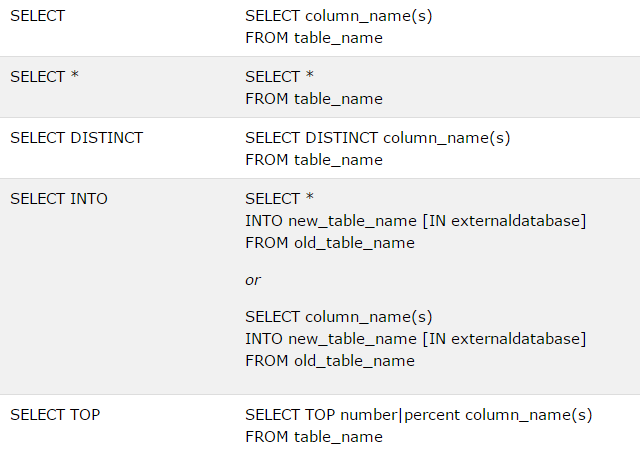
SELECT Syntax



**Example: SELECT:**

SELECT column1, column2, ...

FROM table\_name;

SELECT \* FROM table;

SELECT CustomerName, City From Customer;

**EXAMPLE: SELECT DISTINCT**

SELECT DISTINCT City from Customers;

SELECT Count(DISTINCT City) FROM Customers;

Using subquery:

SELECT Count(\*) as DistinctCities

From (SELECT DISTINCT City FROM Customers);

**WHERE Syntax:**

SELECT column1,column2, ...  
FROM table\_name  
WHERE condition;

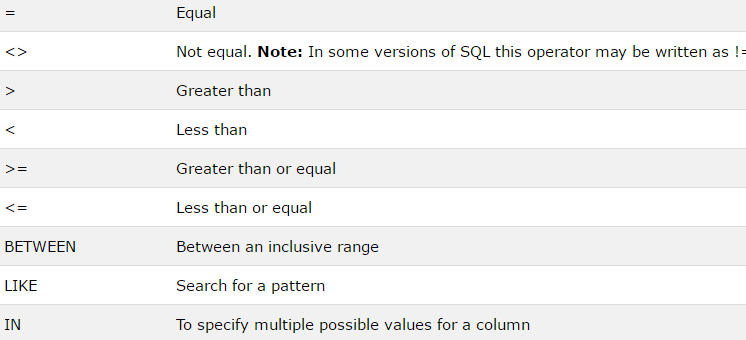
Select Distinct City

FROM Customers

WHERE Country = ‘Mexico’ or Country = ‘USA’

SELECT \* FROM Customers

WHERE CutomerID = 1;



# **SQL AND, OR and NOT Operators**

### AND Syntax

SELECT column1,column2, ...  
FROM table\_name  
WHERE condition1 AND condition2 AND condition3 ...;

### OR Syntax

SELECT column1,column2, ...  
FROM table\_name  
WHERE condition1 OR condition2 OR condition3 ...;

### NOT Syntax

SELECT column1,column2, ...  
FROM table\_name  
WHERE NOT condition;

# All customer from Kansas City

SELECT \* FROM Customer

WHERE country = ‘USA’ AND city = ‘Kansas City’;

# All customer from Kansas are Overland Park

SELECT \* FROM Customer

WHERE country =’USA’ AND (city = ‘Kansas City’ OR city = ‘Overland Park’)

SELECT \* FROM Customers

WHERE NOT Country =’USA’

**The following SQL statement selects all fields from "Customers" where country is NOT "Germany" and NOT "USA":**

SELECT \* FROM Customers

WHERE NOT Country =’USA’ and NOT Country =’Germany’

## **The SQL ORDER BY Keyword**

SELECT column1,column2, ...  
FROM table\_name  
ORDER BY column1, column2, ...ASC|DESC;

All customers from the "Customers" table, sorted by the "Country" column:

SELECT CustomerName, City, Address

FROM Customers

ORDER BY County ASC|DESC;

**Count of all these records**

SELECT Count(\*) AS CustomerRecords

FROM (SELECT CustomerName, City, Address

FROM Customers

ORDER BY County ASC|DESC);

**Following SQL statement selects all customers from the "Customers" table, sorted ascending by the "Country" and descending by the "CustomerName" column:**

SELECT \* Customers

ORDER BY County ASC, CustomerName DESC;

### INSERT INTO Syntax

INSERT INTO table\_name (column1,column2,column3, ...)  
VALUES (value1,value2,value3, ...);

Following SQL statement inserts a new record in the "Customers" table:

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

**Inserting Multiple Values:**

INSERT INTO Customers (Title, CustomerName, Address, City, Country, ZipCode)

VALUES

(‘Mr’, ‘Ravi’, ‘Check ST’, ‘Chicago’, ‘USA’, ‘64110’),

(‘Mr’, ‘Ravi’, ‘Check ST’, ‘Chicago’, ‘USA’, ‘64110’);

INSERT INTO CustomerLog (CalledTime, Title, CustomerName, Address, City, PostalCode, Coutry)

VALUES (DEFAULT, ‘Mr’, ‘Rav’, NULL, ‘Chicago’, NULL, ‘USA’)

***CREATE TABEL Customer.CallLog***

***(***

***CallID int IDENTITY PRIMARY KEY NOT NULL,***

***CallTime datatime NOT NULL Default GETDATE(),***

***SalesPerson nvarchar(100) NOT NULL,***

***CustomerID int NOT NULL REFERENCE Customer.Customers(CustomerID),***

***PhoneNumber nvarchar(25) NOT NULL,***

***Notes nvarchar(max) NULL***

***);***

***GO***

-- Insert a row

INSERT INTO Customer.CallLog

VALUES

('2015-01-01T12:30:00', 'SalesP/Goomes', 1, '245-555-0173', 'Returning call re: enquiry and delivery');

-- Insert **defaults** and **nulls**

INSERT INTO Customer.CallLog

VALUES

(DEFAULT, 'SalesP/david8', 2, '170-555-0127', NULL);

-- Insert a row with **explicit columns**

INSERT INTO Customer.CallLog (SalesPerson, CustomerID, PhoneNumber)

VALUES

('SalesP\jillian0', 3, '279-555-0130');

-- Insert multiple rows

INSERT INTO Customer.CallLog

VALUES

(DATEADD(mi,-2, GETDATE()), 'adventure-works\jillian0', 4, '710-555-0173', NULL),

(DEFAULT, 'adventure-works\shu0', 5, '828-555-0186', 'Called to arrange deliver of order 10987');

-- Insert the **results** of a query

INSERT INTO Customer.CallLog (SalesPerson, CustomerID, PhoneNumber, Notes)

SELECT SalesPerson, CustomerID, Phone, 'Sales promotion call'

FROM SalesLT.Customer

WHERE CompanyName = 'Big-Time Bike Store';

## **What is a NULL Value?**

A field with a NULL value is a field with no value.

If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value.

## How to Test for NULL Values?

SELECT column\_namesFROM table\_name  
WHERE column\_name IS NULL;

SELECT column\_namesFROM table\_name  
WHERE column\_name IS NOT NULL;

# All customers without address

SELECT \* From Customers

WHERE Address IS NULL;

# SQL statement uses the IS NOT NULL operator to list all persons that do have an address:

SELECT LastName, FirstName, Address FROM Persons

WHERE Address IS NOT NULL;

### UPDATE Syntax

UPDATE table\_name  
SET column1=value1,column2=value2, ...  
WHERE condition;

## UPDATE Table

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

## UPDATE Multiple Records

UPDATE Customers  
SET ContactName='Juan'  
WHERE Country='Mexico';

## Update Warning!

UPDATE Customers  
SET ContactName='Juan';

-- Update from results of a query

UPDATE SalesLT.CallLog

SET SalesPerson = c.SalesPerson, PhoneNumber = c.Phone

FROM SalesLT.Customer AS c

WHERE c.CustomerID = SalesLT.CallLog.CustomerID;

UPDATE Customer.CallLog

SET SalesPerson = c.SalesPerson, PhoneNumber =c.Phone

FROM Customer.Customers AS C

WHERE c.CustomerID = Customer.CallLog.CustomerID;

# **SQL DELETE Statement**

-- Delete rows : Keps last 7 day records

DELETE FROM SalesLT.CallLog

WHERE CallTime < DATEADD(dd, -7, GETDATE());

# **SQL TOP, LIMIT or ROWNUM Clause**

SELECT TOP number|percent column\_name(s)  
FROM table\_nameWHERE condition;

SELECT TOP 3 \* FROM Customers;

SELECT TOP 50 PERCENT \* FROM Customers

## ADD a WHERE CLAUS

SELECT TOP 3 \* FROM Customers  
WHERE Country='Germany';

# SQL MIN() and MAX() Functions

The MIN() function returns the smallest value of the selected column.

The MAX() function returns the largest value of the selected column.

SELECT MIN(Price) AS SmallestPrice  
FROM Products;

SELECT MAX(Price) AS LargestPrice  
FROM Products;

# **SQL COUNT(), AVG() and SUM() Functions**

SELECT COUNT(column\_name)  
FROM table\_name  
WHERE condition;

SELECT AVG(column\_name)  
FROM table\_name  
WHERE condition

SELECT SUM(column\_name)  
FROM table\_name  
WHERE condition;

# **SQL LIKE Operator**

There **are two wildcards** used in conjunction with the LIKE operator:

* % - The percent sign represents zero, one, or multiple characters
* \_ - The underscore represents a single character

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"

SELECT \* FROM Customers  
WHERE CustomerName LIKE 'a%';

***CustomerName that have "or" in any position:***

SELECT \* FROM Customers  
WHERE CustomerName LIKE '%or%';

SQL statement selects all customers with a CustomerName that NOT starts with "a":

SELECT \* FROM Customers  
WHERE CustomerName NOT LIKE 'a%';

# **SQL Wildcards**

A wildcard character is used to substitute any other character(s) in a string.

There are two wildcards used in conjunction with the LIKE operator:

* % - The percent sign represents zero, one, or multiple characters
* \_ - The underscore represents a single character

## Using the % Wildcard

## All customers with a City starting with "ber":

## SELECT \* FROM Customers WHERE City LIKE 'ber%';

City containing the pattern "es":

## SELECT \* FROM Customers WHERE City LIKE '%es%';

## Using the \_ Wildcard

City starting with any character, followed by "erlin":

SELECT \* FROM Customers  
WHERE City LIKE '\_erlin';

City starting with "L", followed by any character, followed by "n", followed by any character, followed by "on":

SELECT \* FROM Customers  
WHERE City LIKE 'L\_n\_on';

## Using the [charlist] Wildcard

The following SQL statement selects all customers with a City starting with "b", "s", or "p":

SELECT \* FROM Customers  
WHERE City LIKE '[bsp]%';

Selects all customers with a City starting with "a", "b", or "c":

SELECT \* FROM Customers  
WHERE City LIKE '[a-c]%';

## Using the [!charlist] Wildcard

The two following SQL statements selects all customers with a **City NOT starting** with "b", "s", or "p":

SELECT \* FROM Customers  
WHERE City LIKE '[!bsp]%';

SELECT \* FROM Customers  
WHERE City NOT LIKE '[bsp]%';

## **The SQL IN Operator**

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

The IN operator is a shorthand for multiple OR conditions.

### IN Syntax

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_name IN (value1,value2, ...);

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_name IN (SELECTSTATEMENT);

All customers that are located in "Germany", "France" and "UK":

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

NOT located in "Germany", "France" or "UK":

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

Same countries as the suppliers: (Sub quires) : Selecting customers from same countries as suppliers.

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

# **SQL BETWEEN Operator**

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.

The BETWEEN operator is inclusive: begin and end values are included.

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_nameBETWEEN value1 AND value2;

To display the products outside the range of the previous example, use NOT BETWEEN

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

## BETWEEN with IN Example

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

## Text Values Example

All products with a ProductName BETWEEN 'Carnarvon Tigers' and 'Mozzarella di Giovanni':

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

SELECT \* FROM Products  
WHERE ProductName NOT BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni'  
ORDER BY ProductName;

OrderDate BETWEEN '04-July-1996' and '09-July-1996':

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

## **SQL Aliases**

SQL aliases are used to give a table, or a column in a table, a temporary name.

Aliases are often used to make column names more readable.

An alias only exists for the duration of the query.

### Alias Column Syntax

SELECT column\_name AS alias\_name  
FROM table\_name;

Example:

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;

### Alias Table Syntax

SELECT column\_name(s)  
FROM table\_nameAS alias\_name;

Example:

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;

**Without Alias:**

SELECT Orders.OrderID, Orders.OrderDate, Customers.CustomerName  
FROM Customers, Orders  
WHERE Customers.CustomerName="Around the Horn" AND Customers.CustomerID=Orders.CustomerID;

Aliases can be useful when:

* There are more than one table involved in a query
* Functions are used in the query
* Column names are big or not very readable
* Two or more columns are combined together

# **SQL Joins**

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

## **Different Types of SQL JOINs**

Here are the different types of the JOINs in SQL:

* **(INNER) JOIN**: Returns records that have matching values in both tables
* **LEFT (OUTER) JOIN**: Return all records from the left table, and the matched records from the right table
* **RIGHT (OUTER) JOIN**: Return all records from the right table, and the matched records from the left table
* **FULL (OUTER) JOIN**: Return all records when there is a match in either left or right table

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

# **SQL INNER JOIN Keyword**

The INNER JOIN keyword selects records that have matching values in both tables.

SELECT column\_name(s)  
FROM table1  
INNER JOIN table2ON table1.column\_name=table2.column\_name;

Example:

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

## JOIN Three Tables

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);

## **SQL LEFT JOIN Keyword**

The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match.

SELECT column\_name(s)  
FROM table1  
LEFT JOIN table2ON table1.column\_name=table2.column\_name;

Example:

SELECT Customers.CustomerName, Orders.OrderID  
FROM Customers  
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID  
ORDER BY Customers.CustomerName;

# **SQL RIGHT JOIN Keyword**

The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match.

SELECT column\_name(s)  
FROM table1  
RIGHT JOIN table2ON table1.column\_name=table2.column\_name;

Example:

SELECT Orders.OrderID, Employees.LastName, Employees.FirstName  
FROM Orders  
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID  
ORDER BY Orders.OrderID;

## **SQL FULL OUTER JOIN Keyword**

The FULL OUTER JOIN keyword return all records when **there is a match in either** left (table1) or right (table2) table records.

SELECT column\_name(s)  
FROM table1  
FULL OUTER JOIN table2ON table1.column\_name=table2.column\_name;

SELECT Customers.CustomerName, Orders.OrderID  
FROM Customers  
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID  
ORDER BY Customers.CustomerName;

## **SQL Self JOIN**

A self JOIN is a regular join, but the table is joined with itself.

SELECT column\_name(s)  
FROM table1 T1, table1 T2  
WHERE condition;

SQL statement matches customers that are from the same city:

SELECT A.CustomerName AS CustomerName1, B.CustomerName AS CustomerName2, A.City  
FROM Customers A, Customers B  
WHERE A.CustomerID <> B.CustomerID  
AND A.City = B.City   
ORDER BY A.City;

# **SQL UNION Operator**

## **The SQL UNION Operator**

The UNION operator is used to combine the result-set of two or more SELECT statements.

* Each SELECT statement within UNION must have the same number of columns
* The columns must also have similar data types
* The columns in each SELECT statement must also be in the same order

SELECT column\_name(s) FROM table1  
UNION  
SELECT column\_name(s) FROM table2;

SQL statement selects all the different cities **(only distinct values)** from "Customers" and "Suppliers":

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

*SQL statement selects all cities (duplicate values also) from "Customers" and "Suppliers":*

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

# **SQL GROUP BY Statement**

SELECT column\_name(s)  
FROM table\_name  
WHERE condition  
GROUP BY column\_name(s)ORDER BY column\_name(s);

SELECT COUNT(CustomerID), Country  
FROM Customers  
GROUP BY Country;

SQL statement lists the number of customers in each country, sorted high to low

SELECT COUNT(CustomerID), Country  
FROM Customers  
GROUP BY Country  
ORDER BY COUNT(CustomerID) DESC;

# **SQL HAVING Clause**

SELECT column\_name(s)  
FROM table\_name  
WHERE condition  
GROUP BY column\_name(s)HAVING conditionORDER BY column\_name(s);

SQL statement lists the number of customers in each country. Only include countries with more than 5 customers:

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

SQL statement lists the number of customers in each country, sorted high to low (Only include countries with more than 5 customers):

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

SQL statement lists the employees that have registered more than 10 orders:

SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders  
FROM (Orders  
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID)  
GROUP BY LastName  
HAVING COUNT(Orders.OrderID) > 10;

# **SQL EXISTS Operator**

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 column\_name(s)  
FROM table\_name  
WHERE EXISTS  
(SELECT column\_nameFROM table\_name WHERE condition);

SELECT SupplierName  
FROM Suppliers  
WHERE EXISTS (SELECT ProductName FROM Products WHERE SupplierId = Suppliers.supplierId AND Price < 20);

## **SQL Database**

# **SQL CREATE DATABASE Statement**

CREATE DATABASE databasename;

## **DROP DATABASE Example**

The following SQL statement drops the existing database "testDB

DROP DATABASE testDB;

# **SQL CREATE TABLE Statement**

CREATE TABLE table\_name(  
    column1 datatype,  
    column2 datatype,  
    column3 datatype,  
   ....  
);

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

CREATE TABLE SalesLT.CallLog

(

CallID int IDENTITY PRIMARY KEY NOT NULL,

CallTime datetime NOT NULL DEFAULT GETDATE(),

SalesPerson nvarchar(256) NOT NULL,

CustomerID int NOT NULL REFERENCES SalesLT.Customer(CustomerID),

PhoneNumber nvarchar(25) NOT NULL,

Notes nvarchar(max) NULL

);

GO

## **Create Table Using Another Table**

CREATE TABLE new\_table\_name AS  
    SELECT column1, column2,...  
    FROM existing\_table\_name  
    WHERE ....;

# **SQL Constraints**

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

Constraints can be **specified when the table is created** with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement.

CREATE TABLE table\_name(  
    column1 datatype constraint,  
    column2 datatype constraint,  
    column3 datatype constraint,  
    ....  
);

* [**NOT NULL**](https://www.w3schools.com/sql/sql_notnull.asp) - Ensures that a column cannot have a NULL value
* [**UNIQUE**](https://www.w3schools.com/sql/sql_unique.asp) - Ensures that all values in a column are different
* [**PRIMARY KEY**](https://www.w3schools.com/sql/sql_primarykey.asp) - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
* [**FOREIGN KEY**](https://www.w3schools.com/sql/sql_foreignkey.asp) - Uniquely identifies a row/record in another table
* [**CHECK**](https://www.w3schools.com/sql/sql_check.asp) - Ensures that all values in a column satisfies a specific condition
* [**DEFAULT**](https://www.w3schools.com/sql/sql_default.asp) - Sets a default value for a column when no value is specified
* [**INDEX**](https://www.w3schools.com/sql/sql_create_index.asp) - Use to create and retrieve data from the database very quickly

# **SQL DROP TABLE Statement**

DROP TABLE table\_name;

## **SQL TRUNCATE TABLE**

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

TRUNCATE TABLE table\_name;

# **SQL ALTER TABLE Statement**

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 table\_name  
ADD column\_name datatype;

**Example:**

ALTER TABLE Persons  
ADD DateOfBirth date;

## **ALTER TABLE - DROP COLUMN**

ALTER TABLE table\_name  
DROP COLUMN column\_name;

Example:

ALTER TABLE Persons  
DROP COLUMN DateOfBirth;

**ALTER TABLE - ALTER/MODIFY COLUMN**

To change the data type of a column in a table, use the following syntax:

ALTER TABLE table\_name  
ALTER COLUMN column\_name datatype;

Example:

ALTER TABLE Persons  
ALTER COLUMN DateOfBirth year;

# **SQL NOT NULL Constraint**

By default, a column can hold NULL values.

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  
);

# **SQL UNIQUE Constraint**

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

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

A PRIMARY KEY constraint automatically has a UNIQUE constraint.

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

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

## **SQL UNIQUE Constraint on ALTER TABLE**

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

ALTER TABLE Persons  
ADD UNIQUE (ID);

To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax

ALTER TABLE Persons  
ADD CONSTRAINT UC\_Person UNIQUE (ID,LastName);