1)ADD

- add a column in an existing table
- ```
  ALTER TABLE Customers
  ADD Email varchar(255);
  ```

2)ADD CONSTRAINT

- adds a constraint named "PK_Person" that is a PRIMARY KEY constraint on multiple columns (ID and LastName)
- ```
  ALTER TABLE Persons
  ADD CONSTRAINT PK_Person PRIMARY KEY (ID,LastName);
  ```

3)ALTER

- adds, deletes, or modifies columns in a table
- 1)Adding a column
- ```
  ALTER TABLE Customers
  ADD Email varchar(255);
  ```
- 2)Dropping a column
- ```
  ALTER TABLE Customers
  DROP COLUMN Email;
  ```
- 3)Changes the datatype of the column
- ```
  ALTER TABLE Employees
  ALTER COLUMN BirthDate year;
  ```

4)ALL

- SQL statement returns TRUE and lists the productnames if ALL the records in the OrderDetails table has quantity = 10
- ```
  SELECT ProductName
  FROM Products
  WHERE ProductID
  = ALL (SELECT ProductID FROM OrderDetails WHERE Quantity = 10);
  ```

5)AND

- selects all fields from "Customers" where country is "Germany" AND city is "Berlin"
- ```
  SELECT * FROM Customers
  WHERE Country='Germany' AND City='Berlin';
  ```

6)ANY
  ```

- returns true if any of the subquery values meet the condition
- ```
  SELECT ProductName
  FROM Products
  WHERE ProductID
  = ANY (SELECT ProductID FROM OrderDetails WHERE Quantity = 10);
  ```

7)AS

- rename a column or table with an alias
- ```
  SELECT CustomerID AS ID, CustomerName AS Customer
  FROM Customers;
  ```
- it requires double quotation marks or square brackets if the alias name contains spaces
- ```
  SELECT CustomerName AS Customer, ContactName AS [Contact Person]
  FROM Customers;
  ```
- ```
  SELECT CustomerName, Address + ', ' + PostalCode + ' ' + City + ',
  ' + Country AS Address
  FROM Customers;
  ```
- ```
  SELECT o.OrderID, o.OrderDate, c.CustomerName
  FROM Customers AS c, Orders AS o
  WHERE c.CustomerName="Around the
  Horn" AND c.CustomerID=o.CustomerID;
  ```

8)ASC

- sort the data returned in ascending order
- ```
  SELECT * FROM Customers
  ORDER BY CustomerName ASC;
  ```

9)BACKUP DATABASE

- creates a full back up of the existing database
- ```
  BACKUP DATABASE testDB
  TO DISK = 'D:\backups\testDB.bak';
  ```
- differential back up only backs up the parts of the database that have changed since the last full database backup
- ```
  BACKUP DATABASE testDB
  TO DISK = 'D:\backups\testDB.bak'
  WITH DIFFERENTIAL;
  ```

10)BETWEEN KEYWORD

- ```
  SELECT * FROM Products
  WHERE Price BETWEEN 10 AND 20;
  ```
- ```
  SELECT * FROM Products
  WHERE Price NOT BETWEEN 10 AND 20;
  ```

- ```
  SELECT * FROM Products
  WHERE ProductName BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di
  Giovanni'
  ORDER BY ProductName;
  ```

11)CASE KEYWORD:

- CASE command is used is to create different output based on conditions

```
SELECT OrderID, Quantity,
CASE
    WHEN Quantity > 30 THEN "The quantity is greater than 30"
    WHEN Quantity = 30 THEN "The quantity is 30"
    ELSE "The quantity is under 30"
END AS QUANTITYTEXT
FROM OrderDetails;
```

- SQL will order the customers by City. However, if City is NULL, then order by Country
- ```
  SELECT CustomerName, City, Country
  FROM Customers
  ORDER BY
  (CASE
      WHEN City IS NULL THEN Country
      ELSE City
  END);
  ```

12) CHECK KEYWORD

- limits the value that can be placed in a column
- CHECK constraint on the "Age" column when the "Persons" table is created
- ```
  CREATE TABLE Persons (
      Age int,
      CHECK (Age>=18)
  );
  ```
- defining a CHECK constraint on multiple columns
- ```
  CREATE TABLE Persons (
      Age int,
      City varchar(255),
      CONSTRAINT CHK_Person CHECK (Age>=18 AND City='Sandnes')
  );
  ```
- create a CHECK constraint on the "Age" column when the table is already created

- ```
  ALTER TABLE Persons
  ADD CHECK (Age>=18);
  ```
- defining a CHECK constraint on multiple columns
- ```
  ALTER TABLE Persons
  ADD CONSTRAINT CHK_PersonAge CHECK (Age>=18 AND City='Sandnes');
  ```
- drop a CHECK constraint
- ```
  ALTER TABLE Persons
  DROP CHECK CHK_PersonAge;
  ```

13)COLUMN KEYWORD

- used to change the data type of a column in a table
- ```
  ALTER TABLE Employees
  ALTER COLUMN BirthDate year;
  ```
- ```
  ALTER TABLE Customers
  DROP COLUMN ContactName;
  ```

14)CONSTRAINT KEYWORD

- `ADD CONSTRAINT` command is used to create a constraint after a table is already created
- ```
  ALTER TABLE Persons
  ADD CONSTRAINT PK_Person PRIMARY KEY (ID,LastName);
  ```
- `DROP CONSTRAINT` command is used to delete a UNIQUE, PRIMARY KEY, FOREIGN KEY, or CHECK constraint
- **DROP A UNIQUE CONSTRAINT:**
- ```
  ALTER TABLE Persons
  DROP INDEX UC_Person;
  ```
- **DROP A PRIMARY KEY:**
- ```
  ALTER TABLE Persons
  DROP PRIMARY KEY;
  ```
- **DROP A FOREIGN KEY:**
- ```
  ALTER TABLE Orders
  DROP FOREIGN KEY FK_PersonOrder;
  ```
- **DROP A CHECK CONSTRAINT:**
- ```
  ALTER TABLE Persons
  DROP CHECK CHK_PersonAge;
  ```

15)CREATE KEYWORD

- **CREATE DATABASE:**
- ```
  CREATE DATABASE testDB;
  ```
- **TO CHECK THE DATABASES WHICH WE HAVE:**`SHOW DATABASES;`
  ```

- **CREATE TABLE:**
- ```
  CREATE TABLE Persons (
      PersonID int,
      LastName varchar(255),
      FirstName varchar(255),
      Address varchar(255),
      City varchar(255)
  );
  ```
- **CREATE TABLE USING ANOTHER TABLE:**
- ```
  CREATE TABLE TestTable AS
  SELECT customername, contactname
  FROM customers;
  ```
- **TO CREATE AN INDEX ON THE COLUMN: (ALLOW DUPLICATE VALUES)**
- ```
  CREATE INDEX idx_pname
  ON Persons (LastName, FirstName);
  ```
- UPDATING A TABLE WITH INDEX TAKES SOME AMOUNT OF TIME SO GIVE INDEX TO A SPECIFIC USUAL COLUMNS
- **CREATE A UNIQUE INDEX:**
- ```
  CREATE UNIQUE INDEX uidx_pid
  ON Persons (PersonID);
  ```
- **CREATE VIEW**
- view is a virtual table based on the result set of an SQL statement
- ```
  CREATE VIEW [Brazil Customers] AS
  SELECT CustomerName, ContactName
  FROM Customers
  WHERE Country = "Brazil";
  ```
- `CREATE OR REPLACE VIEW` command updates a view
- ```
  CREATE OR REPLACE VIEW [Brazil Customers] AS
  SELECT CustomerName, ContactName, City
  FROM Customers
  WHERE Country = "Brazil";
  ```
- **DISPLAY THE VIEW**
- ```
  SELECT * FROM [Brazil Customers];
  ```
- **CREATING A PROCEDURE:**
- Sql code can be saved so that it can be reused again and again
- ```
  CREATE PROCEDURE SelectAllCustomers
  AS
  SELECT * FROM Customers
  GO;
  ```
- To execute the procedure
- ```
  EXEC SelectAllCustomers;
  ```
- **DROP A DATABASE**
- ```
  DROP DATABASE testDB;
  ```

16)DEFAULT KEYWORD:

- default value will be added to all new records if no other value is specified
- sets a DEFAULT value for the "City" column when the "Persons" table is created
- ```
  CREATE TABLE Persons (
      City varchar(255) DEFAULT 'Sandnes'
  );
  ```
- DEFAULT ON ALTER TABLE:
- ```
  ALTER TABLE Persons
  ALTER City SET DEFAULT 'Sandnes';
  ```
- DROP A DEFAULT
- ```
  ALTER TABLE Persons
  ALTER City DROP DEFAULT;
  ```

17)DELETE KEYWORD

- The `DELETE` command is used to delete existing records in a table
- ```
  DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
  ```
- If you omit the WHERE clause, all records in the table will be deleted!
- ```
  DELETE FROM Customers;
  ```

18)DESC KEYWORD

- sort the data returned in descending order
- ```
  SELECT * FROM Customers
  ORDER BY CustomerName DESC;
  ```

19)DISTINCT KEYWORD

- returns only distinct (different) values in the result set
- ```
  SELECT DISTINCT Country FROM Customers;
  ```

20)DROP KEYWORD

- ```
  ALTER TABLE Customers
  DROP COLUMN ContactName;
  ```
- ```
  ALTER TABLE Persons
  DROP INDEX UC_Person;
  ```
- ```
  ALTER TABLE Persons
  DROP PRIMARY KEY;
  ```

- ```
  ALTER TABLE Orders
  DROP FOREIGN KEY FK_PersonOrder;
  ```
- ```
  ALTER TABLE Persons
  DROP CHECK CHK_PersonAge;
  ```
- ```
  ALTER TABLE Persons
  ALTER City DROP DEFAULT;
  ```
- ```
  DROP DATABASE testDB;
  ```
- ```
  DROP TABLE Shippers;
  ```
- ```
  DROP VIEW [Brazil Customers];
  ```

21)EXEC KEYWORD

- execute a stored procedure
- ```
  EXEC SelectAllCustomers;
  ```

22)SQL EXISTS KEYWORD

- tests for the existence of any record in a subquery
- ```
  SELECT SupplierName
  FROM Suppliers
  WHERE EXISTS (SELECT ProductName FROM Products WHERE SupplierId =
  Suppliers.supplierId AND Price < 20);
  ```

23)SQL FOREIGN KEY

- FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table
- ```
  CREATE TABLE Orders (
      OrderID int NOT NULL,
      OrderNumber int NOT NULL,
      PersonID int,
      PRIMARY KEY (OrderID),
      FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
  );
  ```

24) FROM KEYWORD

- used to specify which table to select or delete data from
- **SELECT SPECIFIC COLUMNS FROM TABLE**
- ```
  SELECT CustomerName, City FROM Customers;
  ```
- **SELECT WHOLE TABLE**
- ```
  SELECT * FROM Customers;
  ```
- **DELETE DETAILS OF PARTICULAR USER**
  ```

- ```
  DELETE FROM Customers
  WHERE CustomerName='Alfreds Futterkiste';
  ```

25)FULL OUTER JOIN

- returns all rows when there is a match in either left table or right table
- ```
  SELECT Customers.CustomerName, Orders.OrderID
  FROM Customers
  FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
  ORDER BY Customers.CustomerName;
  ```

26)GROUPBY KEYWORD

- group the result set
- used with aggregate functions: COUNT, MAX, MIN, SUM, AVG
- ```
  SELECT COUNT(CustomerID), Country
  FROM Customers
  GROUP BY Country
  ORDER BY COUNT(CustomerID) DESC;
  ```

27)HAVING KEYWORD

- `HAVING` command is used instead of WHERE with aggregate functions
- ```
  SELECT COUNT(CustomerID), Country
  FROM Customers
  GROUP BY Country
  HAVING COUNT(CustomerID) > 5
  ORDER BY COUNT(CustomerID) DESC;
  ```

28)IN KEYWORD

- specify multiple values in a WHERE clause
- shorthand for multiple OR conditions
- ```
  SELECT * FROM Customers
  WHERE Country NOT IN ('Germany', 'France', 'UK');
  ```
- SQL selects all customers that are from the same countries as the suppliers
- ```
  SELECT * FROM Customers
  WHERE Country IN (SELECT Country FROM Suppliers);
  ```

29)INDEX KEYWORD

- **create indexes in tables**
- If you want to create an index on a combination of columns, you can list the column names within the parentheses, separated by commas
- ```
  CREATE INDEX idx_pname
  ON Persons (LastName, FirstName);
  ```
- Updating a table with indexes takes more time than updating a table without
- **delete an index in a table**
- ```
  ALTER TABLE table_name
  DROP INDEX index_name;
  ```

30)INNER JOIN KEYWORD

- `INNER JOIN` command returns rows that have matching values in both tables
- ```
  SELECT Orders.OrderID, Customers.CustomerName
  FROM Orders
  INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
  ```
- selects all orders with customer and shipper information
- ```
  SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName
  FROM ((Orders
  INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)
  INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);
  ```

31)INSERT INTO KEYWORD

- insert new rows in a table
- ```
  INSERT INTO Customers (CustomerName, ContactName, Address, City,
  PostalCode, Country)
  VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen
  21', 'Stavanger', '4006', 'Norway');
  ```

32)INSERT INTO SELECT KEYWORD

- copies data from one table and inserts it into another table
- ```
  INSERT INTO Customers (CustomerName, City, Country)
  SELECT SupplierName, City, Country FROM Suppliers
  WHERE Country='Germany';
  ```

33)IS NULL KEYWORD

- test for empty values

- SQL lists all customers with a NULL value in the "Address" field
- ```
  SELECT CustomerName, ContactName, Address
  FROM Customers
  WHERE Address IS NULL;
  ```

34) IS NOT NULL KEYWORD:

- test for non-empty values
- lists all customers with a value in the "Address" field
- ```
  SELECT CustomerName, ContactName, Address
  FROM Customers
  WHERE Address IS NOT NULL;
  ```

35)LEFT JOIN KEYWORD

- returns all rows from the left table, and the matching rows from the right table
- ```
  SELECT Customers.CustomerName, Orders.OrderID
  FROM Customers
  LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
  ORDER BY Customers.CustomerName;
  ```

36)RIGHT JOIN KEYWORD

- returns all rows from the right table and the matching records from the left table.
- ```
  SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
  FROM Orders
  RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
  ORDER BY Orders.OrderID;
  ```

37)LIKE KEYWORD

- used in a WHERE clause to search for a specified pattern in a column
  - two wildcards with `LIKE`
  - % - Represents zero, one, or multiple characters
  - _ - Represents a single character
- ```
  SELECT * FROM Customers
  WHERE CustomerName LIKE 'a%';//This gives all the records that
  starts with a
  ```
- ```
  SELECT * FROM Customers
  WHERE CustomerName LIKE '%a'; //This gives all the records that
  ending with a
  ```

- CustomerName that have "or" in any position
- ```
  SELECT * FROM Customers
  WHERE CustomerName LIKE '%or%';
  ```
- starts with "a" and are at least 3 characters in length
- ```
  SELECT * FROM Customers
  WHERE CustomerName LIKE 'a_%_%';
  ```

38)SELECT TOP,LIMIT,ROWNUM

- `SELECT TOP` command is used to specify the number of records to return
- MySQL uses `LIMIT`
- selects the first three records
- ```
  SELECT * FROM Customers
  LIMIT 3;
  ```

39)SELECT NOT KEYWORD

- `NOT` command is used with WHERE to only include rows where a condition is not true
- ```
  SELECT * FROM Customers
  WHERE NOT Country='Germany';
  ```

40)SELECT NOT NULL KEYWORD

- The `NOT NULL` constraint enforces a column to not accept NULL values
- ```
  CREATE TABLE Persons (
      ID int NOT NULL,
      LastName varchar(255) NOT NULL,
      FirstName varchar(255) NOT NULL,
      Age int
  );
  ```
- ```
  ALTER TABLE Persons
  MODIFY Age int NOT NULL;
  ```

41)OR KEYWORD

- `OR` command is used with WHERE to include rows where either condition is true
- ```
  SELECT * FROM Customers
  WHERE City='Berlin' OR City='München';
  ```

42)ORDERBY KEYWORD

- `ORDER BY` command is used to sort the result set in ascending or descending order
- ```
  SELECT * FROM Customers
  ORDER BY CustomerName ASC;
  ```
- ```
  SELECT * FROM Customers
  ORDER BY CustomerName DESC;
  ```
  ```

43)PRIMARY KEY KEYWORD

- PRIMARY KEY constraint uniquely identifies each record in a table
- table can have only one primary key
- ```
  CREATE TABLE Persons (
      ID int NOT NULL,
      LastName varchar(255) NOT NULL,
      FirstName varchar(255),
      Age int,
      PRIMARY KEY (ID)
  );
  ```
- ```
  CREATE TABLE Persons (
      ID int NOT NULL,
      LastName varchar(255) NOT NULL,
      FirstName varchar(255),
      Age int,
      CONSTRAINT PK_Person PRIMARY KEY (ID,LastName)
  );
  ```
- example above there is only ONE PRIMARY KEY (PK_Person). However, the VALUE of the primary key is made up of TWO COLUMNS (ID + LastName)
- ```
  ALTER TABLE Persons
  ADD PRIMARY KEY (ID);
  ```
- ```
  ALTER TABLE Persons
  ADD CONSTRAINT PK_Person PRIMARY KEY (ID,LastName);
  ```
- **DROP A PRIMARY KEY**
- ```
  ALTER TABLE Persons
  DROP PRIMARY KEY;
  ```

44)SELECT KEYWORD

- select data from a database
- SELECT SPECIFIC COLUMNS OF TABLE
- ```
  SELECT CustomerName, City FROM Customers;
  ```
- SELECT ALL COLUMNS OF TABLE
- ```
  SELECT * FROM Customers;
  ```

45)SELECT DISTINCT KEYWORD

- returns only distinct (different) values in the result set
- ```
  SELECT DISTINCT Country FROM Customers;
  ```

46)SELECT INTO KEYWORD

- SELECT INTO command copies data from one table and inserts it into a new table
- **TO COPY THE TABLE INTO OTHER TABLE OF SAME DATABASE**
- ```
  SELECT * INTO CustomersBackup2017
  FROM Customers;
  ```
- **TO COPY THE TABLE INTO OTHER TABLE OF DIFFERENT DATABASE**
  ```

- ```
  SELECT * INTO CustomersBackup2017 IN 'Backup.mdb'
  FROM Customers;
  ```
- ```
  SELECT Customers.CustomerName, Orders.OrderID
  INTO CustomersOrderBackup2017
  FROM Customers
  LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
  ```

47)SQL SET KEYWORD

- SET command is used with UPDATE to specify which columns and values that should be updated in a table
- ```
  UPDATE Customers
  SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
  WHERE CustomerID = 1;
  ```

48)SQL TABLE KEYWORD

- **CREATE TABLE**
- ```
  CREATE TABLE Persons (
      PersonID int,
      LastName varchar(255),
      FirstName varchar(255),
      Address varchar(255),
      City varchar(255)
  );
  ```
- **COPY OF TABLE**
- ```
  CREATE TABLE TestTable AS
  SELECT customername, contactname
  FROM customers;
  ```
- **ALTER TABLE**
- adds, deletes, or modifies columns in a table
- ```
  ALTER TABLE Customers
  ADD Email varchar(255);
  ```
- **DROP THE COLUMN**
- ```
  ALTER TABLE Customers
  DROP COLUMN Email;
  ```
- **DROP THE TABLE**
- ```
  DROP TABLE Shippers;//deletes the table including its structure
  ```
- **TRUNCATE TABLE**
- ```
  TRUNCATE TABLE Categories;//it just delete the content not the structure
  ```

49)SQL UNION AND UNION ALL KEYWORDS

- combines the result set of two or more SELECT statements (only distinct values)
- ```
  SELECT City FROM Customers
  UNION
  SELECT City FROM Suppliers
  ORDER BY City;
  ```

- combines the result set of two or more SELECT statements (allows duplicate values)
- ```
  SELECT City FROM Customers
  UNION ALL
  SELECT City FROM Suppliers
  ORDER BY City;
  ```

50) SQL UNIQUE KEYWORD

- ```
  CREATE TABLE Persons (
      ID int NOT NULL,
      LastName varchar(255) NOT NULL,
      FirstName varchar(255),
      Age int,
      UNIQUE (ID)
  );
  ```
- ```
  CREATE TABLE Persons (
      ID int NOT NULL,
      LastName varchar(255) NOT NULL,
      FirstName varchar(255),
      Age int,
      CONSTRAINT UC_Person UNIQUE (ID,LastName)
  );
  ```
- ```
  ALTER TABLE Persons
  ADD UNIQUE (ID);
  ```
- ```
  ALTER TABLE Persons
  ADD CONSTRAINT UC_Person UNIQUE (ID,LastName);
  ```
- ```
  ALTER TABLE Persons
  DROP INDEX UC_Person;
  ```

MYSQL Functions

- **STRING FUNCTIONS**
  - ASCII
    - ```
      SELECT CustomerName,ASCII(CustomerName) AS
      NumCodeOfFirstChar
      FROM Customers;
      ```
  - CHAR_LENGTH()
    - ```
      SELECT CHAR_LENGTH(CustomerName) AS LengthOfName
      FROM Customers;
      ```
  - CHARACTER_LENGTH()
    - ```
      SELECT CHARACTER_LENGTH(CustomerName) AS LengthOfName
      FROM Customers;
      ```
  - CONCAT()
      ```

```
        SELECT CONCAT(Address, " ", PostalCode, " ",
        City) AS Address
        FROM Customers;
```

- CONCAT_WS()
  - ```
    SELECT CONCAT_WS(" ", Address, PostalCode,
    City) AS Address
    FROM Customers;
    ```
  - Here we have separator first
- FIELD()
  - If the specified value is not found in the list of values, this function will return 0.
  - ```
    SELECT FIELD(5, 0, 1, 2, 3, 4, 5);
    ```
  - Indexing starts from 1
- FIND IN SET()
  - If *string* is not found in *string_list*, this function returns 0
  - If *string* or *string_list* is NULL, this function returns NULL
  - If *string_list* is an empty string (""), this function returns 0
  - ```
    SELECT FIND_IN_SET("a", "s,q,l");
    ```
- FORMAT()
  - ```
    SELECT FORMAT(250500.5634, 2);
    ```
  - If the value is 0 then none of the decimal will be there
- INSERT()
  - ```
    SELECT INSERT("W3Schools.com", 1, 9, "Example");
    ```
  - Replaces first 9 charecters with the second string
  - If the initial index and number of charecters to change is greater than original length of string then whole string would be modified
- INSTR()
  - ```
    SELECT INSTR("W3Schools.com", "3") AS MatchPosition;
    ```
  - returns the position of the first occurrence of a string in another string
- LCASE() or LOWER() OR UCASE() OR UPPER()
  - ```
    SELECT LCASE("SQL Tutorial is FUN!");
    ```
  - ```
    SELECT LOWER(CustomerName) AS LowercaseCustomerName
    FROM Customers;
    ```
  - Converts into lower case
- LEFT()
  - ```
    SELECT LEFT(CustomerName, 5) AS ExtractString
    FROM Customers;
    ```
  - Extract string from left
- LENGTH()
  - ```
    SELECT LENGTH(CustomerName) AS LengthOfName
    FROM Customers;
    ```
  - Returns the length of that customer name
- LOCATE()
  - ```
    SELECT LOCATE("3", "W3Schools.com") AS MatchPosition;
    ```
  - Search first string in second string
    ```

        - SELECT LOCATE("com", "W3Schools.com", 3) AS MatchPosition;
    - LPAD()
        - SELECT LPAD("SQL Tutorial", 20, "ABC");
        - left-pads a string with another string, to a certain length
    - LTRIM()
        - SELECT LTRIM("   SQL Tutorial") AS LeftTrimmedString;
    - SUBSTR() or MID() OR SUBSTRING()
        - SELECT SUBSTR("SQL Tutorial", 5, 3) AS ExtractString;
        - Index can be of positive or negative number
        - Parameters are (string,start,length)
        - SELECT SUBSTR("SQL Tutorial", -5, 5) AS ExtractString;
        - SELECT SUBSTRING_INDEX("www.w3schools.com", ".", 2);
    - POSITION() FUNCTION
        - SELECT POSITION("a" IN CustomerName) FROM Customers
    - REPEAT() FUNCTION
        - SELECT REPEAT("SQL Tutorial", 3);
        - SELECT REPEAT(CustomerName, 2) FROM Customers;
    - REPLACE() FUNCTION
        - case-sensitive replacement
        - SELECT REPLACE("XYZ FGH XYZ", "X", "m");
    - REVERSE() FUNCTION
        - SELECT REVERSE(CustomerName) FROM Customers;
    - RIGHT() FUNCTION
        - SELECT RIGHT("SQL Tutorial is cool", 4) AS ExtractString;
    - RPAD() FUNCTION
        - SELECT RPAD(CustomerName, 30, "ABC") AS RightPadCustomerName FROM Customers;
    - RTRIM() FUNCTION
        - SELECT RTRIM("SQL Tutorial   ") AS RightTrimmedString;
        - Remove trailing spaces from a string
    - SPACE() FUNCTION
        - SELECT SPACE(10);
        - Return a string with 10 space charecters
    - STRCMP() FUNCTION
        - If *string1* = *string2*, this function returns 0
        - If *string1* < *string2*, this function returns -1
        - If *string1* > *string2*, this function returns 1
        - SELECT STRCMP("SQL Tutorial", "HTML Tutorial");

MySQL Numeric Functions

Function	Description
ABS	Returns the absolute value of a number
ACOS	Returns the arc cosine of a number
ASIN	Returns the arc sine of a number
ATAN	Returns the arc tangent of one or two numbers
ATAN2	Returns the arc tangent of two numbers
AVG	Returns the average value of an expression
CEIL	Returns the smallest integer value that is >= to a number
CEILING	Returns the smallest integer value that is >= to a number
COS	Returns the cosine of a number
COT	Returns the cotangent of a number

COUNT	Returns the number of records returned by a select query
DEGREES	Converts a value in radians to degrees
DIV	Used for integer division
EXP	Returns e raised to the power of a specified number
FLOOR	Returns the largest integer value that is <= to a number
GREATEST	Returns the greatest value of the list of arguments
LEAST	Returns the smallest value of the list of arguments
LN	Returns the natural logarithm of a number
LOG	Returns the natural logarithm of a number, or the logarithm of a number to a specified base
LOG10	Returns the natural logarithm of a number to base 10
LOG2	Returns the natural logarithm of a number to base 2

MAX	Returns the maximum value in a set of values
MIN	Returns the minimum value in a set of values
MOD	Returns the remainder of a number divided by another number
PI	Returns the value of PI
POW	Returns the value of a number raised to the power of another num
POWER	Returns the value of a number raised to the power of another num
RADIANS	Converts a degree value into radians
RAND	Returns a random number
ROUND	Rounds a number to a specified number of decimal places
SIGN	Returns the sign of a number
SIN	Returns the sine of a number

SQRT	Returns the square root of a number
SUM	Calculates the sum of a set of values
TAN	Returns the tangent of a number
TRUNCATE	Truncates a number to the specified number of decimal places

MySQL Date Functions

Function	Description
ADDDATE	Adds a time/date interval to a date and then returns the date
ADDTIME	Adds a time interval to a time/datetime and then returns the time/datetime
CURDATE	Returns the current date
CURRENT_DATE	Returns the current date
CURRENT_TIME	Returns the current time

CURRENT_TIMESTAMP	Returns the current date and time
CURTIME	Returns the current time
DATE	Extracts the date part from a datetime expression
DATEDIFF	Returns the number of days between two date values
DATE_ADD	Adds a time/date interval to a date and then returns the date
DATE_FORMAT	Formats a date
DATE_SUB	Subtracts a time/date interval from a date and then returns the d
DAY	Returns the day of the month for a given date
DAYNAME	Returns the weekday name for a given date
DAYOFMONTH	Returns the day of the month for a given date
DAYOFWEEK	Returns the weekday index for a given date

DAYOFYEAR	Returns the day of the year for a given date
EXTRACT	Extracts a part from a given date
FROM_DAYS	Returns a date from a numeric datevalue
HOUR	Returns the hour part for a given date
LAST_DAY	Extracts the last day of the month for a given date
LOCALTIME	Returns the current date and time
LOCALTIMESTAMP	Returns the current date and time
MAKEDATE	Creates and returns a date based on a year and a number of days value
MAKETIME	Creates and returns a time based on an hour, minute, and second value
MICROSECOND	Returns the microsecond part of a time/datetime
MINUTE	Returns the minute part of a time/datetime

MONTH	Returns the month part for a given date
MONTHNAME	Returns the name of the month for a given date
NOW	Returns the current date and time
PERIOD_ADD	Adds a specified number of months to a period
PERIOD_DIFF	Returns the difference between two periods
QUARTER	Returns the quarter of the year for a given date value
SECOND	Returns the seconds part of a time/datetime
SEC_TO_TIME	Returns a time value based on the specified seconds
STR_TO_DATE	Returns a date based on a string and a format
SUBDATE	Subtracts a time/date interval from a date and then returns the d
SUBTIME	Subtracts a time interval from a datetime and then returns the time/datetime

SYSDATE	Returns the current date and time
TIME	Extracts the time part from a given time/datetime
TIME_FORMAT	Formats a time by a specified format
TIME_TO_SEC	Converts a time value into seconds
TIMEDIFF	Returns the difference between two time/datetime expressions
TIMESTAMP	Returns a datetime value based on a date or datetime value
TO_DAYS	Returns the number of days between a date and date "0000-00-0
WEEK	Returns the week number for a given date
WEEKDAY	Returns the weekday number for a given date
WEEKOFYEAR	Returns the week number for a given date
YEAR	Returns the year part for a given date

YEARWEEK	Returns the year and week number for a given date

MySQL Advanced Functions

Function	Description
BIN	Returns a binary representation of a number
BINARY	Converts a value to a binary string
CASE	Goes through conditions and return a value when the first condition met
CAST	Converts a value (of any type) into a specified datatype
COALESCE	Returns the first non-null value in a list
CONNECTION_ID	Returns the unique connection ID for the current connection
CONV	Converts a number from one numeric base system to another
CONVERT	Converts a value into the specified datatype or character set

CURRENT_USER	Returns the user name and host name for the MySQL account that server used to authenticate the current client
DATABASE	Returns the name of the current database
IF	Returns a value if a condition is TRUE, or another value if a conditi FALSE
IFNULL	Return a specified value if the expression is NULL, otherwise return expression
ISNULL	Returns 1 or 0 depending on whether an expression is NULL
LAST_INSERT_ID	Returns the AUTO_INCREMENT id of the last row that has been inserted or updated in a table
NULLIF	Compares two expressions and returns NULL if they are equal. Otherwise, the first expression is returned
SESSION_USER	Returns the current MySQL user name and host name
SYSTEM_USER	Returns the current MySQL user name and host name
USER	Returns the current MySQL user name and host name

VERSION	Returns the current version of the MySQL database

- **SQL OPERATORS** (https://www.w3schools.com/sql/sql_operators.asp)
- 1)ARTHEMATIC OPERATOR
- 2)BITWISE OPERATOR
- 3)LOGICAL OPERATOR
- 4)COMPOUND OPERATOR
- 5)COMPARISION OPERATOR
- **MY SQL DATATYPES(**https://www.w3schools.com/sql/sql_datatypes.asp)