Unit-3(SQL)

What is SQL?

SQL is a database language designed for the retrieval and management of data in a relational database.

SQL is the standard language for database management. All the RDBMS systems like MySQL, MS Access, Oracle, Sybase, Postgres, and SQL Server use SQL as their standard database language. SQL programming language uses various commands for different operations. +

Why Use SQL?

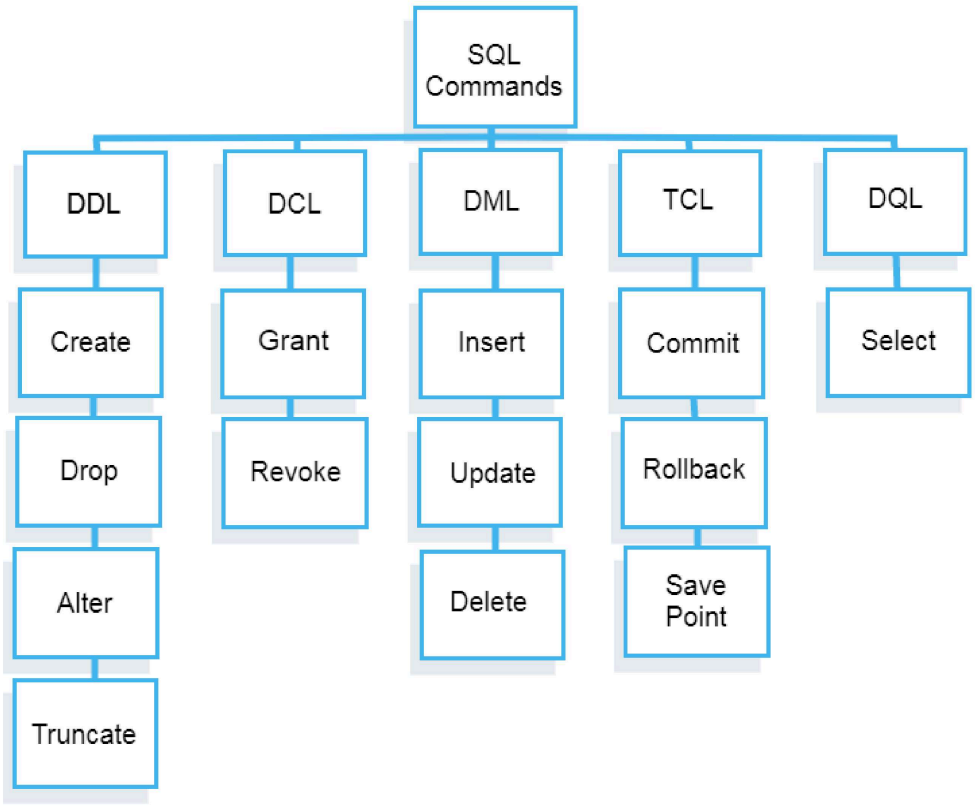
Here, are important reasons for using SQL

- It helps users to access data in the RDBMS system.
- It helps you to describe the data.
- It allows you to define the data in a database and manipulate that specific data.
- With the help of SQL commands in DBMS, you can create and drop databases and tables.
- SQL offers you to use the function in a database, create a view, and stored procedure.
- You can set permissions on tables, procedures, and views.

Types of SQL

Here are five types of widely used SQL queries.

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Control Language(DCL)
- Transaction Control Language(TCL)
- Data Query Language (DQL)



Types of SQL

Let see each of them in detail:

What is DDL?

Data Definition Language helps you to define the database structure or schema. Let's learn about DDL commands with syntax.

Five types of DDL commands in SQL are:

CREATE

CREATE statements is used to define the database structure schema:

Syntax:

CREATE TABLE TABLE_NAME (COLUMN_NAME DATATYPES[,....]);

For example:

Create database university;

Create table students;

Create view for_students;

DROP

Drops commands remove tables and databases from RDBMS.

Syntax

DROP TABLE TABLENAME;

For example:

Drop object_type object_name;

Drop database university;

Drop table student;

ALTER

Alters command allows you to alter the structure of the database.

Syntax:

To add a new column in the table

ALTER TABLE table_name ADD column_name COLUMN-definition;

To modify an existing column in the table:

ALTER TABLE MODIFY(COLUMN DEFINITION....);

For example:

Alter table guru99 add subject varchar;

TRUNCATE

This command used to delete all the rows from the table and free the space containing the table.

Syntax:

TRUNCATE TABLE table_name;

Example:

TRUNCATE table students;

What is Data Manipulation Language?

Data Manipulation Language (DML) allows you to modify the database instance by inserting, modifying, and deleting its data. It is responsible for performing all types of data modification in a database.

There are three basic constructs which allow database program and user to enter data and information are:

Here are some important DML commands in SQL:

- INSERT
- UPDATE
- DELETE

INSERT

This is a statement is a SQL query. This command is used to insert data into the row of a table.

Syntax:

INSERT INTO TABLE_NAME (col1, col2, col3,.... col N)

VALUES (value1, value2, value3, ... valueN);

Or

INSERT INTO TABLE_NAME

VALUES (value1, value2, value3, ... valueN);

For example:

INSERT INTO students (RollNo, FIrstName, LastName) VALUES ('60', 'Tom', Erichsen');

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]

For example:

UPDATE students

SET FirstName = 'Jhon', LastName= 'Wick'

WHERE StudID = 3;

DELETE

This command is used to remove one or more rows from a table.

Syntax:

DELETE FROM table_name [WHERE condition];

For example:

DELETE FROM students

WHERE FirstName = 'Jhon';

What is DCL?

DCL (Data Control Language) includes commands like GRANT and REVOKE, which are useful to give “rights & permissions.” Other permission controls parameters of the database system.

Examples of DCL commands

Commands that come under DCL:

- Grant
- Revoke

Grant

This command is use to give user access privileges to a database.

Syntax:

GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;

For example:

GRANT SELECT ON Users TO'Tom'@'localhost;

Revoke

It is useful to take back permissions from the user.

Syntax:

REVOKE privilege_nameON object_nameFROM {user_name |PUBLIC |role_name}

For example:

REVOKE SELECT, UPDATE ON student FROM BCA, MCA;

Defining constraints:

SQL Constraints

SQL constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

- NOT NULL - Ensures that a column cannot have a NULL value
- UNIQUE - Ensures that all values in a column are different
- PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- FOREIGN KEY - Prevents actions that would destroy links between tables
- CHECK - Ensures that the values in a column satisfies a specific condition
- DEFAULT - Sets a default value for a column if no value is specified
- CREATE INDEX - Used to create and retrieve data from the database very quickly

SQL NOT NULL Constraint

By default, a column can hold NULL values.

The NOT NULL constraint enforces a column to NOT accept NULL values.

This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

SQL NOT NULL on CREATE TABLE

The following SQL ensures that the "ID", "LastName", and "FirstName" columns will NOT accept NULL values when the "employee" table is created:

Example:

```
CREATE TABLE employee (
  ID int NOT NULL,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255) NOT NULL,
  Age int
);
```

To create a NOT NULL constraint on the "Age" column when the "employee" table is already created, use the following SQL:

SQL NOT NULL on ALTER TABLE

```
ALTER TABLE employee
MODIFY Age int NOT NULL;
```

SQL UNIQUE Constraint

The UNIQUE constraint ensures that all values in a column are different.

Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint.

However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

SQL UNIQUE Constraint on CREATE TABLE:

The following SQL creates a UNIQUE constraint on the "ID" column when the "employee" table is created:

```
CREATE TABLE employee (
   ID int NOT NULL,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255),
   Age int,
   CONSTRAINT UC_employee UNIQUE (ID,LastName)
);
```

SQL UNIQUE Constraint on ALTER TABLE

To create a UNIQUE constraint on the "ID" column when the table is already created, use the following SQL:

SQL UNIQUE Constraint on ALTER TABLE

To create a UNIQUE constraint on the "ID" column when the table is already created, use the following SQL:

Syntax:

```
ALTER TABLE employee
ADD CONSTRAINT UC_employee UNIQUE (ID,LastName);
```

DROP a UNIQUE Constraint

To drop a UNIQUE constraint, use the following SQL:

SQL PRIMARY KEY Constraint

The PRIMARY KEY constraint uniquely identifies each record in a table.

Primary keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

SQL PRIMARY KEY on CREATE TABLE

The following SQL creates a PRIMARY KEY on the "ID" column when the "Persons" table is created:

```
CREATE TABLE employee (
   ID int NOT NULL,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255),
   Age int,
   CONSTRAINT PK_employee PRIMARY KEY (ID,LastName)
);
```

SQL PRIMARY KEY on ALTER TABLE

To create a PRIMARY KEY constraint on the "ID" column when the table is already created, use the following SQL:

```
ALTER TABLE employee
ADD CONSTRAINT PK_employee PRIMARY KEY (ID,LastName);
```

DROP a PRIMARY KEY Constraint

To drop a PRIMARY KEY constraint, use the following SQL:

```
ALTER TABLE employee
DROP CONSTRAINT PK_employee;
```

SQL FOREIGN KEY Constraint

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.

The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

Look at the following two tables:

Persons Table

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

Orders Table

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.

The "PersonID" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.

The "PersonID" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

The FOREIGN KEY constraint prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the parent table.

SQL FOREIGN KEY on CREATE TABLE

The following SQL creates a FOREIGN KEY on the "PersonID" column when the "Orders" table is created:

MySQL:

```
CREATE TABLE Persons (
  ID int NOT NULL,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255) NOT NULL,
```

Age int
);

```
CREATE TABLE Orders (
   OrderID int NOT NULL,
   OrderNumber int NOT NULL,
   PersonID int,
   PRIMARY KEY (OrderID),
   FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

syntax:

```
CREATE TABLE Orders (
   OrderID int NOT NULL,
   OrderNumber int NOT NULL,
   PersonID int,
   PRIMARY KEY (OrderID),
   CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID)
   REFERENCES Persons(PersonID)
);
```

SQL FOREIGN KEY on ALTER TABLE

To create a FOREIGN KEY constraint on the "PersonID" column when the "Orders" table is already created, use the following SQL:

MySQL :

```
ALTER TABLE Orders
ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Orders
ADD CONSTRAINT FK_PersonOrder
FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

DROP a FOREIGN KEY Constraint

To drop a FOREIGN KEY constraint, use the following SQL:

syntax:

```
ALTER TABLE Orders
DROP FOREIGN KEY FK_PersonOrder;
```

SQL Server :

```
ALTER TABLE Orders
DROP CONSTRAINT FK_PersonOrder;
```

SQL CHECK Constraint

The CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a column it will allow only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

SQL CHECK on CREATE TABLE

The following SQL creates a CHECK constraint on the "Age" column when the "Persons" table is created. The CHECK constraint ensures that the age of a person must be 18, or older:

MySQL:

```
CREATE TABLE employee (
  ID int NOT NULL,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Age int,
```

```
    CHECK (Age>=18)
);
```

To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

```
CREATE TABLE employee (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255),
    CONSTRAINT CHK_employee CHECK (Age>=18 AND City='Sandnes')
);
```

SQL CHECK on ALTER TABLE

To create a CHECK constraint on the "Age" column when the table is already created, use the following SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE employee
ADD CHECK (Age>=18);
```

To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

```
ALTER TABLE employee
ADD CONSTRAINT CHK_employeeAge CHECK (Age>=18 AND City='Sandnes');
```

DROP a CHECK Constraint

To drop a CHECK constraint, use the following SQL:

```
ALTER TABLE employee
DROP CHECK CHK_employeeAge;
```

The SQL IN Operator

The IN operator allows you to specify multiple values in a WHERE clause.

The IN operator is a shorthand for multiple OR conditions.

Example

Return all customers from 'Germany', 'France', or 'UK'

```
SELECT * FROM Customers
WHERE Country IN ('Germany', 'France', 'UK');
```

Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

Below is a selection from the **Customers** table used in the examples:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds	Christina	Berguvsvägen	Luleå	S-958 22	Sweden

	snabbköp	Berglund	8

NOT IN

By using the NOT keyword in front of the IN operator, you return all records that are NOT any of the values in the list.

Example

Return all customers that are NOT from 'Germany', 'France', or 'UK':

```
SELECT * FROM Customers
WHERE Country NOT IN ('Germany', 'France', 'UK');
```

IN (SELECT)

You can also use IN with a subquery in the WHERE clause.

With a subquery you can return all records from the main query that are present in the result of the subquery.

Example

Return all customers that have an order in the **Orders** table:

```
SELECT * FROM Customers
WHERE CustomerID IN (SELECT CustomerID FROM Orders);
```

NOT IN (SELECT)

The result in the example above returned 74 records, that means that there are 17 customers that haven't placed any orders.

Let us check if that is correct, by using the NOT IN operator.

Example

Return all customers that have NOT placed any orders in the **Orders** table:

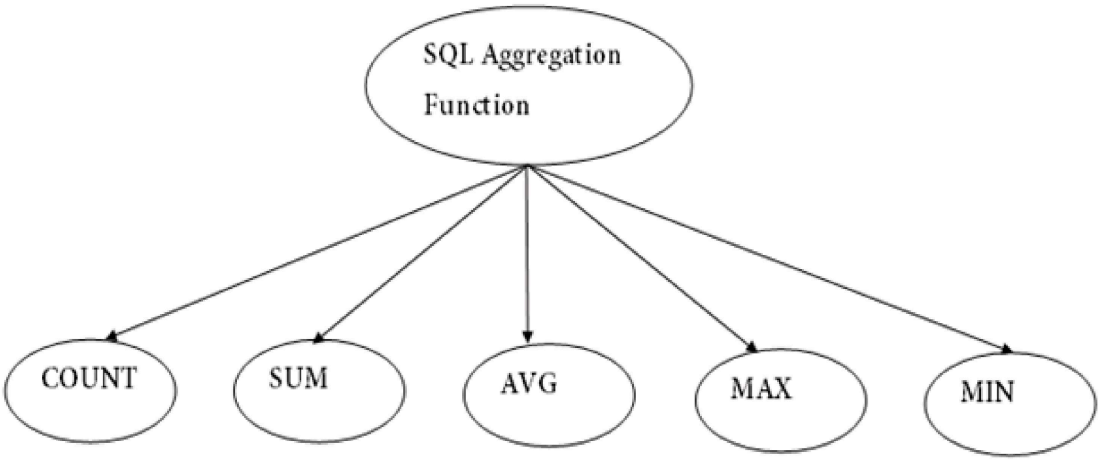
```
SELECT * FROM Customers
WHERE CustomerID NOT IN (SELECT CustomerID FROM Orders);
```

Functions

SQL Aggregate Functions

- SQL aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns a single value.
- It is also used to summarize the data.

Types of SQL Aggregation Function



1. COUNT FUNCTION

- COUNT function is used to Count the number of rows in a database table. It can work on both numeric and non-numeric data types.

- COUNT function uses the COUNT(*) that returns the count of all the rows in a specified table. COUNT(*) considers duplicate and Null.

Syntax

1. COUNT(*)
2. or
3. COUNT([ALL|DISTINCT] expression)

Sample table:

PRODUCT_MAST

PRODUCT	COMPANY	QTY	RATE	COST
Item1	Com1	2	10	20
Item2	Com2	3	25	75
Item3	Com1	2	30	60
Item4	Com3	5	10	50
Item5	Com2	2	20	40
Item6	Cpm1	3	25	75
Item7	Com1	5	30	150
Item8	Com1	3	10	30
Item9	Com2	2	25	50
Item10	Com3	4	30	120

Example: COUNT()

```
1. SELECT COUNT(*)
2. FROM PRODUCT_MAST;
```

Output:

10

Example: COUNT with WHERE

```
1. SELECT COUNT(*)
2. FROM PRODUCT_MAST;
3. WHERE RATE>=20;
```

Output:

7

Example: COUNT() with DISTINCT

```
1. SELECT COUNT(DISTINCT COMPANY)
2. FROM PRODUCT_MAST;
```

Output:

3

Example: COUNT() with GROUP BY

```
1. SELECT COMPANY, COUNT(*)
2. FROM PRODUCT_MAST
3. GROUP BY COMPANY;
```

Output:

Com1 5

Com2 3

Com3 2

Example: COUNT() with HAVING

```
1. SELECT COMPANY, COUNT(*)
```

2. FROM PRODUCT_MAST
3. GROUP BY COMPANY
4. HAVING COUNT(*)>2;

Output:

Com1 5

Com2 3

2. SUM Function

Sum function is used to calculate the sum of all selected columns. It works on numeric fields only.

Syntax

1. SUM()
2. or
3. SUM([ALL|DISTINCT] expression)

Example: SUM()

1. SELECT SUM(COST)
2. FROM PRODUCT_MAST;

Output:

670

Example: SUM() with WHERE

1. SELECT SUM(COST)
2. FROM PRODUCT_MAST
3. WHERE QTY>3;

Output:

320

Example: SUM() with GROUP BY

```
1. SELECT SUM(COST)
2. FROM PRODUCT_MAST
3. WHERE QTY>3
4. GROUP BY COMPANY;
```

Output:

Com1 150

Com2 170

Example: SUM() with HAVING

```
1. SELECT COMPANY, SUM(COST)
2. FROM PRODUCT_MAST
3. GROUP BY COMPANY
4. HAVING SUM(COST)>=170;
```

Output:

Com1 335

Com3 170

3. 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

```
1. AVG()
2. or
3. AVG( [ALL|DISTINCT] expression )
```

Example:

1. SELECT AVG(COST)
2. FROM PRODUCT_MAST;

Output:

67.00

4. 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

1. MAX()
2. or
3. MAX([ALL|DISTINCT] expression)

Example:

1. SELECT MAX(RATE)
2. FROM PRODUCT_MAST;

30

5. 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

1. MIN()
2. or
3. MIN([ALL|DISTINCT] expression)

Example:

1. SELECT MIN(RATE)
2. FROM PRODUCT_MAST;

Output:

10

SQL Server Built-in Functions

The following is the list of built-in String functions, DateTime functions, Numeric functions and conversion functions.

String Functions

Function	Description
ASCII	Returns the ASCII code value for the leftmost character of a character expression.
CHAR	Returns a character for an ASCII value.
CHARINDEX	Searches for one character expression within another character expression and returns the starting position of the first expression.
CONCAT	Concatenates two or more string values in an end to end manner and returns a single string.
LEFT	Returns a given number of characters from a character string starting from the left
LEN	Returns a specified number of characters from a character string.
LOWER	Converts a string to lower case.
LTRIM	Removes all the leading blanks from a character string.
NCHAR	Returns the Unicode character with the specified integer code, as defined by the Unicode standard.
PATINDEX	Returns the starting position of the first occurrence of the pattern in a given string.

Function	Description
REPLACE	Replaces all occurrences of a specified string with another string value.
RIGHT	Returns the right part of a string with the specified number of characters.
RTRIM	Returns a string after truncating all trailing spaces.
SPACE	Returns a string of repeated spaces.
STR	Returns character data converted from numeric data. The character data is right justified, with a specified length and decimal precision.
STUFF	Inserts a string into another string. It deletes a specified length of characters from the first string at the start position and then inserts the second string into the first string at the start position.
SUBSTRING	Returns part of a character, binary, text, or image expression
UPPER	Converts a lowercase string to uppercase.

DateTime Functions

Function	Description
CURRENT_TIMESTAMP	Returns the current system date and time of the computer on which the SQL server instance is installed. Time zone is not included.
DATEADD	Returns a new datetime value by adding an interval to the specified datepart of the specified date
DATEDIFF	Returns the difference in datepart between two given dates.
DATENAME	Returns a datepart as a character string.
DATEPART	Returns a datepart as an integer
DAY	Returns the Day as an integer representing the Day part of a specified date.
GETDATE	Returns a datetime value containing the date and time of the computer on which the SQL Server instance is installed. It does not include the time

Function	Description
	zone.
GETUTCDATE	Returns a datetime value in UTC format (Coordinated Universal Time), containing the date and time of the computer on which the SQL Server instance is installed.
MONTH	Returns the Month as an integer representing the Month part of a specified date.
YEAR	Returns the Year as an integer representing the Year part of a specified date.
ISDATE	Determines whether the input is a valid date, time or datetime value.

Numeric Functions

Function	Description
ABS	Returns the absolute value of a number.
AVG	Returns the average value of an expression/column values.
CEILING	Returns the nearest integer value which is larger than or equal to the specified decimal value.
COUNT	Returns the number of records in the SELECT query.
FLOOR	Returns the largest integer value that is less than or equal to a number. The return value is of the same data type as the input parameter.
MAX	Returns the maximum value in an expression.
MIN	Returns the minimum value in an expression.
RAND	Returns a random floating point value using an optional seed value.
ROUND	Returns a numeric expression rounded to a specified number of places right of the decimal point.
SIGN	Returns an indicator of the sign of the input integer expression.

Function	Description
SUM	Returns the sum of all the values or only the distinct values, in the expression. NULL values are ignored.

Set operations:

- Union The SQL Union operation is used to combine the result of two or more SQL SELECT queries.
- In the union operation, all the number of datatype and columns must be same in both the tables on which UNION operation is being applied.
- The union operation eliminates the duplicate rows from its result set.

Syntax

1. SELECT column_name FROM table1
2. UNION
3. SELECT column_name FROM table2;

Example:

The First table

ID	NAME
1	Jack
2	Harry
3	Jackson

The Second table

ID	NAME
3	Jackson
4	Stephan
5	David

Union SQL query will be:

```
1. SELECT * FROM First
2. UNION
3. SELECT * FROM Second;
```

The resultset table will look like:

ID	NAME
1	Jack
2	Harry
3	Jackson
4	Stephan
5	David

2. Union All

Union All operation is equal to the Union operation. It returns the set without removing duplication and sorting the data.

Syntax:

```
1. SELECT column_name FROM table1
2. UNION ALL
3. SELECT column_name FROM table2;
```

Example: Using the above First and Second table.

Union All query will be like:

```
1. SELECT * FROM First
2. UNION ALL
3. SELECT * FROM Second;
```

The resultset table will look like:

ID	NAME
1	Jack
2	Harry
3	Jackson
3	Jackson
4	Stephan
5	David

3. Intersect

- It is used to combine two SELECT statements. The Intersect operation returns the common rows from both the SELECT statements.
- In the Intersect operation, the number of datatype and columns must be the same.

- It has no duplicates and it arranges the data in ascending order by default.

Syntax

```
1. SELECT column_name FROM table1
2. INTERSECT
3. SELECT column_name FROM table2;
```

Example:

Using the above First and Second table.

Intersect query will be:

```
1. SELECT * FROM First
2. INTERSECT
3. SELECT * FROM Second;
```

The resultset table will look like:

ID	NAME
3	Jackson

4. Minus

- It combines the result of two SELECT statements. Minus operator is used to display the rows which are present in the first query but absent in the second query.
- It has no duplicates and data arranged in ascending order by default.

Syntax:

```
1. SELECT column_name FROM table1
2. MINUS
3. SELECT column_name FROM table2;
```

Example

Using the above First and Second table.

Minus query will be:

```
1. SELECT * FROM First
2. MINUS
3. SELECT * FROM Second;
```

The resultset table will look like:

ID	NAME
1	Jack
2	Harry

X:Cartesian product or cross product