SQL

Created by	Chandrashekhar Robbi

This is the tutorial on SQL

SQL Constraint

- 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.
- Constraint 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. A `FOREIGN KEY` is a field (or collection of fields) in one table, that refers to the `PRIMARY KEY` in another table.
  - `CHECK` : Ensures that the values in a column satisfies a specific condition.
  - `DEFAULT` : Seta 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 Date Data Types

MySQL comes with the following data types for storing a date or a date/time value in the database:

- `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

Create Table

```
CREATE TABLE customer (
ID INT PRIMARY KEY AUTO_INCREMENT,
Name VARCHAR(255),
Address VARCHAR(255),
City VARCHAR(255),
PostalCode INT,
Country VARCHAR(255));
```

Insert Rows into Table

Syntax

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

By Single Line

```
INSERT INTO customer VALUES
(1, 'Alfred Futterkiste', "Obere Str. 57", "Berlin", 12209, 'Country');
```

By Multiple Lines

```
INSERT INTO customer VALUES
(2, 'Ana Trujillo Emparedados y helados', "Avda. de la Constitución 2222", "México D.
F.", 05021, 'Mexico'),
(3,"Antonio Moreno Taquería","Mataderos","México D.F.",05023  ,"Mexico"),
(4,"Around the Horn","120 Hanover Sq.","London","WA1 1DP","UK");
```

In the above the `Wa1 1DP` is String but `postal code` column was INT so it is inserted 0 to Data base

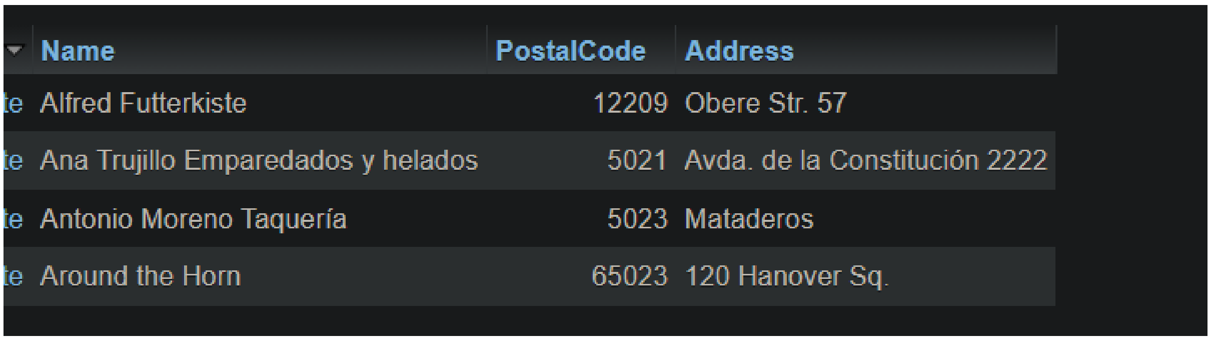
SELECT

All Rows at once

```
SELECT * FROM CUSTOMERS;
```

Selective Rows

```
SELECT Name, PostalCode, Address FROM customer;
```



	Name	PostalCode	Address
te	Alfred Futterkiste	12209	Obere Str. 57
te	Ana Trujillo Emparedados y helados	5021	Avda. de la Constitución 2222
te	Antonio Moreno Taquería	5023	Mataderos
te	Around the Horn	65023	120 Hanover Sq.

DISTINCT

It selects the DISTINCT value from each specified column

```
SELECT DISTINCT * FROM customer;
```

WHERE

```
SELECT * FROM customer WHERE ID = 4
```

AND , OR , NOT

AND

```
SELECT * from customer WHERE id = 1 and name = "Alfred Futterkiste";
```

1 ROW IS DISPLAYED

OR

```
SELECT * from customer WHERE id = 1 and name = "Alfred" OR id = 2;
```

NOT

```
SELECT * from customer WHERE NOT id = 2;
```

Order By

```
SELECT * from customer ORDER BY Address;
```

ASC

```
SELECT * from customer ORDER BY Address ASC;
```

DESC

```
SELECT * from customer ORDER BY Address DESC;
```

2 or more columns

```
SELECT * from customer ORDER BY Name,Address;
```

NULL/ NOT NULL

```
SELECT * from customer Where PostalCode is not NUll;
```

```
SELECT * from customer Where PostalCode is NUll;
```

UPDATE

```
UPDATE customer SET Name = "Robbi" , PostalCode = 20232 WHERE ID = 2;
```

DELETE

Column

```
DELETE FROM customer WHERE ID = 4;
```

Entire Records

```
DELETE FROM customer;
```

LIMIT

```
SELECT * FROM Customer WHERE ID < 4 LIMIT 2;
```

It can also work without WHERE :)

MIN / MAX

MIN

```
SELECT MIN(PostalCode) FROM customer;
```

```
SELECT MIN(PostalCode) AS MIN_POS FROM customer;
```

MAX

```
SELECT MAX(PostalCode) FROM customer;
```

```
SELECT MAX(PostalCode) AS MAX_AMT FROM Customer;
```

COUNT(), AVG() and SUM()

Count()

```
SELECT COUNT(Name) FROM Customer;
```

```
SELECT COUNT(Name) as COUNT_NUMBER FROM Customer;
```

SUM()

```
SELECT SUM(PostalCode) FROM customer;
```

AVG()

```
SELECT AVG(PostalCode) FROM customer;
```

❗ Note :
The AVG(), SUM() only works if the colum has INT value else returns 0

LIKE

Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"

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

```
SELECT * FROM customer WHERE Name LIKE 'c%r';
```

In

```
SELECT * FROM customer WHERE NAME IN ('Chandrashekhar','Robbi');
```

BETWEEN

```
SELECT * FROM customer WHERE PostalCode BETWEEN 200 AND 200000;
```

BETWEEN with IN

```
SELECT * FROM customer
WHERE PostalCode BETWEEN 200 AND 200000
OR Name IN ('Chandrashekhar','Robbi')
```

ALIAS

```
SELECT Name AS Customer_name FROM customer;
```

JOINS

Inner Join

https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcR8d-gaXVHQcQQOSkrDpN7WIj2y9aK-geZu7YCoiOmEDzjZXrdER6jb4G7kMs4FjVwVpxQ&usqp=CAU

```
SELECT customer.ID, Orders.OrderId , Customer.Name FROM orders INNER JOIN customer ON
 customer.ID=orders.CustomerId;
```

Left Join/ Outer Join

https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcTsZy1mI4JTCwwmT6XEzLyDvHaN5IV9ZK_UbVrljN9KRyKtR2c_Or7iZLVT2_ixlPckYo0&usqp=CAU

```
SELECT Orders.OrderId, customer.ID,customer.Name, Orders.OrderDate, Orders.ShipperId,
 Customer.City, customer.PostalCode FROM orders LEFT JOIN customer ON customer.ID = or
ders.CustomerId;
```

Full Join

```
SELECT Orders.OrderId, customer.ID,customer.Name, Orders.OrderDate, Orders.ShipperId,
 Customer.City, customer.PostalCode FROM orders LEFT JOIN customer ON customer.ID = or
ders.CustomerId
UNION
SELECT Orders.OrderId, customer.ID,customer.Name, Orders.OrderDate, Orders.ShipperId,
 Customer.City, customer.PostalCode FROM orders RIGHT JOIN customer ON customer.ID=ord
ers.CustomerId;
```

Union & UnionAll

Union

```
SELECT customer.Name FROM customer
UNION
SELECT orders.OrderDate FROM orders;
```

UnionAll

```
SELECT customer.Name FROM customer
UNION ALL
SELECT orders.OrderDate FROM orders;
```

Union with Where

```
SELECT customer.Name from customer WHERE customer.Name = "Chandrashekhar"
UNION ALL
SELECT suppliers.Name FROM suppliers WHERE suppliers.Name = "John";
```

GroupBy

```
SELECT COUNT(ID) , Country FROM customer GROUP BY Country;
```

Having

The `HAVING` clause was added to SQL because the `WHERE` keyword cannot be used with aggregate functions.

```
SELECT Name,Country,COUNT(Country) from customer GROUP BY ID HAVING COUNT(ID) = 1;
```

EXISTS

The `Exists` operator is used to test for the existence of any record in a subquery

The `EXISTS` operator returns `True` if the subquery returns one or more records

```
Select ID,Name from customer WHERE EXISTS (SELECT Name FROM customer WHERE ID = 1);
```

Any

```
SELECT Name, ID from customer
where ID = ANY
(SELECT ID FROM customer WHERE ID > 2);
```

ALL

```
SELECT Name, ID from customer where ID = ALL (SELECT ID FROM customer WHERE ID > 2);
```

Select INTO

```
SELECT * INTO CustomersBackup2017 FROM Customers;
```

INSERT INTO

```
INSERT INTO suppliers(ID, Name) SELECT ID, Name FROM customer where ID > 3;
```

Case

```
SELECT ID, Name,
CASE
  WHEN ID = 1 THEN "ID is 1"
  WHEN ID = 2 THEN "ID is 2"
  ELSE "ELSE ID"
END AS ID_STATUS
FROM customer;
```

IFNULL()

```
SELECT Name, Address , COALESCE(Name, 0) FROM customer;
```

`or`

```
SELECT Name, Address , IFNULL(Name, 0) FROM customer;
```

Single Line Comment

```
SELECT * from customer -- WHERE id = 2;;
```

Operators

ARITHMETIC OPERATORS

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulo

BitWise Operators

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR

SQL Comparison Operators

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

SQL Compound Operators

Operator	Description
+=	Add equals
-=	Subtract equals
*=	Multiply equals
/=	Divide equals
%=	Modulo equals
&=	Bitwise AND equals
^-=	Bitwise exclusive equals
|*=	Bitwise OR equals

SQL Logical Operators

Operator	Description
ALL	TRUE if all of the subquery values meet the condition
AND	TRUE if all the conditions separated by AND is TRUE
ANY	TRUE if any of the subquery values meet the condition
BETWEEN	TRUE if the operand is within the range of comparisons
EXISTS	TRUE if the subquery returns one or more records
IN	TRUE if the operand is equal to one of a list of expressions
LIKE	TRUE if the operand matches a pattern
NOT	Displays a record if the condition(s) is NOT TRUE
OR	TRUE if any of the conditions separated by OR is TRUE

Operator	Description
SOME	TRUE if any of the subquery values meet the condition

ALTER Table

The `ALTER TABLE` statement is used to add, delete, or modify columns in an existing table.

The `ALTER TABLE` statement is also used to add and drop various constraints on an existing table.

Alter table ADD column

```
ALTER TABLE customer ADD Status VARCHAR(255);
```

Alter table RENAME column

```
ALTER TABLE Employees_detail RENAME COLUMN Emp_age to age_of_Emp;
```

Alter table change constraint

```
ALTER TABLE customer MODIFY COLUMN customer.Status INT;
```

Drop Column

```
ALTER TABLE customer DROP COLUMN Status;
```