SQL Tutorial

Important Functions on MYSQL: https://www.w3schools.com/sql/sql_ref_mysql.asp

- **SELECT** - extracts data from a database
- **UPDATE** - updates data in a database
- **DELETE** - deletes data from a database
- **INSERT INTO** - inserts new data into a database
- **CREATE DATABASE** - creates a new database
- **ALTER DATABASE** - modifies a database
- **CREATE TABLE** - creates a new table
- **ALTER TABLE** - modifies a table
- **DROP TABLE** - deletes a table
- **CREATE INDEX** - creates an index (search key)
- **DROP INDEX** - deletes an index

```
SELECT address, city FROM Customers;
```
Selects columns: address and city from customers

```
SELECT * FROM Customers;
```
- * is used only for retrieving a complete table

```
SELECT DISTINCT column1, column2, ...
FROM table_name;
```
Selects the distinct values from the table

```
SELECT COUNT(DISTINCT Country) FROM Customers;
```
Counts the values in a column

WHERE clause is used to filter records

```
SELECT customername FROM Customers
WHERE Country='Mexico';
```
→ Gave all the name of the customers from mexico

Operator	**Description**
=	Equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

The AND operator displays a record if all the conditions separated by AND is TRUE.

- Eg: Person A is from Country X and from City Y

The OR operator displays a record if any of the conditions separated by OR is TRUE.

- Eg: Person is from Country A or Country B

The NOT operator displays a record if the condition(s) is NOT TRUE.
- Eg: Person is NOT from Country A

```
SELECT (city) FROM Customers
WHERE NOT City='Berlin' AND NOT City='México D.F.';
```

Combining the 2 functions

When not function is involved expand everything imagining it is a negative function in math. A – (B+C) = (A -B) + (A-C)

```
SELECT * FROM Customers
WHERE Country='Germany' AND (City='Berlin' OR City='München') OR City='London';
```

```
SELECT * FROM Customers
ORDER BY Country ASC, CustomerName DESC;
```

Ordering the country in ascending order, but the customer name for the same country in descending order

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');
```

A new row will be produced for this new set of values.
If value isn't insert into a certain column 'null value' will appear

```
SELECT LastName, FirstName, Address FROM Persons
WHERE Address IS NULL;
```

Takes values from the selected columns, then compares the certain column values if null then reports back all the values with the null value too

```
SELECT ProductName
FROM Products
WHERE ProductID = ANY (SELECT ProductID FROM OrderDetails WHERE Quantity = 10);
```

ANY function is used to verify if the following statement is true for any of the values from the selected table. Output will be TRUE/FALSE.

```
SELECT ProductName
FROM Products
WHERE ProductID = ALL (SELECT ProductID FROM OrderDetails WHERE Quantity = 10);
```

ANY function is used to verify if the following statement is true for any of the values from the selected table. Output will be TRUE/FALSE.

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
WHERE CustomerID = 1;
```

Without a where statement, all values of the table will be updated

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

Remember the where statements.

OR ELSE

```
DELETE FROM table_name;
```

→ In deleting the whole table

```
SELECT * FROM Customers
WHERE Country='USA' OR City='London'
```

Selecting 5 rows from Customer table where country is USA or city is London

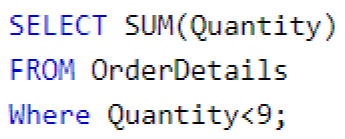
```
SELECT MIN(Price)
FROM Products
Where SupplierID<=8;
```

Selecting minimum price from products where supplier ID <=8

MAX is used for maximum price

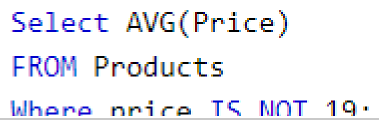
```
SELECT * FROM Products
Order By Price ASC
Limit 3;
```

If 3 minimum prices are required, min isn't used, because it will give only one value. Therefore we used ascending, then select the first three values.



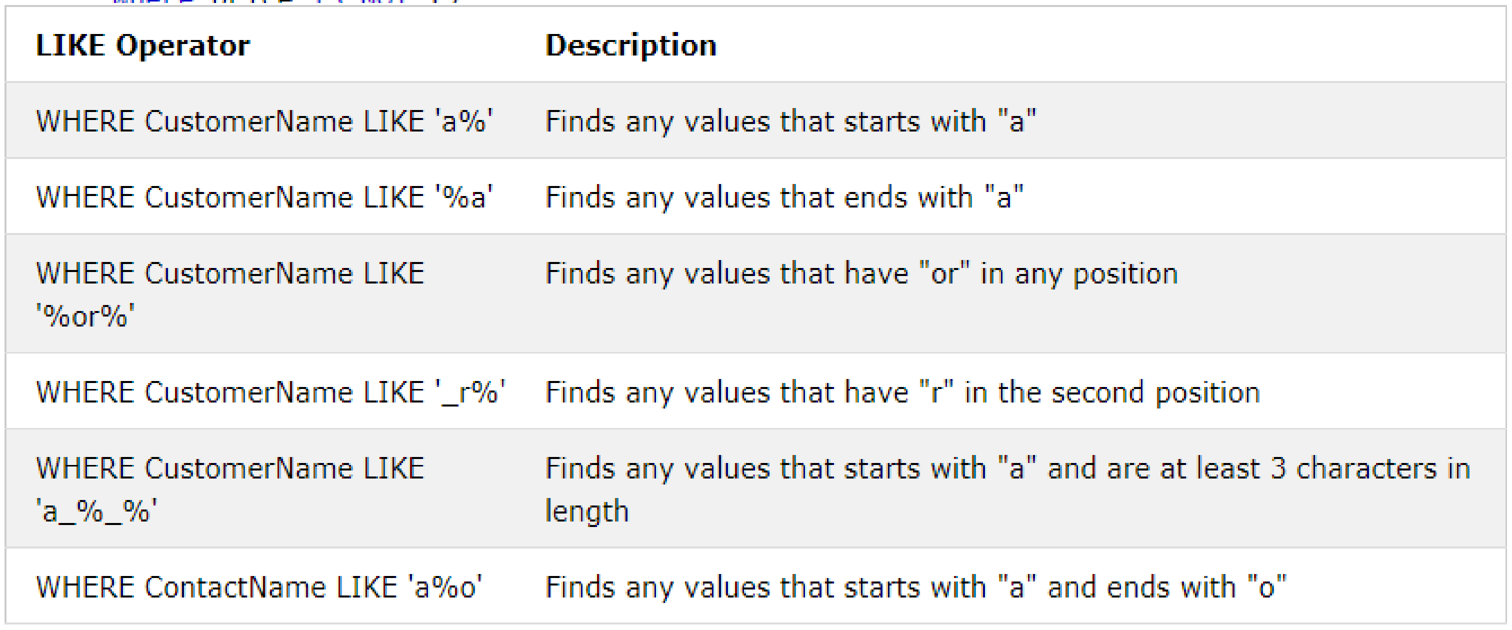
```
SELECT SUM(Quantity)
FROM OrderDetails
Where Quantity<9;
```

Total sum of quantity values, where quantity < 9



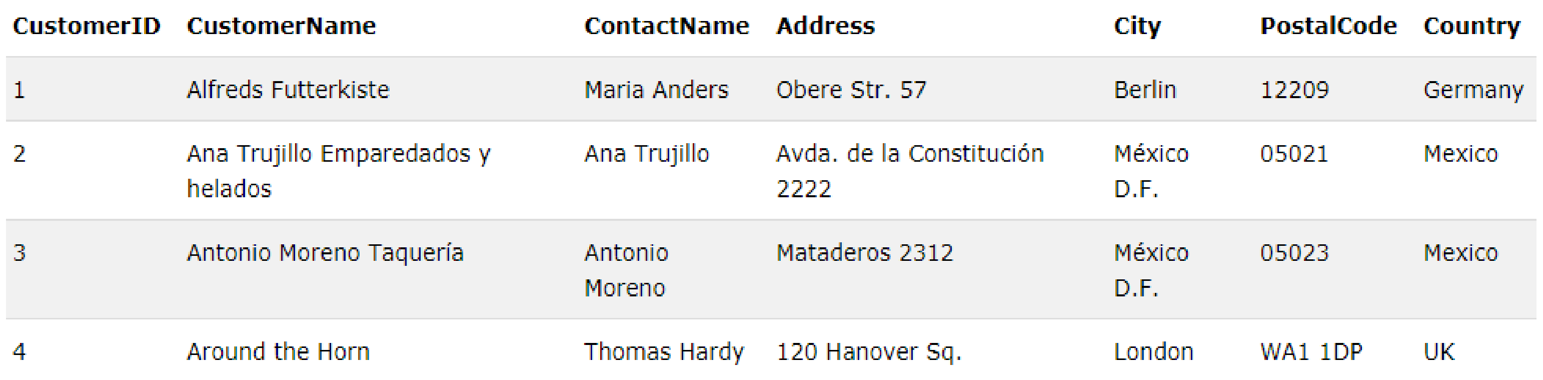
```
Select AVG(Price)
FROM Products
Where price IS NOT 19;
```

Average Price of products, where the price doesn't equal 19



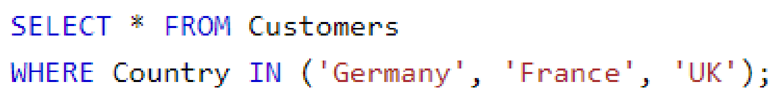
LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that starts with "a"
WHERE CustomerName LIKE '%a'	Finds any values that ends with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in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

Remember that the values will contain the whole set of words in a column not just an individual word in a group



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

If a%a is applied, "Ana Trujillo Emparedados y helados" customer name won't come. "Antonio Moreno Taquería" will appear since that is that starts with a and ends with a



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

Selecting customers from these countries, replacement for OR function with the same tab values

```
SELECT * FROM Customers
WHERE Country IN (SELECT Country FROM Suppliers);
```

Selecting the customer from the same country as the supplier

```
SELECT * FROM Products
WHERE (Price BETWEEN 10 AND 20)
AND NOT CategoryID IN (1,2,3);
```

- Not Selecting products between 10 and 20 and not with categoryIDs 1,2 and 3

Taking products between the 2 word's characters

```
SELECT * FROM Orders
WHERE OrderDate BETWEEN #07/04/1996# AND #07/09/1996#;
```

Month, Day, Year
Applying dates

```
SELECT CustomerName, CONCAT(Address,', ',PostalCode,', ',City,', ',Country) AS Address
FROM Customers;
```

Output is:

CustomerName	Address
Alfreds Futterkiste	Obere Str. 57, 12209 Berlin, Germany

Concat chains 2 or more functions together

```
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;
```

abc.xyz represents that xyz is inside the table abc
Such as the example above. C = customer, therefore customer.orderid represents orderid column in within the customer table

<u>Join</u>

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

Selecting the columns, from the order table and merging it with customer table by linking the customer ID in order and customer ID in customers as one.

Inner join: Combing the 2 tables with values matching in both

John	Adams	05/23/1784	$124.00
Thomas	Jefferson	03/14/1760	$78.50
Thomas	Jefferson	09/03/1790	$65.50

Left join: Even if values don't exist in Table B, it will be merged with null values replacing table B

Thomas	Jefferson	09/03/1790	$65.50
James	Madison	NULL	NULL
James	Monroe	NULL	NULL

Right join: Even if values don't exist in Table A, it will be merged with null values replacing table A

Thomas	Jefferson	09/03/1790	$65.50
NULL	NULL	07/21/1795	$25.50
NULL	NULL	11/27/1787	$14.40

Full join: Combing both tables irrespective if values exist or not in matching tables

Thomas	Jefferson	09/03/1790	$65.50
NULL	NULL	07/21/1795	$25.50
NULL	NULL	11/27/1787	$14.40
James	Madison	NULL	NULL
James	Monroe	NULL	NULL

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

2 columns from the same table will combine to form a single column

```
SELECT City FROM Customers
UNION
SELECT City FROM Suppliers
ORDER BY City;
```

Union combines the 2 individual columns of distinct values from 2 different tables into 1 output column.

- Must have the same data type
- Selected columns in each table must be in the same order
- Both table must have same number of columns

Union All – Same as Union except copies repeated values if present in both tables

```
SELECT 'Customer' As Type, ContactName, City, Country
FROM Customers
UNION
SELECT 'Pokemon', ContactName, City, Country
FROM Suppliers
```

The red words are inserted as the value under type, as it has been programmed above
The rest are used as referred to in the table provided.

```
SELECT Count(customerID), Country
From Customers
Group by City
```

This statement groups the same cities and counts the number. Then 2 columns are produced: containing the number of unique IDs and the relevant country of the city the selected count is of

```
SELECT Shippers.ShipperName, COUNT(Orders.OrderID) AS NumberOfOrders FROM Orders
LEFT JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID
GROUP BY ShipperName;
```

2 columns are created: shipper name and number of orders (based on orderid)
Left join occurs between shippers and orders
Then grouped by shippername

Aggregate functions are used to compute against a "returned column of numeric data" from your SELECT statement. Functions such as sum, min, avg, etc..

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5
ORDER BY COUNT(CustomerID) DESC;
```

2 columns are created: count from customer ID and Country list
Grouped by country
Having function > 5
Ordered in descending order of count customerID

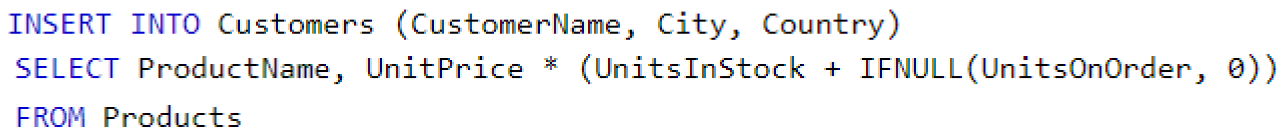
Having function replaces where function when group count (aggregate functions are used)

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM Orders
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
WHERE LastName = 'Davolio' OR LastName = 'Fuller'
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 25;
```

2 columns created, one of the employees' last name and count of the OrderID
Inner Join the employees and orderid based on the employeeID
With last name davolio or fuller
Group by last name
Have count of orderid > 25

```
SELECT ProductName
FROM Products
WHERE ProductID = ANY (SELECT ProductID FROM OrderDetails WHERE Quantity = 10);
```

https://www.w3schools.com/sql/sql_select_into.asp – How to copy data into another data sheet or database etc...



```
INSERT INTO Customers (CustomerName, City, Country)
 SELECT ProductName, UnitPrice * (UnitsInStock + IFNULL(UnitsOnOrder, 0))
 FROM Products
```

Inserting data from suppliers database into customer database with the relevant column.
IfNull function allows a value if null to be replaced by another value.
In this case, if unitsonorder is null, the value will be replaced with 0. This is because, a null value in the equation will output a null value no matter what. 0 has a fixed value and can be used in the function.
Any text between “/* and */” **AND** “--” will be ignored (as both are comments)

SQL Database

```
CREATE DATABASE testDB;
```
- Creates a database called testDB

```
DROP DATABASE testDB;
```
- Deletes a database called testDB

```
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

Creating a table called persons with a column that can accept the followed character type and 255 characters

VARCHAR (x) – varchar can contain any type of character with x character limit
Char (x) – can contain any type of character with only x character length
Declare statement is reference to a certain statement, and is used in the body of the program

```
DROP TABLE Shippers;
```
- Deletes the table called shippers

```
TRUNCATE TABLE table_name;
```
- Deletes the data inside the table, but not the table itself

```
ALTER TABLE table_name
ADD column_name datatype;
```
To alter a table, by add/drop column

```
ALTER TABLE table_name
MODIFY COLUMN column_name datatype;
```
Modifies data-type of column for a certain table

```
CREATE TABLE table_name (
    column1 datatype constraint,
    column2 datatype constraint,
    column3 datatype constraint,
    ....
);
```

Type of constraints

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 - Uniquely identifies a row/record in another table
CHECK - Ensures that all values in a column satisfies a specific condition
DEFAULT - Sets a default value for a column when no value is specified
INDEX - Used to create and retrieve data from the database very quickly

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

- Programming the certain columns to have constraints

```
ALTER TABLE Persons
DROP INDEX UC_Person;
```

- Delete for 1 column

```
ALTER TABLE Persons
ADD CONSTRAINT UC_Person UNIQUE (ID,LastName);
```

Adding unique constraint for ID and Last Name

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

Setting the primary key to 1 column (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)
);
```

Keep 2 columns (ID and LastName) as the primary keys

```
ALTER TABLE Persons
ADD PRIMARY KEY (ID);
```

Add/Drop a primary key to a different table

```
ALTER TABLE Persons
ADD CONSTRAINT PK_Person PRIMARY KEY (ID,LastName);
```

Add/Drop multiple primary keys to a different table

A FOREIGN KEY is a field (or collection of fields) that links 2 tables, where 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)
);
```

Order ID – Not Null
Order number – Not Null
Person ID – integer
Primary Key – OrderID
PersonID is the foreign key which is present in persons table

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

Alternate table adding a constraint on the foreign key

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

Deleting a foreign key from an alternate table

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

This constraint checks that the age is >=18 and the city is sandness

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

Add/drop check the condition in an alternate table

```
CREATE TABLE Orders (
    ID int NOT NULL,
    OrderNumber int NOT NULL,
    OrderDate date DEFAULT GETDATE()
);
```

Date gets set by default when the table is created

Defaults – If no other value is given, a certain value provided will be the default

```
ALTER TABLE Persons
ALTER City DROP DEFAULT;
```

In the alternate table, the city column to drop by default

```
CREATE UNIQUE INDEX index_name
ON table_name (column1, column2, ...);
```

Create a unique index for certain table
If a unique index, isn't required remove the word unique and use the same function

```
CREATE TABLE Persons (
    ID int NOT NULL AUTO_INCREMENT,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
);
```

Auto_Increment is a feature which auto increases each value starting from 1 by each record

Primary Key is an auto_increment value

```
ALTER TABLE Persons AUTO_INCREMENT=100;
```

Use this function is the value is required to start at a certain number

DATE - format YYYY-MM-DD
DATETIME - format: YYYY-MM-DD HH:MI:SS
TIMESTAMP - format: YYYY-MM-DD HH:MI:SS
YEAR - format YYYY or YY

The types of date formats and the relevant code

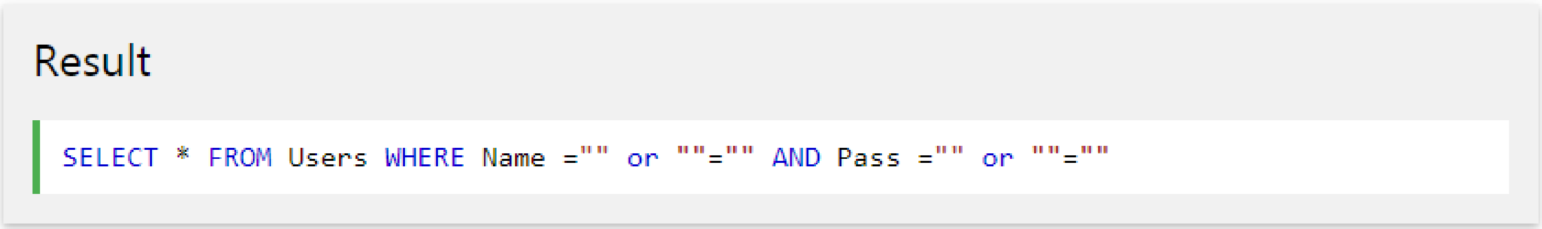
```
CREATE VIEW [Products Above Average Price] AS
SELECT ProductName, UnitPrice
FROM Products
WHERE UnitPrice > (SELECT AVG(UnitPrice) FROM Products);
```

Create view is a custom view of a sample part of the database for testing

```
CREATE OR REPLACE VIEW [Current Product List] AS
SELECT ProductID, ProductName, Category
FROM Products
WHERE Discontinued = No;
```

- This function allows the view to be created or updated

SQL Injection



Result

```
SELECT * FROM Users WHERE Name ="" or ""="" AND Pass ="" or ""=""
```

The SQL above is valid and will return all rows from the "Users" table, since **OR ""=""** is always TRUE.

This is a type of code that can be entered by a user to add, modify, or delete codes without permission. People create codes to block users from hacking their database, such as using the functions below

```
txtNam = getRequestString("CustomerName");
txtAdd = getRequestString("Address");
txtCit = getRequestString("City");
txtSQL = "INSERT INTO Customers (CustomerName,Address,City) Values(@0,@1,@2)";
db.Execute(txtSQL,txtNam,txtAdd,txtCit);
```

This restricts the values from being used as a function. For example: @value can't be replaced with a function since it has @ and is followed by others.
getRequestString("blabla") is also another prevention

```
SELECT INSTR("W3Schools.com", "3") AS MatchPosition;
```

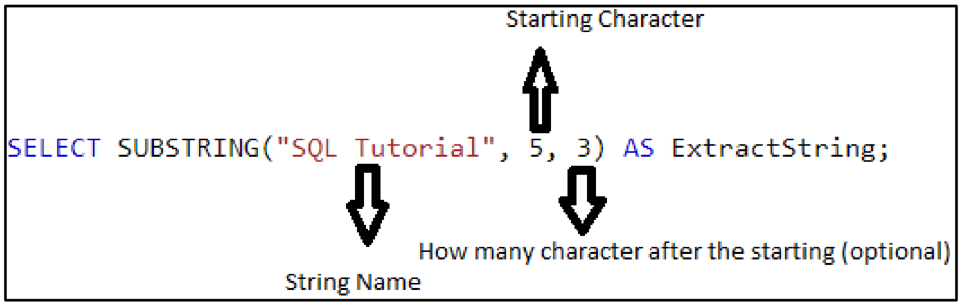
INSTR Searches 3 in W3schools.com and give the position of the first 3 it finds

```
SELECT CHARACTER_LENGTH("SQL Tutorial") AS LengthOfString;
```

Character_Length
Mentions character length. Char_Length is also another substitute

```
SELECT CONCAT_WS("-", "SQL", "Tutorial", "is", "fun!") AS ConcatenatedString;
```

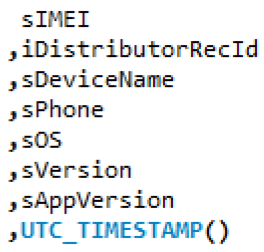
Concat_WS will give a common character(s) between the strings



```
SELECT SUBSTRING("SQL Tutorial", 5, 3) AS ExtractString;
```

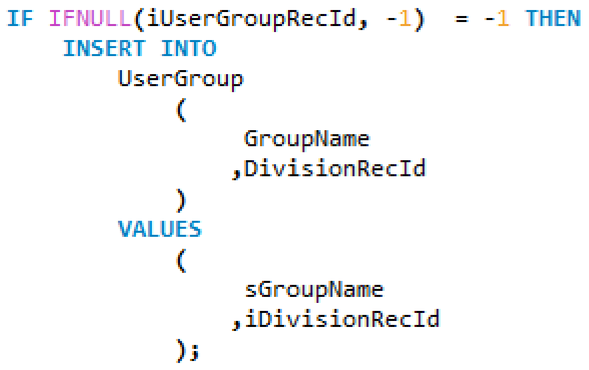
Normalization

Software Learnt



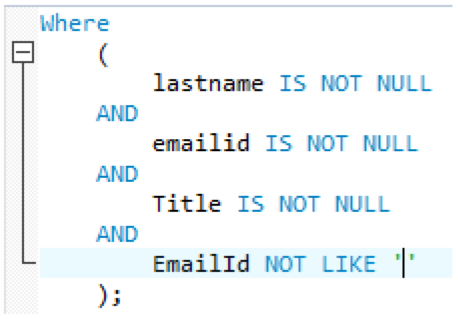
```
 sIMEI
,iDistributorRecId
,sDeviceName
,sPhone
,sOS
,sVersion
,sAppVersion
,UTC_TIMESTAMP()
```

The small letters are used to indicate (reference) what type of characters are allowed, such as s – string, i-integer, b-Boolean



```
IF IFNULL(iUserGroupRecId, -1)  = -1 THEN
    INSERT INTO
        UserGroup
            (
                 GroupName
                ,DivisionRecId
            )
        VALUES
            (
                 sGroupName
                ,iDivisionRecId
            );
```

When creating a new record, value will be null. Therefore replaced with -1 in ifnull function. Then creates a new record for it based on another code.



```
Where
    (
        lastname IS NOT NULL
    AND
        emailid IS NOT NULL
    AND
        Title IS NOT NULL
    AND
        EmailId NOT LIKE ''
    );
```

This code makes sure: lastname, email, and title is not null. Also checks that the emailid is not a blank one.

Didn't Understand

https://www.w3schools.com/sql/sql_any_all.asp