by common fields between the columns of the table. It also provides relational operators to manipulate the data stored into the tables.

Example: SQL Server.

5.) Tables & Fields:

A table is a set of data that are organized in Columns and Rows. Columns can be categorized as vertical, and Rows are horizontal. A table has specified number of column called fields but can have any number of rows which is called record.

Example:.

Table Name : Employee.

Fields: Emp ID, Emp Name, Date of Birth.

Data: 201456, David, 11/15/1960.

SQL

● SQL stands for **Structured Query Language.**

● SQL was earlier known as SEQUEL.

● It is used to communicate with the Database.

● This is a standard language used to perform tasks such as retrieval, updation, insertion and deletion of data from a database.

➢ **What Sql can do?**

● SQL can execute queries against a database

● SQL can retrieve data from a database

● SQL can insert records in a database

● SQL can update records in a database

● SQL can delete records from a database

● SQL can create new databases

● SQL can create new tables in a database

● SQL can create views in a database

● SQL can set permissions on tables and views

Types of SQL Commands

SQL Categorizes its commands on the basis of functionalities performed by them. There are five types of SQL Commands which can be classified as:

Group	Statements	Description
Data Definition Language (DDL)	CREATE ALTER DROP TRUNCATE	Allows to create data structures i.e. tables, delete, alter etc.,
Data Manipulation Language (DML)	INSERT UPDATE DELETE	Allows modification of data in the database
Data Query Language (DQL)	SELECT SHOW HELP	Helps in Retrieving data and information from the database.
Data Control Language (DCL)	GRANT REVOKE	Allows or denies permissions to access the tables.
Transaction Control Language (TCL)	COMMIT ROLLBACK SAVEPOINT	Manages the changes made by the DML statements.

Operators in Sql

An operator is a reserved character or word which is used in a SQL statement to query our database. We use a WHERE clause to query a database using operators.Operators are needed to specify conditions in a SQL statement.

1.	Types	Operators
	Arithmetic Operators	+ , -, *, /, %
2.	Comparison Operators	<, >, =, !=, <=, >=
3.	Logical Operators	AND, OR, NOT,BETWEEN, IN, ANY, ALL,LIKE

Expressions in SQL

● An expression is a combination of one or more values, operators and SQL functions used to query a database to get a specific set of data.

● These SQL EXPRESSIONS are like formulae and they are written in query language.

There are different types of SQL expressions, which are mentioned below –

1. Boolean Expression:

SQL Boolean Expressions fetch the data based on matching a single value.

Syntax: SELECT column1, column2, columnN

FROM table_name

WHERE SINGLE VALUE MATCHING EXPRESSION;

E.g: SELECT * FROM CUSTOMERS WHERE SALARY = 10000;

1. Numeric Expression:

These expressions are used to perform any mathematical operation in any query.

Syntax: SELECT numerical_expression as OPERATION_NAME

[FROM table_name

WHERE CONDITION] ;

E.g: SELECT (15 + 6) AS ADDITION

3.) **Date Expression:**

Date Expressions return current system date and time values .

E.g.: SELECT CURRENT_TIMESTAMP;

```
MariaDB [customer]> select current_timestamp;
+---------------------+
| current_timestamp   |
+---------------------+
| 2022-03-11 15:45:54 |
+---------------------+
1 row in set (0.007 sec)
```

DataTypes in SQL

● Data types are used to represent the nature of the data that can be stored in the database table. ● Each column in a database table is required to have a name and a data type.

● We must decide what type of data that will be stored inside each column when creating a table. **1. Numeric:**

Category	Datatype	Description	Size
Numeric	Int	A normal-sized integer that can be signed or unsigned.	4 bytes
Numeric	Tinyint	A very small integer that can be signed or unsigned.	1 byte
Numeric	Smallint	A small integer that can be signed or unsigned.	2 bytes
Numeric	Mediumint	A medium-sized integer that can be signed or unsigned.	3 bytes
Numeric	Bigint	A large integer that can be signed or unsigned.	8 bytes
Numeric	Float(m,d)	A floating-point number that cannot be unsigned.	4 bytes
Numeric	Double(m,d)	A double precision floating-point number that cannot be unsigned.	8 bytes
Numeric	Decimal(m,d)	An unpacked floating-point number that cannot be unsigned.	Length + 1 byte

2.) Date & Time:

Date & Time	Date	A date in YYYY-MM-DD format, between 1000-01-01 and 9999-12-31.	3 bytes
Date & Time	Datetime	A date and time combination in YYYY-MM-DD HH:MM:SS format, between 1000-01-01 00:00:00 and 9999-12-31 23:59:59.	8 bytes
Date & Time	Timestamp	A timestamp between midnight, January 1st, 1970 and sometime in 2037.	4 bytes
Date & Time	Time	Stores the time in a HH:MM:SS format.	3 bytes
Date & Time	Year(m)	Stores a year in a 2-digit or a 4-digit format.	2 bytes

3.) String:

String	Char(m)	A fixed-length string between 1 and 255 characters in length (for example CHAR(5)), right-padded with spaces to the specified length when stored.	Length bytes
String	Varchar(m)	A variable-length string between 1 and 255 characters in length.	String length + 1 byte
String	Blob or Text	A field with a maximum length of 65,535 characters.	String length + 2 byte
String	Tinyblob or tinytext	A BLOB or TEXT column with a maximum length of 255 characters. You do not specify a length with TINYBLOB or TINYTEXT.	String length + 1 byte
String	Mediumblob or mediumtext	A BLOB or TEXT column with a maximum length of 1,67,77,215 characters. You do not specify a length with MEDIUMBLOB or MEDIUMTEXT.	String length + 3 byte
String	Longblob or Longtext	A BLOB or TEXT column with a maximum length of 429,49,67,295 characters. You do not specify a length0 with LONGBLOB or LONGTEXT.	String length + 4 byte

Constraints in SQL

● Constraints are certain conditions,rules or restrictions we apply on the database. ● Constraints could be either on a column level or a table level.

● The column level constraints are applied only to one column, whereas the table level constraints are applied to the whole table.

1. NOT NULL:

● NULL means empty, i.e., the value is not available.

● Whenever a table's column is declared as NOT NULL, then the value for that column cannot be empty for any of the table's records.

Syntax:

CREATE TABLE TableName (ColumnName1 datatype NOT NULL, ColumnName2 datatype,…., ColumnNameN datatype); E.g:

CREATE TABLE student(StudentID **INT** NOT NULL, Student_FirstName **VARCHAR**(20), Student_LastName **VARCHAR**(20));

2.) UNIQUE:

● Duplicate values are not allowed in the columns to which the UNIQUE constraint is applied. ● The column with the unique constraint will always contain a unique value.

● This constraint can be applied to one or more than one column of a table, which means more than one unique constraint can exist on a single table.

● Using the UNIQUE constraint, you can also modify the already created tables.

Syntax:

CREATE TABLE TableName (ColumnName1 datatype **UNIQUE**, ColumnName2 datatype,…., ColumnNameN datatype); E.g:

CREATE TABLE student(StudentID **INT UNIQUE**, Student_FirstName **VARCHAR**(20), Student_LastName **VARCHAR**(20));

3. PRIMARY KEY:

● PRIMARY KEY Constraint is a combination of NOT NULL and Unique constraints.

● NOT NULL constraint and a UNIQUE constraint together forms a PRIMARY constraint. ● The column to which we have applied the primary constraint will always contain a unique value and will not allow null values.

● Primary keys cannot be NULL, unique keys can be. There can be more UNIQUE columns, but only one primary key in a table.

● Primary keys become foreign keys in other tables, when creating relations among tables. Syntax:

CREATE TABLE TableName (ColumnName1 datatype **PRIMARY KEY**, ColumnName2 datatype,…., ColumnNameN datatype); E.g:

CREATE TABLE student(StudentID **INT PRIMARY KEY**, Student_FirstName **VARCHAR**(20), Student_LastName **VARCHAR**(20));

4. FOREIGN KEY

● A FOREIGN KEY in one table points to a PRIMARY KEY in another table. It is a referential constraint between two tables.

● When we have two tables, and one table takes reference from another table, i.e., the same column is present in both the tables and that column acts as a primary key in one table. That particular column will act as a foreign key in another table.

Syntax:

CREATE TABLE tablename(ColumnName1 Datatype(**SIZE**) **PRIMARY KEY**, ColumnNameN Datatype(**SIZE**), **FOREIGN KEY**(ColumnName) **REFERENCES** PARENT_TABLE_NAME(Primary_Key_ColumnName));

E.g.:

CREATE TABLE employee (Emp_ID **INT** NOT NULL **PRIMARY KEY**, Emp_Name **VARCHAR** (40), Emp_Salary **VARCHAR** (40));

5. CHECK

● Whenever a check constraint is applied to the table's column, and the user wants to insert the value in it, then the value will first be checked for certain conditions before inserting the value into that column.

Syntax:

CREATE TABLE TableName (ColumnName1 datatype **CHECK** (ColumnName1 Condition), ColumnName2 datatype,...., ColumnNameN datatype);

E.g.:

CREATE TABLE student(

StudentID **INT**, Student_FirstName **VARCHAR**(20),

Student_LastName **VARCHAR**(20),

Student_PhoneNumber **VARCHAR**(20),

Student_Email_ID **VARCHAR**(40),

Age **INT CHECK**(Age <= 15));

6. DEFAULT

● Whenever a default constraint is applied to the table's column, and the user has not specified the value to be inserted in it, then the default value which was specified while applying the default constraint will be inserted into that particular column.

Syntax:

CREATE TABLE TableName (ColumnName1 datatype **DEFAULT** Value, ColumnName2 datatype,…., ColumnNameN datatype);

E.g:

CREATE TABLE student(

StudentID **INT**,

Student_FirstName **VARCHAR**(20),

Student_LastName **VARCHAR**(20),

Student_PhoneNumber **VARCHAR**(20),

Student_Email_ID **VARCHAR**(40) **DEFAULT** 'xyz8@gmail.com');

DATA DEFINITION LANGUAGE (DDL)

● Data Definition Language consists of the SQL commands that can be used to define the database schema. ● It is a set of SQL commands used to create, modify, and delete database structures but not data.

S.No	DDL Commands	Description	Sample Query
1.	CREATE	Used to create tables or databases.	CREATE table student;
2.	ALTER	Used to modify the values in the tables.	ALTER table student add column roll_no int;
3.	RENAME	Used to rename the table or database name.	RENAME student to student_details;
4.	DROP	Deletes the table from the database.	DROP table student_details;
5.	TRUNCATE	Used to delete a table from database.	TRUNCATE table student_details;

Creating a Database

● **Creating a database:**

Syntax: **CREATE DATABASE** Database_Name;

E.g: **CREATE DATABASE** Student ;

● **To check that your database is created in SQL:** SHOW **DATABASES** ;

● **Selecting a MySQL Database:**

USE database_name;

● **Removing Databases:**

DROP DATABASE database_name;

What is a Table?

● Table is a collection of data, organized in terms of rows and columns.

● In DBMS term, table is known as relation, columns as fields and row as a record or tuple. ● A table has a specified number of columns, but can have any number of rows. ● It is a simple form of data storage.

EMP_NAME	ADDRESS	SALARY
Ankit	Lucknow	15000
Raman	Allahabad	18000
Mike	New York	20000

1.Creating a Table

SQL CREATE TABLE statement is used to create table in a database. **Syntax:**

create table "tablename"

("column1" "data type",

"column2" "data type",

"column3" "data type",

...

"columnN" "data type");

E.g:

CREATE TABLE STUDENTS (

ID **INT**,

NAME VARCHAR (20) NOT NULL,

AGE **INT** NOT NULL,

ADDRESS **CHAR** (25),

PRIMARY KEY (ID)

);

2. ALTER TABLE COMMAND

● The ALTER TABLE command allows you to add, modify, and delete columns of an existing table. ● This statement also allows database users to add and remove various SQL constraints on the existing tables.

● Any user can also change the name of the table using this statement.

1. <u>ALTER TABLE ADD Column:</u>

Syntax: ALTER TABLE table_name ADD column_name column-definition;

E.g: ALTER TABLE student

ADD marks INT;

Note: To add multiple columns:

Alter table table_name

ADD column_name,ADD column_name;

1. <u>ALTER TABLE MODIFY Column:</u>

Syntax: ALTER TABLE table_name MODIFY column_name column-definition;

E.g:

ALTER TABLE student

MODIFY NAME varchar(40);

3.) <u>ALTER TABLE RENAME Column:</u>

Syntax:

ALTER TABLE table_name change COLUMN old_name new_name datatype;

E.g: ALTER TABLE STUDENTS

Change column First_NAME Stud_Name varchar(20);

4.)<u>ALTER TABLE DROP Column:</u>

Syntax:

ALTER TABLE table_name DROP column_name ;

E.g: ALTER TABLE students

DROP ADDRESS;

Note: For Dropping Multiple columns:

Alter table table_name

DROP column_name,

DROP column_name;

3. RENAME TABLE COMMAND

The RENAME TABLE and ALTER TABLE syntax helps us to change the name of the table.

Syntax: ALTER TABLE old_table_name

RENAME TO new_table_name;

E.g: ALTER TABLE STUDENTS

RENAME TO STUDENT_DETAILS;

4. TRUNCATE TABLE COMMAND

● A truncate SQL statement is used to remove all rows (complete data) from a table. ● It is similar to the DELETE statement without WHERE clause.

● Truncate table is faster and uses less resources than DELETE TABLE command. ● Drop table command can also be used to delete complete table but it deletes table structure too. ● TRUNCATE TABLE doesn't delete the structure of the table.

Syntax: **TRUNCATE TABLE** table_name;

E.g: TRUNCATE TABLE STUDENTS;

5. DROP TABLE COMMAND

● DROP TABLE statement is used to delete a table definition and all data from a table. Syntax: **DROP TABLE** "table_name";

E.g: DROP TABLE STUDENTS;

DATA MODIFICATION LANGUAGE (DML)

Once the tables are created and database is generated using DDL commands, manipulation inside those tables and databases is done using DML commands.

S.No	DML Command	Description	Sample Query
1.	INSERT	Used to insert new rows in the tables.	INSERT into student(roll_no, name) values(1, Anoop);
2.	DELETE	Used to delete a row or entire table.	DELETE table student;
3.	UPDATE	Used to update values of existing rows of tables.	UPDATE students set s_name = 'Anurag' where s_name like 'Anoop';

INSERT COMMAND

● It is used to insert a single or a multiple records in a table.

● There are two basic syntaxes of the INSERT INTO statement which are shown below:

Syntax 1: INSERT INTO TABLE_NAME (column1, column2, column3,...column N) VALUES (value1, value2, value3,...valueN);

E.g: INSERT INTO STUDENTS (ID,NAME,AGE,ADDRESS)

VALUES (101,'PRACHITI',25,'THANE');

Syntax 2: INSERT INTO TABLE_NAME VALUES (value1,value2,value3,...valueN); E.g: INSERT INTO STUDENTS VALUES (101,'PRACHITI',25,'THANE');

UPDATE COMMAND

● We use the UPDATE statement to update existing data in a table.

● We can use the UPDATE statement to change column values of a single row, a group of rows, or all rows in a table.

● Following is syntax to **update all rows in a table**:

Syntax: UPDATE table_name

SET

column_name1 = expr1,

column_name2 = expr2,

E.g: UPDATE STUDENTS

SET CITY='THANE'

● Following the syntax to **update a particular record/row**:

Syntax: UPDATE table_name

SET

column_name1 = expr1,column_name2 = expr2,

WHERE

Condition;

E.g: UPDATE STUDENTS

SET MARKS=50

WHERE ID=104;

DELETE COMMAND

● The **DELETE statement** is used to delete rows from a table. ● Generally DELETE statement removes one or more records from a table.

Syntax: **DELETE FROM** table_name

[**WHERE** condition];

E.g: **DELETE FROM** students

WHERE ID=101;

● To delete all the records from the table:

Syntax: **DELETE FROM** table_name;

E.g: DELETE FROM students;

Difference between DELETE and TRUNCATE statements:

● The **DELETE statement** only deletes the rows from the table based on the condition defined by WHERE clause or delete all the rows from the table when condition is not specified.

● But it does not free the space containing by the table.

● The **TRUNCATE statement:** it is used to delete all the rows from the table **and free the containing space.**

DATA QUERY LANGUAGE

● Data query language consists of command over which data selection in SQL relies. ● SELECT command in combination with other SQL clauses is used to retrieve and fetch data from database/tables on the basis of certain conditions applied by user.

● By using this command, we can also access the particular record from the particular column of the table. ● The table which stores the record returned by the SELECT statement is called a result-set table.

Syntax:

● **To Select all attributes(columns) and tuples(rows) from a table**:

SELECT * **FROM** table_name;

E.g: **SELECT** * **FROM** Students;

● **To Select few attributes and all tuples from a table:**

SELECT Column_Name_1, Column_Name_2, ..., Column_Name_N **FROM** Table_Name; E.g: SELECT NAME,AGE FROM Students;

● **Select Distinct :**

The **SQL DISTINCT command** is used with SELECT keyword to retrieve only distinct or unique data.

Syntax:

SELECT DISTINCT column_name ,column_name

FROM table_name;

E.g:

SELECT DISTINCT city

FROM students;

' WHERE ' Clause

● The WHERE clause is used to filter records.

● It is used to extract only those records that fulfill a specified condition. ● WHERE clause is used in SELECT, UPDATE, DELETE statement etc.

Syntax:

SELECT column1, column2,..column N

FROM table_name

WHERE condition;

E.G:

SELECT ID, NAME, SALARY

FROM CUSTOMERS

WHERE SALARY > 2000;

Where clause with Arithmetic operators Where clause with comparison operators

S.No	Operator	Description
1.	+	Addition
2.	-	Subtraction
3.	*	Multiply
4.	/	Division
5.	%	Modulo

S. No	Operator	Description
1.	<	Less Than
2.	>	Greater Than
3.	=	Equal To
4.	<=	Less Than Or Equal To
5.	>=	Greater Than Or Equal To
6.	<>	Not Equal To

Logical Operators

● Logical SQL operators are used between queries or used to join two or more conditions. ● These operators compare two conditions at a time to determine whether a row can be selected for the output.

S.No	Operator	Description
1.	LIKE	Used for matching or checking.
2.	AND	Used for combining two conditions together.
3.	IN	Used to compare values.
4.	BETWEEN	Used to compare value within a range.
5.	OR	Used to check either of the two comparison.
6.	NOT	Used to fetch value other than values which are not required.

1. AND Operator:

Logical AND compares two Booleans as expression and returns TRUE when both of the conditions are TRUE and returns FALSE when either is FALSE.

Syntax:

SELECT column1, column2, ...

FROM table_name

WHERE condition1 AND condition2 AND condition 3 ...;

For Example:

To find the names of the students between the age 10 to 15 years, the query would be like:

SELECT first_name, last_name, age

FROM student_details

WHERE age >= 10 AND City='Thane';

2. OR operator:

Logical OR compares two Booleans as expression and returns TRUE when either of the conditions is TRUE and returns FALSE when both are FALSE.

Syntax:

SELECT column1, column2, ...

FROM table_name

WHERE condition1 OR condition2 OR condition3 ...;

For example:

To find the names of students who are studying either Maths or Science, the query would be like, SELECT first_name, last_name, subject

FROM student_details

WHERE subject = 'Maths' OR subject = 'Science'

3. NOT Operator:

● If you want to find rows that do not satisfy a condition, you can use the logical operator, NOT. ● NOT results in the reverse of a condition.

● That is, if a condition is satisfied, then the row is not returned.

Syntax:

SELECT column1, column2, ...

FROM table_name

WHERE NOT condition;

For example:

To find out the names of the students who do not play football, the query would be like:
SELECT first_name, last_name, games

FROM student_details

WHERE NOT games = 'Football'

4. BETWEEN Operator:

● This operator displays the records which fall between the given ranges. ● The results of the BETWEEN operator include begin and end values of the given range.

Syntax:

SELECT column_name(s)

FROM table_name

WHERE column_name BETWEEN value1 AND value2;

E.g:

SELECT * **FROM** employees **WHERE** Salary BETWEEN 5000 AND 9000;

➢ NOT BETWEEN:

To display the products outside a specific range we make use of NOT BETWEEN. E.g: **SELECT** * **FROM** employees **WHERE** Salary NOT BETWEEN 5000 AND 9000;

5. IN Operator

● When we want to check for one or more than one value in a single SQL query, we use IN operator. ● The IN operator allows you to determine if a value matches any value in a list of values.

Syntax:

SELECT column_name(s)

FROM table_name

WHERE column_name IN (value1, value2, ...);

● IN operator is functionally equivalent to the combination of multiple OR operators: value = value1 OR value = value2 OR value = value3 OR …

E.g: **SELECT** * **FROM** employees **WHERE** age IN (25, 27, 24);

➢ **NOT IN operator:**

● The NOT IN operator is used when we don't want to match certain values in the list. E.g: **SELECT** * **FROM** employees **WHERE** age NOT IN (25, 27, 24);

6. LIKE Operator

● LIKE Operator in SQL displays only those data from the table which matches the pattern specified in the query. ● There are two wildcards used in conjunction with the LIKE operator:

The percent sign (%) : Represents zero, one or multiple characters.

The underscore (_): Represents a single number or character.

Syntax:

SELECT column1, column2, ...

FROM table_name

WHERE columnN LIKE pattern;

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

➢ NOT LIKE Operator:

● NOT LIKE Operator works exactly opposite the Like operator.

● It displays only those data from the table which does not match the pattern specified in the query. Syntax:

SELECT column1, column2, ...

FROM table_name

WHERE columnN NOT LIKE pattern;

E.g:

SELECT * FROM Customers

WHERE CustomerName NOT LIKE 'a%';

SQL NULL Values

● A field with a NULL value is a field with no value.

● It is not possible to test for NULL values with comparison operators, such as =, <, or <>. ● We will have to use the ISNULL and IS NOT NULL operators instead.

IS NULL Syntax:

SELECT column_names

FROM table_name

WHERE column_name IS NULL;

E.g: SELECT *

FROM student

WHERE age IS NULL;

IS NOT NULL Syntax:

SELECT column_names

FROM table_name

WHERE column_name IS NOT NULL;

E.g: SELECT *

FROM student

WHERE age IS NOT NULL;

LIMIT Command

● The LIMIT clause is used to specify the number of records to return.

● The LIMIT clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

Syntax:

SELECT column_list

FROM table_name

LIMIT offset, count;

E.g:

SELECT * FROM Emp_info

LIMIT 3,7;

ORDER BY Clause

● The ORDER BY clause is used to sort the result-set in ascending or descending order. ● The ORDER BY clause sorts the records in ascending order by default. ● To sort the records in descending order, use the DESC keyword.

Syntax:

SELECT column1, column2, ...

FROM table_name

ORDER BY column1, column2, ... ASC|DESC;

E.g:

SELECT * FROM Emp_info

ORDER BY City;

SELECT * FROM CUSTOMERS

ORDER BY NAME DESC;

Renaming Attributes or SQL Aliases

● SQL aliases are used to give a table, or a column in a table, a temporary name. ● Aliases are often used to make column names more readable. ● An alias only exists for the duration of that query.

● An alias is created with the AS keyword.

Syntax:

SELECT column_name AS alias_name

FROM table_name;

E.g:

SELECT ID AS emp_ID, Name AS emp_name

FROM emp_info;

BUILT-IN SQL FUNCTIONS

● A function is **a set of SQL statements that perform a specific task**.

● Functions provides code reusability.

● If you have to repeatedly write large SQL scripts to perform the same task, you can create a function that performs that task.

● Next time instead of rewriting the SQL, you can simply call that function.

● There are 4 types of functions in sql,as follow:

1. String()

1. Math()

1. Date()

1. Aggregate()

1. STRING():

String methods in SQL are useful for processing the string data type or manipulation of string values. **1. Concat:**

The CONCAT() function adds two or more strings together.

Syntax: Select concat('string1','string2',....'stringN');

E.g: Select concat('xyz','abc');

Select concat('xyz',' ','abc');

2. **Lower:** This function is used to convert the upper case character into lower case. Syntax: select LOWER(text)

E.g: SELECT LOWER('GOOD MORNING');

SELECT LOWER(Name) AS LowercaseempName

FROM emp_info;

3. Upper: This function converts the lower case character into the upper case.

Syntax: select UPPER(text)

E.g: SELECT UPPER('good morning');

SELECT UPPER(Name) AS UppercaseempName

FROM emp_info;

4. Replace: This function is used to replace all occurrences of the substring in a specified string with another string value. Syntax: REPLACE(string, old_string, new_string)

E.g: SELECT REPLACE('Good Morning', 'Good', 'Happy');

SELECT REPLACE('Good Morning', 'G', 'F');

5. Reverse() : This function displays the character string in reverse order.

Syntax: REVERSE(string)

E.g: SELECT REVERSE('Good Morning');

SELECT REVERSE(Name) AS reversename

FROM emp_info;

6. Length() : This function returns the number of characters in a string, including trailing spaces. Syntax: LENGTH(string)

E.g: SELECT LENGTH('Good Morning');

SELECT LENGTH(name) from emp_info;

7. Substring() : This function extracts a substring from a string that begins at a specific position and ends at a specific length. Syntax: SUBSTRING(string, start, length)

E.g: SELECT SUBSTRING('Good morning', 1, 3) AS ExtractString;

SELECT SUBSTRING(Name, 1, 4) AS ExtractString

FROM emp_info;

8. Ltrim(): This function returns a string from a given string after removing all leading spaces.

Syntax: LTRIM(string)

E.g: SELECT LTRIM(' Hello world') AS LeftTrimmedString;

9. Rtrim(): This function returns a string from a given string after removing all trailing spaces.

Syntax: RTRIM(string)

E.g: SELECT RTRIM('Hello world ') AS RightTrimmedString;

Math() Functions:

● Mathematical functions are present in SQL which can be used to perform mathematical calculations. ● Some commonly used mathematical functions are given below:

1. ABS(X): This function returns the absolute value of a number.

E.g: Select abs(-6);

1. MOD(X,Y): The variable X is divided by Y and their remainder is returned.

E.g: Select mod(9,5);

1. FLOOR(X): This returns the largest integer value that is either less than X or equal to it. E.g: SELECT FLOOR(25.75)

SELECT FLOOR(-13.5)

1. CEIL(X): This returns the smallest integer value that is either more than X or equal to it. E.g: SELECT CEILING(25.75)

SELECT CEILING(-13.5)

5. TRUNCATE(X,D): Returns the number X, truncated to D decimal places. If D is 0, the result has no decimal point or fractional part.

E.g: SELECT TRUNCATE(123.321,2)

SELECT TRUNCATE(123.321,-1)

6. **EXP(X):** The EXP() function returns e raised to the power of a specified number.

E.g: SELECT EXP(2);

7. **POWER(X,Y) :** The POWER() function returns the value of a number raised to the power of another number. E.g: SELECT POWER(4, 2);

8. **SQRT(X):** Return the square root of a number.

E.g: Select sqrt(144);

DATE Function

1. CURDATE(): Returns the current date.

Syntax:

Select Curdate();

Eg:

create table orders

(

order_id int,

pro_name varchar(50),

order_date datetime default curdate(),

primary key(order_id));

insert into orders (order_id,pro_name) values (102,'Pen'); **1. Now():** Returns the current date and time.

Syntax:

SELECT NOW();

1. Sysdate(): Returns the system's current date & time.

Syntax:

SELECT SYSDATE();

4. **Last_day(date):** Returns the last day of the corresponding month for a date or datetime value**.**

Syntax:

Select last_day(date);

E.g:

Select last_day('2022-03-12');

5. **Date_format(date, format):** To format a date value to a specific format, you use the DATE_FORMAT function.

Syntax:

DATE_FORMAT(date,format)

E.g:

DATE_FORMAT(NOW(),'%b %d %Y %h:%i %p')

DATE_FORMAT(NOW(),'%m-%d-%Y')

DATE_FORMAT(NOW(),'%d %b %y')

6. **DATEDIFF():** The DATEDIFF() function returns the time between two dates.

Syntax: DATEDIFF(date1,date2)

E.g: SELECT DATEDIFF('2014-11-30','2014-11-29') AS DiffDate

7. **MONTH(date):** Returns the month for date, in the range 1 to 12 for January to December, or 0 for dates such as '0000-00-00' or '2008-00-00' that have a zero month part.

Syntax: Select month(date);

E.g: select month(now());

8. **YEAR(date):** Returns the year for date, in the range 1000 to 9999, or 0 for the "zero" date.

Syntax: Select year(date);

E.g: select year(now());

Aggregate() Function

● An aggregate function in SQL returns one value after calculating multiple values of a column. ● Aggregate functions are also known as group functions.

1. AVG(): The AVG() function calculates the average of a set of values. Syntax:

Select avg(column_name) as alias_name from table_name;

E.g:

Select avg(salary) as avg_sal from emp_info;

1. Count(): The COUNT() function returns the number of rows in a database table.

Syntax:

Select count(column_name) from table_name;

E.g:

Select count(name) from emp_info;

3. **Max():** The MAX() function returns the maximum value of the selected column. Syntax: SELECT MAX(column_name) FROM table_name;

E.g: Select max(salary) from emp_info;

4. **Min():** The MIN() function returns the minimum value of the selected column.

Syntax:

SELECT MIN(column_name) FROM table_name;

E.g:

Select min(age) from emp_info;

5. **Sum():** The SUM() function returns the total sum of a numeric column.

Syntax:

SELECT SUM(column_name) FROM table_name;

E.g:

Select sum(age) from emp_info;

' Group By ' Clause

● The **Group By** statement is used along with the select statement for organizing similar data into groups. ● It is a command that is used to **group rows that have the same values**.

● The GROUP BY clause is used in the SELECT statement. Optionally it is used in conjunction with aggregate functions to produce summary reports from the database.

● It is used to summarize data from the database.

Points to remember:

● **The SELECT** statement is used with the **GROUP BY** clause in the SQL query.

● **WHERE** clause is placed before the **GROUP BY** clause in **SQL**.

● **ORDER BY** clause is placed after the **GROUP BY** clause in **SQL**.

Syntax:

SELECT column1, function_name(column2)

FROM table_name

WHERE condition

GROUP BY column1, column2

ORDER BY column_name(asc|desc)

E.g: **1. Group by with Count() function:**

SELECT CITY, COUNT (NAME) **FROM** Employee **GROUP BY**

```
MariaDB [employee]> select city,count(name) from emp_info group by city;
+--------+-------------+
| city   | count(name) |
+--------+-------------+
| jaipur |           1 |
| mumbai |           2 |
| nagpur |           1 |
| pune   |           2 |
| thane  |           2 |
+--------+-------------+
```
CITY;

2. Group by with sum() Function: Select city,sum(salary) from emp_info

Group by City;

```
MariaDB [employee]> select city,sum(salary) from emp_info group by city;
+--------+-------------+
| city   | sum(salary) |
+--------+-------------+
| jaipur |       10000 |
| mumbai |       25000 |
| nagpur |        7000 |
| pune   |       44000 |
| thane  |       15000 |
+--------+-------------+
5 rows in set (0.001 sec)
```


Select age,city from emp_info group by age,city;

```
MariaDB [employee]> select * from emp_info;
+-----+---------+-----+--------+--------+
| id  | name    | age | city   | salary |
+-----+---------+-----+--------+--------+
| 101 | aditi   |  25 | thane  |  10000 |
| 102 | riya    |  24 | pune   |  22000 |
| 103 | raj     |  27 | mumbai |   5000 |
| 104 | diya    |  28 | nagpur |   7000 |
| 105 | ravi    |  24 | jaipur |  10000 |
| 106 | priya   |  25 | thane  |   5000 |
| 107 | manisha |  28 | pune   |  22000 |
| 108 | seema   |  28 | mumbai |  20000 |
+-----+---------+-----+--------+--------+
```

Note: If the age is the same but the city is different, then a row is treated as a unique one . If the age and the city is the same for more than one row, then it's considered a duplicate and only one row is shown.

3. Group by with min() function:

SELECT dept_id, MIN(salary) FROM employees GROUP BY dept_id;

4. Group by with max() function:

SELECT dept_id, Max(salary) FROM employees GROUP BY dept_id;

Having clause:

The HAVING clause places the condition in the groups defined by the GROUP BY clause in the SELECT statement. The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.

Syntax:

SELECT column_Name1, column_Name2, ..., column_NameN aggregate_function_name(column_Name) **FROM** table_name

GROUP BY column_Name1

HAVING condition;

Eg.

select sum(salary),city from emp_info

group by city

having sum(salary)>10000;

1. Using sum() function:

SELECT designation, SUM(salary) FROM emp_info

GROUP BY designation

HAVING SUM(salary) > 10000; 1. Using Count() function:

SELECT designation, COUNT(*) FROM emp_info

WHERE salary > 2000

GROUP BY designation

HAVING COUNT(*) > 2;

1. Using min() function:

SELECT designation, MIN(salary) FROM emp_info

GROUP BY designation

HAVING MIN(salary) > 5000;

4. Using max() function:

SELECT designation, Max(salary) FROM emp_info

GROUP BY designation

HAVING Max(salary) > 5000;

Foreign Key

● The foreign key is used to link one or more than one table together. It is also known as the **referencing** key.

● In simple words you can say that, a foreign key in one table used to point primary key in another table.

● It means a foreign key field in one table refers to the primary key field of the other table. ● It identifies each row of another table uniquely that maintains the **referential integrity** in MySQL. ● A foreign key makes it possible to create a parent-child relationship with the

tables. ● In this relationship, the parent table holds the initial column values, and column values of child table reference the parent column values.

● MySQL allows us to define a foreign key constraint on the child table.

MySQL defines the foreign key in two ways:

1. Using CREATE TABLE Statement

2. Using ALTER TABLE Statement

1. Using CREATE TABLE Statement:

Syntax:

[**CONSTRAINT** constraint_name]

FOREIGN KEY [foreign_key_name] (col_name, ...)

REFERENCES parent_tbl_name (col_name,...)

ON DELETE referenceOption

ON UPDATE referenceOption

MySQL contains different referential options, which are given below:

CASCADE: It is used when we delete or update any row from the parent table, the values of the matching rows in the child table will be deleted or updated automatically.

RESTRICT: It is used when we delete or update any row from the parent table that has a matching row in the reference(child) table, MySQL does not allow to delete or update rows in the parent table.

SET NULL: With this ON UPDATE and ON DELETE clauses option, if the referenced values in the parent table are deleted or modified, all related values in the child table are set to NULL value.

NO ACTION: When the ON UPDATE or ON DELETE clauses are set to NO ACTION, the performed update or delete operation in the parent table will fail with an error.

• create database data1;

• use dh1;

• create table demo1(id int,name varchar(66),primary key(id));

• insert into demo1 values(1,'Aditi');

• insert into demo1 values(2,'simran');

• select * from demo1;

• create table emp(e_id int,name varchar(88),id int,primary key(e_id),foreign key(id)references demo1(id)); • insert into emp values(11,'xyz',1);

• insert into emp values(13,'pqr',2);

• select * from emp;

• create table stud(s_id int,s_name varchar(88),age int,id int,e_id int,primary key(s_id),foreign key(id)references demo1(id),foreign key(e_id)references emp(e_id));

• insert into stud values(111,'amit',9,1,11);

• insert into stud values(112,'riya',9,1,11);

• select * from stud;

2. Using ALTER TABLE Statement:

ALTER TABLE Contact **ADD CONSTRAINT** fk_person

FOREIGN KEY (PERSON_ID) **REFERENCES** Person (PERSON_ ID) **ON DELETE CASCADE ON UPDATE RESTRICT**;

Table: Person

CREATE TABLE Person (

PERSON_ID **INT** NOT NULL AUTO_INCREMENT,

Name varchar(50) NOT NULL,

City **varchar**(50) NOT NULL,

PRIMARY KEY (PERSON_ID)

);

Table: Contact

CREATE TABLE Contact (

C_ID **INT**,

Person_Id **INT**,

Info **varchar**(50) NOT NULL,

Type **varchar**(50) NOT NULL

);

DROP Foreign Key:

Syntax: **(Perform both the steps one below the other) i.) ALTER TABLE** table_name **DROP FOREIGN KEY** fk_constraint_name(Constraint_name); **ii.) ALTER TABLE** table_name **Drop** key constraint_name;

For eg:

ALTER TABLE contact **DROP FOREIGN KEY** fk_customer;

ALTER TABLE contact **Drop** key fk_customer;

Note:

To find the constraint_name:

show create table table_name;

For e.g.:

show create table student;

Sub-queries

● A subquery is a SELECT statement which is used in another SELECT statement. ● Subqueries are very useful when you need to select rows from a table with a condition that depends on the data of the table itself.

● You can use the subquery in the SQL clauses including WHERE clause, HAVING clause, FROM clause etc.

● The subquery can also be referred as nested SELECT, sub SELECT or inner SELECT. ● In general, the subquery executes first and its output is used in the main query or outer query.

Guidelines for using a subquery:

● Enclose subqueries in ().

● Place subqueries on the right side of the comparison operator.

● Do not add an ORDER BY clause to a subquery.

● Use single-row operators with single-row subqueries. (<,>,<=,>=,<>)

● Use multiple-row operators with multiple-row subqueries (IN, ANY, ALL). ● We can write a sub query with in a subquery.

There are two types of subqueries:

1. Single Row Subqueries:

● The subquery returns only one row i.e It returns only one row from the inner select statement. ● Use single row comparison operators like =, > etc while doing comparisons.

2. **Multiple Row Subqueries:**

● The subquery returns more than one row i.e It returns more than one row from the inner select statement. ● Use multiple row comparison operators like IN, ANY, ALL in the comparisons.

1. SINGLE ROW SUBQUERIES:

When we want to find out the employees of an office in which GEORGE is working.

SELECT officeCode FROM employees

WHERE firstname = 'George';

Once it returns the office code (let us say 3) you would then give

SELECT firstName, lastName FROM employees

WHERE officeCode = 3;

This can be done using the subquery as follows:

SELECT firstName, lastName FROM employees

WHERE officeCode = (SELECT officeCode FROM employees

WHERE firstname = 'George');

1. Write a query to find the salary of employees whose salary is greater than the salary of employee whose id is 100?

SELECT EMPLOYEE_ID,

SALARY

FROM EMPLOYEES

WHERE SALARY >

(

SELECT SALARY

FROM EMPLOYEES

WHERE EMPLOYEED_ID = 100

2. Write a query to find the employees who all are earning the highest salary? SELECT EMPLOYEE_ID,

SALARY

FROM EMPLOYEES

WHERE SALARY =

(

SELECT MAX(SALARY)

FROM EMPLOYEES

)

3. Write a query to find the departments in which the least salary is greater than the highest salary in the department of id 200?

SELECT DEPARTMENT_ID,

MIN(SALARY)

FROM EMPLOYEES

GROUP BY DEPARTMENT_ID

HAVING MIN(SALARY) >

(

SELECT MAX(SALARY)

FROM EMPLOYEES

WHERE DEPARTMENT_ID = 200

)

Multiple Row Subqueries

● They are queries that return more than one row from the inner select statement.

● You may use the IN, ANY, or ALL operator in outer query to handle a subquery that returns multiple rows.

ID	NAME	CITY	SALARY	DEPT_NO	DESIGNATION
1	ADITI	THANE	30000	5	HR
2	JOHN	PUNE	40000	5	HR
3	SMITH	NAGPUR	25000	4	MANAGER

4	RAVI	MUMBAI	43000	4	ANALYST
5	RIYA	NAGPUR	38000	5	CLERK
6	TINA	MUMBAI	25000	5	ANALYST
7	MANISHA	THANE	25000	4	OPERATIONS
8	JAMES	PUNE	55000	1	CLERK

Subquery with 'IN':

Returns values equal to any member in the list.

E.g: Display all the employees who are in same office as 'Tom' or 'Martin'.

SELECT firstName, lastName FROM employees

WHERE officeCode IN (SELECT officeCode FROM employees

WHERE firstName IN ('Tom', 'Martin'));

SELECT Name,City FROM emp_info

WHERE dept_no IN (SELECT dept_no FROM emp_info

WHERE Name IN ('Tom', 'Martin'));

Subquery with 'ANY':

Returns values compared to each value returned by the subquery.

Select f_name from employee

where salary > Any (20000,25000,30000);

Note: > Any → 'More than minimum.

Select f_name from employee

where salary < Any (20000,25000,30000);

Note: < Any → 'Less than the maximum'.

Select f_name from employee

where salary > Any (Select salary from employee where dept_no=5);

Select f_name from employee

where salary < Any (Select salary from employee where dept_no=5);

Subquery with 'ALL':

Returns values compared to every value returned by the subquery.

Select f_name from employee

where salary > All (20000,25000,30000);

Note: > All→ 'More than maximum.

Select f_name from employee

where salary < All (20000,25000,30000);

Note: < All → 'Less than minimum.

Select f_name from employee

where salary > All (Select salary from employee where dept_no=5);

Select f_name from employee

where salary < All (Select salary from employee where dept_no=5);

1. Write a query to find the employees whose salary is equal to the salary of at least one employee in department of id 300?

SELECT EMPLOYEE_ID,

SALARY

FROM EMPLOYEES

WHERE SALARY IN

(

SELECT SALARY

FROM EMPLOYEES

WHERE DEPARTMENT_ID = 300

)

2. Write a query to find the employees whose salary is greater than at least on employee in department of id 500?

SELECT EMPLOYEE_ID,

```sql
SALARY

FROM EMPLOYEES

WHERE SALARY > ANY

(

SELECT SALARY

FROM EMPLOYEES

WHERE DEPARTMENT_ID = 500

)
```