UNIT - 3 SQL

SQL (Structured Query Language) is used to perform operations on the records stored in the database, such as updating records, inserting records, deleting records, creating and modifying database tables, views, etc.

SQL is not a database system, but it is a query language.

Suppose you want to perform the queries of SQL language on the stored data in the database. You are required to install any database management system in your systems, for example, Oracle, MySQL, MongoDB, PostgreSQL, SQL Server, DB2, etc.

What is SQL?

SQL is a short-form of the structured query language, and it is pronounced as S-Q-L or sometimes as See-Quell.

This database language is mainly designed for maintaining the data in relational database management systems. It is a special tool used by data professionals for handling structured data (data which is stored in the form of tables). It is also designed for stream processing in RDSMS.

You can easily create and manipulate the database, access and modify the table rows and columns, etc. This query language became the standard of ANSI in the year of 1986 and ISO in the year of 1987.

If you want to get a job in the field of data science, then it is the most important query language to learn. Big enterprises like Facebook, Instagram, and LinkedIn, use SQL for storing the data in the back-end.

Why SQL?

Nowadays, SQL is widely used in data science and analytics. Following are the reasons which explain why it is widely used:

- The basic use of SQL for data professionals and SQL users is to insert, update, and delete the data from the relational database.
- SQL allows the data professionals and users to retrieve the data from the relational database management systems.
- It also helps them to describe the structured data.
- It allows SQL users to create, drop, and manipulate the database and its tables.

- It also helps in creating the view, stored procedure, and functions in the relational database.
- It allows you to define the data and modify that stored data in the relational database.
- It also allows SQL users to set the permissions or constraints on table columns, views, and stored procedures.

Some SQL Commands

The SQL commands help in creating and managing the database. The most common SQL commands which are highly used are mentioned below:

1. CREATE command
2. UPDATE command
3. DELETE command
4. SELECT command
5. DROP command
6. INSERT command

CREATE Command

This command helps in creating the new database, new table, table view, and other objects of the database.

The CREATE DATABASE statement is used to create a new SQL database.

Syntax: CREATE DATABASE databasename;

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

Syntax: CREATE TABLE table_name (

column1 datatype,

column2 datatype,

column3 datatype, ...);

UNIT - 3 SQL

UPDATE Command

This command helps in updating or changing the stored data in the database.

The UPDATE statement is used to modify the existing records in a table.

Syntax: UPDATE table_name

SET column1 = value1, column2 = value2, ...

WHERE condition;

ALTER TABLE

ALTER TABLE changes the structure of a table. Here is how you would add a column to a database:

ALTER TABLE table_name

ADD column_name datatype;

DELETE Command

This command helps in removing or erasing the saved records from the database tables. It erases single or multiple tuples from the tables of the database.

The DELETE statement is used to delete existing records in a table.

DELETE FROM table_name WHERE condition;

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:
DELETE FROM table_name;

To delete the table completely, use the DROP TABLE statement:
DROP TABLE Customers;

UNIT - 3 SQL

SELECT Command

This command helps in accessing the single or multiple rows from one or multiple tables of the database. We can also use this command with the WHERE clause.

The SELECT statement is used to select data from a database.

Syntax: SELECT column1, column2, ...

FROM table_name;

DROP Command

This command helps in deleting the entire table, table view, and other objects from the database.

The DROP DATABASE statement is used to drop an existing SQL database.

DROP DATABASE databasename;

The DROP TABLE statement is used to drop an existing table in a database.

DROP TABLE table_name;

The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

TRUNCATE TABLE table_name;

INSERT Command

This command helps in inserting the data or records into the database tables. We can easily insert the records in single as well as multiple rows of the table.

The INSERT INTO statement is used to insert new records in a table.

INSERT INTO Syntax:

It is possible to write the INSERT INTO statement in two ways:

1. Specify both the column names and the values to be inserted:

INSERT INTO table_name (column1, column2, column3, ...)

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

2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows:

INSERT INTO table_name

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

—----------

DDL Commands in SQL

DDL is an abbreviation of **Data Definition Language**.

The DDL Commands in Structured Query Language are used to create and modify the schema of the database and its objects. The syntax of DDL commands is predefined for describing the data. The commands of Data Definition Language deal with how the data should exist in the database.

Following are the five DDL commands in SQL:

1. CREATE Command
2. DROP Command
3. ALTER Command
4. TRUNCATE Command
5. RENAME Command

CREATE Command

CREATE is a DDL command used to create databases, tables, triggers and other database objects.

Syntax to Create a Database:

Syntax: **CREATE Database** Database_Name;

Syntax to create a new table:

CREATE TABLE table_name

(column_Name1 data_type (size of the column) ,

column_Name2 data_type (size of the column) ,

column_Name3 data_type (size of the column) ,

...

column_NameN data_type (size of the column)) ;

Syntax to Create a new index:

CREATE INDEX Name_of_Index ON Name_of_Table (column_name_1 ,

column_name_2 , ... , column_name_N);

DROP Command

DROP is a DDL command used to delete/remove the database objects from the SQL database. We can easily remove the entire table, view, or index from the database using this DDL command.

Syntax to remove a database:

DROP DATABASE Database_Name;

Syntax to remove a table:

DROP TABLE Table_Name;

Syntax to remove an index:

DROP INDEX Index_Name;

ALTER Command

ALTER is a DDL command which changes or modifies the existing structure of the database, and it also changes the schema of database objects.

We can also add and drop constraints of the table using the ALTER command.

Syntax to add a newfield in the table:

ALTER TABLE name_of_table ADD column_name column_definition;

Syntax to remove a column from the table:

ALTER TABLE name_of_table DROP Column_Name_1 , column_Name_2 , ..., column_Name_N;

Syntax to modify the column of the table:

ALTER TABLE table_name MODIFY (column_name column_datatype(size));

TRUNCATE Command

TRUNCATE is another DDL command which deletes or removes all the records from the table.

This command also removes the space allocated for storing the table records.

Syntax of TRUNCATE command

TRUNCATE TABLE Table_Name;

RENAME Command

RENAME is a DDL command which is used to change the name of the database table.

Syntax of RENAME command

RENAME TABLE Old_Table_Name TO New_Table_Name;

DML Commands in SQL

DML is an abbreviation of Data Manipulation Language.

The DML commands in Structured Query Language change the data present in the SQL database. We can easily access, store, modify, update and delete the existing records from the database using DML commands.

Following are the four main DML commands in SQL:

1. SELECT Command
2. INSERT Command

3. UPDATE Command
4. DELETE Command

SELECT DML Command

SELECT is the most important data manipulation command in Structured Query Language. The SELECT command shows the records of the specified table. It also shows the particular record of a particular column by using the WHERE clause.

Syntax of SELECT DML command

SELECT column_Name_1, column_Name_2, ..., column_Name_N FROM Name_of_table;

Here, column_Name_1, column_Name_2, ….., column_Name_N are the names of those columns whose data we want to retrieve from the table.

If we want to retrieve the data from all the columns of the table, we have to use the following SELECT command:

*SELECT * FROM table_name;*

INSERT DML Command

INSERT is another most important data manipulation command in Structured Query Language, which allows users to insert data in database tables.

Syntax of INSERT Command

INSERT INTO TABLE_NAME (column_Name1 , column_Name2 , column_Name3 , ... column_NameN) VALUES (value_1, value_2, value_3, ... value_N) ;

UPDATE DML Command

UPDATE is another most important data manipulation command in Structured Query Language, which allows users to update or modify the existing data in database tables.

Syntax of UPDATE Command

UPDATE Table_name SET [column_name1= value_1, ..., column_nameN = value_N] WHERE CONDITION;

Here, 'UPDATE', 'SET', and 'WHERE' are the SQL keywords, and 'Table_name' is the name of the table whose values you want to update.

DELETE DML Command

DELETE is a DML command which allows SQL users to remove single or multiple existing records from the database tables.

This command of Data Manipulation Language does not delete the stored data permanently from the database. We use the WHERE clause with the DELETE command to select specific rows from the table.

Syntax of DELETE Command

DELETE FROM Table_Name WHERE condition;

Data Control Language

DCL commands are used to grant and take back authority from any database user.

Here are some commands that come under DCL:

- Grant
- Revoke

a. Grant: It is used to give user access privileges to a database**.**

Example

GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;

b. Revoke: It is used to take back permissions from the user.

Example

REVOKE SELECT, UPDATE ON MY_TABLE FROM USER1, USER2;

UNIT - 3 SQL

SQL CREATE TABLE

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

If you want to create a table, you should name the table and define its column and each column's data type.

Let's see the simple syntax to create the table.

create table "tablename"

("column1" "data type",

"column2" "data type",

"column3" "data type",

...

"columnN" "data type");

Create a Table using another table

We can create a copy of an existing table using the create table command. The new table gets the same column signature as the old table. We can select all columns or some specific columns.

If we create a new table using an old table, the new table will be filled with the existing value from the old table.

The basic syntax for creating a table with the other table is:

CREATE TABLE table_name AS

SELECT column1, column2,...

FROM old_table_name WHERE ..... ;

The following SQL creates a copy of the employee table.

CREATE TABLE EmployeeCopy AS

SELECT EmployeeID, FirstName, Email

FROM Employee;

UNIT - 3 SQL

SQL ALTER TABLE

The ALTER TABLE statement in Structured Query Language 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.

ALTER TABLE ADD Column statement in SQL

In many situations, you may require to add the columns in the existing table. Instead of creating a whole table or database again you can easily add single and multiple columns using the ADD keyword.

Syntax of ALTER TABLE ADD Column statement in SQL

ALTER TABLE table_name ADD column_name column-definition;

The above syntax only allows you to add a single column to the existing table. If you want to add more than one column to the table in a single SQL statement, then use the following **syntax:**

ALTER TABLE table_name

ADD (column_Name1 column-definition,

column_Name2 column-definition,

.....

column_NameN column-definition);

ALTER TABLE MODIFY Column statement in SQL

The MODIFY keyword is used for changing the column definition of the existing table.

Syntax of ALTER TABLE MODIFY Column statement in SQL

ALTER TABLE table_name MODIFY column_name column-definition;

This syntax only allows you to modify a single column of the existing table. If you want to modify more than one column of the table in a single SQL statement, then use the following syntax.

ALTER TABLE table_name

MODIFY (column_Name1 column-definition,

column_Name2 column-definition,

.....

column_NameN column-definition);

ALTER TABLE DROP Column statement in SQL

In many situations, you may require to delete the columns from the existing table. Instead of deleting the whole table or database you can use DROP keyword for deleting the columns.

Syntax of ALTER TABLE DROP Column statement in SQL

ALTER TABLE table_name DROP Column column_name ;

ALTER TABLE RENAME Column statement in SQL

The RENAME keyword is used for changing the name of columns or fields of the existing table.

Syntax of ALTER TABLE RENAME Column statement in SQL

ALTER TABLE table_name RENAME COLUMN old_name to new_name;

DEFINING CONSTRAINTS:

Primary Key in DBMS

There are certain keys in DBMS that are used for different purposes, from which the most commonly known is the Primary Key.

Here, in this section, we will look at the Primary key - What it is, what is the use of a primary key, and we will also implement some examples to understand that how a primary key works.

What is a Primary Key

A Primary Key is the minimal set of attributes of a table that has the task to uniquely identify the rows, or we can say the tuples of the given particular table.

A primary key of a relation is one of the possible candidate keys which the database designer thinks it's primary. It may be selected for convenience, performance and many other reasons. The choice of the possible primary key from the candidate keys depend upon the following conditions.

Syntax for creating primary key constraint:

The primary key constraint can be defined at the column level or table level.

At column level:

<column_name><datatype> Primary key;

Removing Primary Key

It is also possible to delete the set primary key from an attribute using ALTER and DROP commands.

ALTER TABLE STUDENT_DETAIL DROP PRIMARY KEY ;

Adding Primary Key after creating the table

In order to set the primary key after creating a table, use the ALTER command and add the primary key constraint to do so. The syntax is shown below:

ALTER TABLE STUDENT_DETAIL
ADD CONSTRAINT PK_STUDENT_DETAIL PRIMARY KEY (Roll_no, Name);

We have taken the Name attribute just for understanding the syntax.

SQL FOREIGN KEY constraint ON CREATE TABLE:

(Defining a foreign key constraint on single column)

To create a foreign key on the "S_Id" column when the "Orders" table is created:

```
CREATE TABLE orders
(  O_Id int NOT NULL,
Order_No  int NOT NULL,
S_Id int,
PRIMARY KEY (O_Id),
FOREIGN KEY (S_Id) REFERENCES Persons (S_Id)  )
```

SQL FOREIGN KEY constraint for ALTER TABLE:

If the Order table is already created and you want to create a FOREIGN KEY constraint on the "S_Id" column, you should write the following syntax:

Defining a foreign key constraint on single column:

```
ALTER TABLE Orders
ADD CONSTRAINT fk_PerOrders
FOREIGN KEY(S_Id)
REFERENCES Students (S_Id)
```

DROP SYNTAX for FOREIGN KEY CONSTRAINT:

If you want to drop a FOREIGN KEY constraint, use the following syntax:

```
ALTER TABLE Orders
DROP FOREIGN KEY fk_PerOrders

ALTER TABLE Orders
DROP CONSTRAINT fk_PerOrders
```

Unique Key in SQL

A unique key is a set of one or more than one fields/columns of a table that uniquely identify a record in a database table.

You can say that it is little like primary key but it can accept only one null value and it cannot have duplicate values.

The unique key and primary key both provide a guarantee for uniqueness for a column or a set of columns.

There is an automatically defined unique key constraint within a primary key constraint.

There may be many unique key constraints for one table, but only one PRIMARY KEY constraint for one table.

SQL UNIQUE KEY constraint on CREATE TABLE:

If you want to create a UNIQUE constraint on the "S_Id" column when the "students" table is created, use the following SQL syntax:

CREATE TABLE students

(S_Id int NOT NULL UNIQUE,

LastName varchar (255) NOT NULL,

FirstName varchar (255),

City varchar (255))

——----------

CREATE TABLE students

(S_Id int NOT NULL,

LastName varchar (255) NOT NULL,

FirstName varchar (255),

City varchar (255),

UNIQUE (S_Id))

——----------

Defining a unique key constraint on multiple columns:

CREATE TABLE students

(S_Id int NOT NULL,

LastName varchar (255) NOT NULL,

FirstName varchar (255),

City varchar (255),

CONSTRAINT uc_studentId UNIQUE (S_Id, LastName))

——----------

SQL UNIQUE KEY constraint on ALTER TABLE:

If you want to create a unique constraint on "S_Id" column when the table is already created, you should use the following SQL syntax:

Defining a unique key constraint on single column:

ALTER TABLE students

ADD UNIQUE (S_Id)

Defining a unique key constraint on multiple columns:

ALTER TABLE students

ADD CONSTRAINT uc_StudentId UNIQUE (S_Id, LastName)

DROP SYNTAX FOR A FOREIGN KEY constraint:

If you want to drop a UNIQUE constraint, use the following SQL syntax:

ALTER TABLE students

DROP INDEX uc_studentID

——----------

ALTER TABLE students

DROP CONSTRAINT uc_studentID

NOT NULL CONSTRAINT:

If you don't want to have a null column or a null value you need to define constraints like NOT NULL. NOT NULL constraints make sure that a column does not hold null values, or in other words, NOT NULL constraints make sure that you cannot insert a new record or update a record without entering a value to the specified column(i.e., NOT NULL column).

It prevents for acceptance of NULL values. It can be applied to column-level constraints.

Syntax:

CREATE TABLE table_Name

(column1 data_type(size) NOT NULL,

column2 data_type(size) NOT NULL,….);

SQL NOT NULL on ALTER Table

We can also add a NOT NULL constraint in the existing table using the ALTER statement. For example, if the EMPLOYEES table has already been created then add NOT NULL constraints to the "Name" column using ALTER statements in SQL as follows:

Query: ALTER TABLE Emp modify Name Varchar(50) NOT NULL;

Check Constraint:

Let us see a few practical examples to understand this concept more clearly. We will use the MariaDB database for writing all the queries.

Check constraint on column level:

To apply a check constraint on a column level, we must specify the check constraint just after the column name.

Syntax:

CREATE TABLE TableName(ColumnName1 Datatype(size), ColumnName2 Datatype(size), ColumnName3 Datatype(size) Constraint ConstraintName CHECK(ColumnName CONDITION Value),..., ColumnNameN Datatype(size));

We will write a query to create a table and specify a column-level check constraint on more than one column.

CREATE TABLE students(S_ID INT NOT NULL, S_Name VARCHAR(40), Hometown VARCHAR(20), Percentage INT NOT NULL CHECK(Percentage >90), Favourite_Subject VARCHAR(20) NOT NULL CHECK(Favourite_Subject IN('Science', 'Maths', 'English')), PRIMARY KEY (S_ID));

In the above query, we have specified the CHECK constraint on the Percentage and Favourite_Subject column. According to this constraint, the Percentage column will allow only those records to be inserted where the percentage secured by students is above 90, and the favourite subject of the student is either Science, Maths or English.

Check constraint on table level:

To apply a check constraint on a table level, we must specify the check constraint before ending the table creation.

Syntax:

CREATE TABLE TableName(ColumnName1 Datatype(size), ColumnName2 Datatype(size), ColumnName3 Datatype(size),..., ColumnNameN Datatype(size), Constraint ConstraintName CHECK(ColumnName CONDITION Value));

We will write a query to create a table and specify a table-level check constraint on more than one column.

CREATE TABLE student(S_ID INT NOT NULL, S_Name VARCHAR(40), Hometown VARCHAR(20), Percentage INT NOT NULL, Favourite_Subject VARCHAR(20) NOT NULL, CONSTRAINT Constraint_Percentage (Percentage >90), CONSTRAINT Constraint_Fav_sub CHECK(Favourite_Subject IN('Science', 'Maths', 'English')), PRIMARY KEY (S_ID));

Check constraint after table creation:

A situation may arise when we need to apply a check constraint on a column after the table creation. In such cases, we have to use the ALTER command to apply the check constraint on an already created table.

Syntax:

ALTER TABLE TableName ADD CONSTRAINT ConstraintName CHECK(ColumnName CONDITION Value);

Suppose we created an employ table without any constraints, and later we decided to add a constraint on one of the columns. Then we will execute the following query:

ALTER TABLE employ ADD CONSTRAINT Constraint_Age CHECK (Date_of_Joining <= "2019-01-01");

Remove a check constraint

Suppose we have a check constraint created on the table's column. Later, we decided to remove that constraint from the column. Then, in such a situation, we will use the ALTER command to remove the check constraint.

Syntax:

ALTER TABLE TableName DROP CHECK ConstraintName;

—----------

SQL IN Operator

- IN is an operator in SQL, which is generally used with a WHERE clause.
- Using the IN operator, multiple values can be specified.
- It allows us to easily test if an expression matches any value in a list of values.
- IN operator is used to replace many OR conditions.

Syntax of IN operator in SQL:

SELECT ColumnName FROM TableName WHERE ColumnName IN (Value 1, Value 2,...., ValueN);

Write a query to display all the records from the t_students table where the hometown of the students is one of the following places: Faridabad, Panipat, or Jaipur.

*Query: mysql> SELECT *FROM t_students WHERE Hometown IN ("Faridabad", "Panipat", "Jaipur");*

Write a query to display all the records from the t_students table where the favourite subject of the students is one of the following subjects: History, Biology, Physics or Chemistry.

*Query: mysql> SELECT *FROM t_students WHERE Favourite_Subject IN ("History", "Biology", "Physics", "Chemistry");*

UNIT - 3 SQL

Write a query to display all the records from the t_students table where the percentage secured by the student is one of the following values: 78, 88, 89, 90, or 92.

*Query: mysql> SELECT *FROM t_students WHERE Percentage IN (78, 88, 89, 90, 92);*

Write a query to display all the records from the employee table where the date of birth of an employee is one of the following dates: 1999-01-10, 1989-01-09, 1993-03-05, or 1993-05-03.

*Query: mysql> SELECT *FROM employee WHERE Date_of_Birth IN ("1999-01-10", "1989-01-09", "1993-03-05", "1993-05-03");*

Write a query to display all the records from the employee table where the job location of an employee is among one of the following places: Nashik, Surat, Noida, Delhi, or Pune.

*Query: mysql> SELECT *FROM employee WHERE Job_location IN ("Nashik", "Surat", "Noida", "Delhi", "Pune");*

Write a query to display all the records from the employee table where the salary of an employee is among one of the following values: 60000, 53000, 30000, or 45000.

*Query: mysql> SELECT *FROM employee WHERE Salary IN (60000, 53000, 30000, 45000);*

SQL Aggregate Functions:

In this article let us explore a new topic in SQL called aggregate functions. First let us understand what is aggregate function and how does it actually work.

What is aggregate function?

Aggregate functions in SQL are unique functions that work on a group of rows in a table and produce a single value as a result. These operations are used to calculate a set of numbers and produce summary statistics like sum, count, average, maximum, and minimum. SQL queries frequently employ aggregate procedures to condense data from one or more tables.

UNIT - 3 SQL

After grouping and aggregating, aggregate functions can also be used with the HAVING or WHERE clause to further filter the data. A condition involving an aggregate function can be used to filter the results of a query using the HAVING or WHERE clause.

Below are a few frequently used aggregate functions in SQL. Let us understand each of these functions with the help of various examples.

1. *Count()*
2. *Sum()*
3. *Avg()*
4. *Min()*
5. *Max()*

1. COUNT():

This function returns the number of records(rows) in a table. The Syntax of the count() function is mentioned below.

Syntax: SELECT COUNT(column_name) FROM table_name WHERE condition;

2. SUM ():

This function returns the sum of all values of a column in a table. Here is the syntax for the sum() function.

Syntax: SELECT SUM(column_name) FROM table_name WHERE condition;

3. AVG ()

This function will return the average of all values present in a column. The syntax of the AVG() function is given below.

Syntax: SELECT AVG(column_name) FROM table_name WHERE condition;

4. MIN():

This function produces the lowest value in a column for a group of rows that satisfy a given criterion. The Syntax of the MIN() function is as follows

Syntax: SELECT MIN(column_name) FROM table_name WHERE condition;

5. MAX()

The MAX function in SQL is used to return the highest value in a column for a group of rows that satisfy a given condition in a table. The MAX syntax is as follows:

Syntax: SELECT MAX(column_name) FROM table_name WHERE condition;

—----------

SQL Server Built-in Functions

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

String Functions

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.*
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

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 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

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

What are SQL Set Operators?

A set operator in SQL is a keyword that lets you combine the results of two queries into a single query.

Sometimes when working with SQL, you'll have a need to query data from two more tables. But instead of joining these two tables, you'll need to list the results from both tables in a single result, or in different rows. That's what set operators do.

The Different Types of Set Operators

There are a few different set operators that can be used, depending on your needs, and which database vendor you're using.

The different set operators are:

- UNION
- UNION ALL
- MINUS
- INTERSECT
- EXCEPT

Using Set Operators

Set operators are used like this:

SELECT your_select_query
set_operator
SELECT another_select_query;

It uses two (or more) SELECT queries, with a set operator in the middle.

There are a few things to keep in mind though.

When selecting your columns, the number of columns needs to match between queries, and the data type of each column needs to be compatible.

So, if you select three columns in the first query, you need to select three columns in the second query. The data types also need to be compatible, so if you select a number and two character types in the first query, you need to do the same in the second query.

Also, if you want to order your results, the ORDER BY must go at the end of the last query. You can't add ORDER BY inside each SELECT query before the set operator.

UNION: Combining Results

The UNION keyword or set operator will allow you to combine the results of two queries. It removes any duplicate results and shows you the combination of both.

If we wanted to select from both the employee and customer tables, using UNION, our query would look like this:

SELECT first_name, last_name
FROM customer
UNION
SELECT first_name, last_name
FROM employee;

This shows us all customer and employee records.

UNION ALL: Combining Results in a Different Way

The UNION ALL set operator also combines the results from two queries.

It's very similar to UNION, but it does not remove duplicates.

Let's see an example. If we wanted to select from both the employee and customer tables, using UNION ALL, our query would look like this:

```
SELECT first_name, last_name
FROM customer
UNION ALL
SELECT first_name, last_name
FROM employee;
```

MINUS or EXCEPT: Finding Results That Are Missing

Another set operator we can use is the MINUS keyword or the EXCEPT keyword.

The MINUS set operator will return results that are found in the first query specified that don't exist in the second query.

The EXCEPT keyword is similar to MINUS, but is available in SQL Server and other databases.

Using our example data, we could use the MINUS set operator to find all names in the customer table that don't exist in the employee table.

Our query would look like this:

```
SELECT first_name, last_name
FROM customer
```

MINUS

SELECT first_name, last_name

FROM employee;

INTERSECT: Showing Common Results

The INTERSECT keyword allows you to find results that exist in both queries. Two SELECT statements are needed, and any results that are found in both of them are returned if INTERSECT is used.

Using our example data, we could use the INTERSECT set operator to find all names in the customer table that don't exist in the employee table.

Our query would look like this:

SELECT first_name, last_name

FROM customer

INTERSECT

SELECT first_name, last_name

FROM employee;