❖**What is SQL?**

* SQL stands for Structured Query Language
* SQL lets you access and manipulate databases
* SQL keywords are NOT case sensitive Ex. SELECT as select
* Semicolon is the standard way to separate each SQL statement in database systems. ex. Select \* from employee;
* SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

❖**What Can SQL do?**

* SQL can execute queries against a database
* SQL can retrieve data from a database
* SQL can insert records in a database
* SQL can update records in a database
* SQL can delete records from a database
* SQL can create new databases
* SQL can create new tables in a database
* SQL can create stored procedures in a database
* SQL can create views in a database
* SQL can set permissions on tables, procedures, and views

❖**SELECT Statement:**

* The **SELECT** statement is used to select data from a database.
* The data returned is stored in a result table, called the result-set.
  + 1. **Syntax to fetch particular columns :**

SELECT *column1*,*column2, ...*FROM *table\_name*;

**e.g.** SELECT CustomerName, City FROM Customers;

e.g select SupplierID,Unit from products;

* + 1. **Syntax to fetch all columns :**

SELECT \* FROM *table\_name*;

**e.g.** SELECT \* FROM Customers;

❖**SELECT DISTINCT Statement :**

* The **SELECT DISTINCT** statement is used to return only distinct (different, uniq) values.
* Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

* **Syntax:**

1. SELECT DISTINCT column1, column2, ...FROM table\_name;

**e.g.** SELECT DISTINCT Country FROM Customers;

2. SELECT COUNT(DISTINCT column\_name) FROM table\_name;

**e.g.** SELECT COUNT(DISTINCT Country) FROM Customers;

❖**WHERE Clause :**

* The **WHERE** clause is used to filter records.
* It is used to extract only those records that ful fill a specified condition.
* **Syntax:**

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

**e.g.**  SELECT \* FROM Customers. WHERE Country='Mexico';

2. SELECT \* FROM table\_name WHERE condition;

**e.g.** SELECT \* FROM Customers WHERE CustomerID=1;

**Note:-**Text Fields vs. Numeric Fields

* SQL requires single quotes around text values (most database systems will also allow double quotes).
* However, numeric fields should not be enclosed in quotes:

**Operators in The WHERE Clause :**

**Operator Description**

1. = Equal

2. > Greater than

3. < Less than

4. >= Greater than or equal

5. <= Less than or equal

6. <> Not equal. Note: In some versions of SQL this operator may be written as !=

7. BETWEEN. Between a certain range

8. LIKE Search for a pattern

9. IN To specify multiple possible values for a column

**e.g.**

1. SELECT \* FROM Products WHERE Price = 18; //equal

2. SELECT \* FROM Products WHERE Price > 30; //greater than

3. SELECT \* FROM Products WHERE Price < 30; //less than

4. SELECT \* FROM Products WHERE Price >= 30; //greater than equal

5. SELECT \* FROM Products WHERE Price <= 30;

6. SELECT \* FROM Products WHERE Price <> 18;

7. SELECT \* FROM Products WHERE Price BETWEEN 50 AND 60;

8. SELECT \* FROM Customers WHERE City LIKE 's%';

9. SELECT \* FROM Customers WHERE City IN ('Paris','London');

10. SELECT \* FROM Customers where country In ("Germany", "Spain","Mexico","UK")

10. select \* from products where price not between 10 and 12 ;

❖**SQL AND, OR and NOT Operators :**

* The AND and OR operators are used to filter records based on more than one condition.

**AND Operator :**

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

* **Syntax**

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

* **e.g.** SELECT \* FROM Customers  
   WHERE Country='Germany' AND City='Berlin';

**OR Operator :**

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

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

* **e.g.** SELECT \* FROM Customers  
   WHERE City='Berlin' OR City='München';

**NOT Operator :**

* The **NOT operator** displays a record if the condition(s) is NOT TRUE.
* **Syntax**

SELECT column1, column2, ...FROM table\_name

WHERE NOT condition;

* **e.g.** SELECT \* FROM Customers WHERE NOT Country='Germany';
* **Combining AND, OR and NOT**

**e.g.** 1. SELECT \* FROM Customers WHERE Country='Germany'

AND (City='Berlin' OR City='Munchen');

2. SELECT \* FROM Customers WHERE NOT Country='Germany'

AND NOT Country='USA';

❖**ORDER BY Keyword :**

* The ORDER BY keyword is used to sort the result-set in ascending or descending order.
* The ORDER BY keyword sorts the records in ascending order by default.
* To sort the records in descending order, use the DESC keyword.
* **Syntax :**

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

Select \* from table\_name order by column\_name;

Select \* from table\_name order by column\_name desc;

* **e.g.** 1. SELECT \* FROM Customers ORDER BY Country;

2. SELECT \* FROM Customers ORDER BY Country DESC;

3. SELECT \* FROM Customers ORDER BY Country, CustomerName;

4. SELECT \* FROM Customers ORDER BY Country ASC,

CustomerName DESC;

5. select \* from products order by SupplierID desc, price desc;

❖**INSERT INTO Statement :**

* The INSERT INTO statement is used to insert new records in a table.
* It is possible to write the INSERT INTO statement in two ways:

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

**Syntax :**

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

**e.g.**

INSERT INTO Customers (CustomerName, ContactName, Address,

City,PostalCode,Country)

VALUES ('Cardinal','Tom B.Erichsen','Skagen 21','Stavanger',

'4006', 'Norway');

❖**SQL NULL Values :**

**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.
* **IS NULL Syntax**

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

**e.g.**

SELECT CustomerName, ContactName, Address FROM Customers  
 WHERE Address IS NULL;

* **IS NOT NULL Syntax**

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

**e.g.**

SELECT CustomerName, ContactName, Address FROM Customers  
 WHERE Address IS NOT NULL;

❖ **UPDATE Statement :**

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

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

**e.g.**

UPDATE Customers SET ContactName = 'Alfred Schmidt',

City='Frankfurt' WHERE CustomerID = 1;

* **Update Warning!**
* Be careful when updating records. If you omit the WHERE clause, ALL records will be updated!

**e.g.** UPDATE Customers SET ContactName=’Dhiraj’;

❖**DELETE Statement :**

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

DELETE FROM table\_name WHERE condition;

**e.g.**

DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';

* **Delete All Records**
* 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:
* **Syntax :** DELETE FROM table\_name;

**e.g.** DELETE FROM Customers;

❖**SELECT TOP Clause :**

* The SELECT TOP clause is used to specify the number of records to return.
* **Syntax : SQL Server/MS Access**

SELECT TOP number|*percent* column\_name(s) FROM table\_name

WHERE condition;

**e.g.**

1. SELECT TOP 3 \* FROM Customers;

2. SELECT TOP 50 PERCENT \* FROM Customers;

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

* **Syntax : MySQL**

SELECT column\_name(s) FROM table\_nameWHERE condition

LIMIT number;

**e.g.**

1.SELECT \* FROM Customers LIMIT 3;

2.SELECT \* FROM Customers WHERE Country='Germany' LIMIT 3;

3. SELECT \* FROM [Customers] where CustomerID between 12 and 20

4. SELECT \* FROM [Customers] where CustomerID 72,5 (last 5 records)

5. SELECT \* FROM [Products] where ProductID between 74 and 77

6. SELECT \* FROM [Products] where limit 74,3

❖ **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.
* **MIN() Syntax**

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

**e.g.**

1.SELECT MIN (Price) FROM Products;

2.SELECT Price FROM Products ORDER BY Price LIMIT 5 ; //--MIN 5 VALUES

3. select min (price) from products

* **MAX() Syntax**

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

**e.g.**

1. SELECT MAX (Price) FROM Products;

2. SELECT Price FROM Products ORDER BY Price DESC LIMIT 5 ;//TOP 5 VALUES

3. select max (price) from products

❖**COUNT(), AVG() and SUM() Functions :**

* The COUNT() function returns the number of rows that matches a specified criterion.
* The AVG() function returns the average value of a numeric column.
* The SUM() function returns the total sum of a numeric column.
* **COUNT() Syntax :**

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

**e.g.**

SELECT COUNT(ProductID) FROM Products;

* **AVG() Syntax :**

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

**e.g.**

SELECT AVG(Price) FROM Products;

* **SUM() Syntax :**

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

**e.g.**

SELECT SUM(Quantity) FROM OrderDetails;

❖**LIKE Operator :**

* The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.
* **Syntax :**

SELECT column1, column2, ...FROM table\_name WHERE column\_name

LIKE pattern;

**1.** **WHERE CustomerName LIKE 'a%' :** Finds any values that start with "a"

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

**2.** **WHERE CustomerName LIKE '%a' :** Finds any values that end with "a"

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

**3. WHERE CustomerName LIKE '%or%' :** Finds any values that have "or" in any

position

**e.g.** SELECT \* FROM Customers WHERE CustomerName LIKE '%or%'

**4. WHERE CustomerName LIKE '\_r%' :** Finds any values that have "r" in the second

position

**e.g.** SELECT \* FROM Customers WHERE CustomerName LIKE '\_r%';

**5. WHERE CustomerName LIKE 'a\_%' :** Finds any values that start with "a" and are

at least 2 characters in length

**e.g.** SELECT \* FROM Customers WHERE CustomerName LIKE 'a\_\_%';

**6. WHERE ContactName LIKE 'a%o' :** Finds any values that start with "a" and ends

with "o"

**e.g.** SELECT \* FROM Customers WHERE ContactName LIKE 'a%o';

❖**Wildcard Characters :**

* A wildcard character is used to substitute one or more characters in a string.

e.g. SELECT \* FROM Customers WHERE City LIKE 'ber%';

* Wildcard characters are used with the LIKE  operator. The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

❖**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.
* **Syntax**

SELECT column\_name(s) FROM table\_name WHERE column\_name

IN (value1, value2, ...);

**e.g.**

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

2. SELECT \* FROM Customers WHERE Country

NOT IN ('Germany', 'France', 'UK');

❖**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.
* **Syntax**

SELECT column\_name(s) FROM table\_name

WHERE column\_name BETWEEN value1 AND value2;

**e.g.**

1. SELECT \* FROM Products WHERE Price BETWEEN 10 AND 20;

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

3. SELECT \* FROM Products WHERE Price BETWEEN 10 AND 20

AND CategoryID NOT IN (1,2,3);

4. SELECT \* FROM Products WHERE ProductName BETWEEN 'Carnarvon Tigers'

AND 'Mozzarella di Giovanni' ORDER BY ProductName;

5. SELECT \* FROM Products WHERE ProductName NOT BETWEEN

'Carnarvon Tigers' AND 'Mozzarella di Giovanni' ORDER BY ProductName;

6. SELECT \* FROM Orders WHERE OrderDate BETWEEN '1996-07-01'

AND '1996-07-31';

❖**SQL Aliases :**

* SQL aliases are used to give a table name, or a column name in a table, a temporary name.
* An alias only exists for the duration of that query.
* An alias is created with the AS keyword.ss
* **Alias Column Syntax**

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

**e.g.**

SELECT CustomerID AS ID, CustomerName AS Customer FROM Customers;

* **Alias Table Syntax---you don’t have an authority to change the table name**

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

**e.g.**

SELECT o.OrderID, o.OrderDate, c.CustomerName FROM

Customers AS c, Orders AS o WHERE c.CustomerName='Around the Horn'

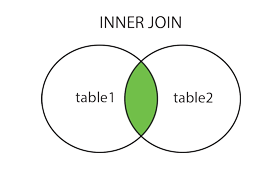
AND c.CustomerID=o.CustomerID;

❖**SQL JOIN :**

* A JOIN clause is used to combine rows from two or more tables, based on a related column between them.
* There are four types of joins

**1. INNER JOIN Keyword:**

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



* **Syntax :**

SELECT column\_name(s) FROM table1 INNER JOIN table2

ON table1.column\_name = table2.column\_name;

**e.g.**

1. SELECT OrderID,CustomerName FROM Orders INNER JOIN Customers

ON Orders.CustomerID = Customers.CustomerID;

* we match primary key with foreign key. first column id is primarykey and in another table it is refer as foreign key

2. SELECT \* FROM Orders INNER JOIN Customers

ON Orders.CustomerID = Customers.CustomerID;

3. SELECT \* FROM customers INNER JOIN orders

ON Orders.CustomerID = Customers.CustomerID;

4. SELECT \* FROM customers INNER JOIN orders

ON Customers.CustomerID = Orders.CustomerID;

* **JOIN Three Tables:**

SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName

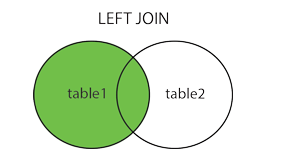
FROM ((Orders INNER JOIN Customers ON Orders.CustomerID =

Customers.CustomerID) INNER JOIN Shippers ON Orders.ShipperID =

Shippers.ShipperID);

**2. LEFT JOIN Keyword:**

* The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.



* **Syntax :**

SELECT column\_name(s) FROM table1 LEFT JOIN table2

ON table1.column\_name = table2.column\_name;

**e.g.**

1. SELECT Customers.CustomerName, Orders.OrderID FROM Customers

LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID

ORDER BY Customers.CustomerName;

2. SELECT CustomerName, OrderID FROM Customers LEFT JOIN Orders

ON Customers.CustomerID = Orders.CustomerID

ORDER BY Customers.CustomerName;

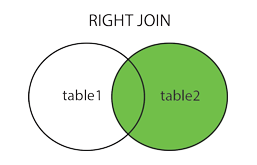
3. SELECT \* FROM Customers LEFT JOIN Orders

ON Customers.CustomerID = Orders.CustomerID

ORDER BY Customers.CustomerName;

**3. RIGHT JOIN Keyword :**

* The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.



* **Syntax :**

SELECT column\_name(s) FROM table1 RIGHT JOIN table2

ON table1.column\_name = table2.column\_name;

**e.g.**

1. SELECT Orders.OrderID, Employees.LastName, Employees.FirstName

FROM Orders RIGHT JOIN Employees

ON Orders.EmployeeID = Employees.EmployeeID ORDER BY Orders.OrderID;

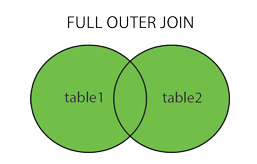
2. SELECT Orders.OrderID, Employees.LastName, Employees.FirstName

FROM Orders RIGHT JOIN Employees

ON Orders.EmployeeID = Employees.EmployeeID ;

**4. FULL OUTER JOIN Keyword :**

* The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.
* **Tip:** FULL OUTER JOIN and FULL JOIN are the same.



* **Syntax :**

SELECT column\_name(s) FROM table1 FULL OUTER JOIN table2

ON table1.column\_name = table2.column\_name WHERE condition;

**e.g.**

SELECT Customers.CustomerName, Orders.OrderID FROM Customers

FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID

ORDER BY Customers.CustomerName;

❖**UNION Operator :**

* The UNION operator is used to combine the result-set of two or more SELECT statements.
* The columns must also have similar data types
* **Syntax :**

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

FROM table2;

**e.g.**

SELECT City FROM Customers UNION SELECT City FROM Suppliers

ORDER BY City;

❖**UNION ALL Operator :**

* The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL.
* **Syntax :**

SELECT column\_name(s) FROM table1 UNION ALL SELECT column\_name(s)

FROM table2;

**e.g.**

SELECT City FROM Customers UNION ALL SELECT City FROM Suppliers

ORDER BY City;

❖ **GROUP BY Statement :**

* The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.
* **Syntax :**

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

**e.g.**

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

select count(CustomerName), country from customers group by country;

❖ **HAVING Clause :**

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

SELECT column\_name(s) FROM table\_name GROUP BY column\_name(s)

HAVING condition;

**e.g.**

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

HAVING COUNT(CustomerID) > 5;

SELECT count(price), SupplierID FROM products GROUP BY SupplierID ;

SELECT max(price), SupplierID FROM products GROUP BY SupplierID ;

SELECT min(price), SupplierID FROM products GROUP BY SupplierID ;

❖**SELECT INTO Statement :**

* The SELECT INTO statement copies data from one table into a new table.
* **Copy all columns into a new table:**

**Syntax :**

SELECT \*INTO newtable [IN externaldb] FROM oldtable WHERE condition;

**e.g.**

1. SELECT \* INTO CustomersBackup2017 FROM Customers;

2. SELECT \* INTO CustomersGermany FROM Customers

WHERE Country = 'Germany';

❖**CREATE DATABASE Statement :**

* The CREATE DATABASE statement is used to create a new SQL database.
* **Syntax :** CREATE DATABASE databasename;

**e.g.** CREATE DATABASE testDB;

❖**DROP DATABASE Statement :**

* The DROP DATABASE statement is used to drop an existing SQL database.
* **Syntax :** DROP DATABASE databasename;

**e.g.** DROP DATABASE Dhiraj;

❖ **CREATE TABLE Statement :**

* 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,);

**e.g.** CREATE TABLE Persons (PersonID int,LastName varchar(255),

FirstName varchar(255),Address varchar(255),City varchar(255));

❖**DROP TABLE Statement :**

* The DROP TABLE statement is used to drop an existing table in a database.
* **Syntax :** DROP TABLE table\_name;

**e.g.** DROP TABLE Shippers;

❖**TRUNCATE TABLE :**

* The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.
* **Syntax :** TRUNCATE TABLE table\_name;

**e.g.** TRUNCATE TABLE orders ;

❖ **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**

**To add a column in a table, use the following syntax:**

ALTER TABLE table\_name ADD column\_name datatype;

**e.g.** ALTER TABLE Customers ADD Email varchar(255);

* **ALTER TABLE - DROP COLUMN**

**To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):**

ALTER TABLE table\_name DROP COLUMN column\_name;

**e.g.** ALTER TABLE Customers DROP COLUMN Email;

❖**SQL Constraints :**

* 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.
* Constraints 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:**

**1. NOT NULL** - Ensures that a column cannot have a NULL value

**2. UNIQUE -** Ensures that all values in a column are different

**3. PRIMARY KEY -** A combination of a NOT NULL and UNIQUE. Uniquely

identifies each row in a table

**4. FOREIGN KEY -** Prevents actions that would destroy links between tables

**5. CHECK -** Ensures that the values in a column satisfies a specific condition

**6. DEFAULT -** Sets a default value for a column if no value is specified

* 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.

## 1. NOT NULL Constraint :

* By default, a column can hold NULL values.
* The NOT NULL constraint enforces a column to NOT accept NULL values.
* This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.
  1. **SQL NOT NULL on CREATE TABLE**

**Syntax :**

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

**e.g.**

CREATE TABLE Persons (

ID int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255) NOT NULL,

Age int ) ;

* 1. **SQL NOT NULL on ALTER TABLE**

**Syntax :**

ALTER TABLE tableName

MODIFY columnName data\_type constraintName;

**e.g.**

ALTER TABLE Persons  
 MODIFY Age int NOT NULL;

## 2. 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.

1. **UNIQUE Constraint on CREATE TABLE :**

**Syntax :**

* UNIQUE constraint on single columns, use the following SQL syntax:

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

* UNIQUE constraint on multiple columns, use the following SQL syntax:

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

1. **UNIQUE Constraint on ALTER TABLE :**

* ADD UNIQUE constraint on single columns, use the following SQL syntax:

ALTER TABLE Persons  
 ADD UNIQUE (ID);

* ADD UNIQUE constraint on multiple columns, use the following SQL syntax:

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

* To drop a UNIQUE constraint, use the following SQL Syntax :

ALTER TABLE Persons  
 DROP INDEX UC\_Person;

## 3. PRIMARY KEY Constraint :

* The PRIMARY KEY constraint uniquely identifies each record in a table.
* Primary keys must contain UNIQUE values, and cannot contain NULL values.
* A table can have only ONE primary key and in the table, this primary key can consist of single or multiple columns (fields).

**1. PRIMARY KEY on CREATE TABLE :**

* PRIMARY KEY constraint on single columns, use the following SQL syntax:

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

* PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    CONSTRAINT PK\_Person PRIMARY KEY (ID,LastName)  
);

* **Note:** In the 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).

**2. PRIMARY KEY on ALTER TABLE :**

* ADD PRIMARY KEY constraint on single columns, use the following SQL syntax:

ALTER TABLE Persons  
 ADD PRIMARY KEY (ID);

* ADD PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

ALTER TABLE Persons  
 ADD CONSTRAINT PK\_Person PRIMARY KEY (ID,LastName);

* To drop a PRIMARY KEY constraint, use the following SQL Syntax:

ALTER TABLE Persons  
 DROP PRIMARY KEY;

## 4. FOREIGN KEY Constraint :

* The FOREIGN KEY constraint is used to prevent 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](https://www.w3schools.com/sql/sql_primarykey.asp) in another table.
* The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

1. **FOREIGN KEY on CREATE TABLE :**

* FOREIGN KEY constraint on single columns, use the following SQL syntax:

CREATE TABLE Orders (  
    OrderID int NOT NULL,  
    OrderNumber int NOT NULL,  
    PersonID int,  
    PRIMARY KEY (OrderID),  
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)  
);

* FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

CREATE TABLE Orders (  
    OrderID int NOT NULL,  
    OrderNumber int NOT NULL,  
    PersonID int,  
    PRIMARY KEY (OrderID),  
    CONSTRAINT FK\_PersonOrder FOREIGN KEY (PersonID)  
    REFERENCES Persons(PersonID)  
);

1. **FOREIGN KEY on ALTER TABLE :**

* ADD FOREIGN KEY constraint on single columns, use the following SQL syntax:

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

* ADD FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

ALTER TABLE Orders ADD CONSTRAINT FK\_PersonOrder  
FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);

* To drop a FOREIGN KEY constraint, use the following SQL Syntax:

ALTER TABLE Orders DROP FOREIGN KEY FK\_PersonOrder;

## 5. CHECK Constraint :

* The CHECK constraint is used to limit the value range that can be placed in a column.
* If you define a CHECK constraint on a column it will allow only certain values for this column.
* If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

1. **CHECK constraint on CREATE TABLE :**

* CHECK constraint on single columns, use the following SQL syntax:

CREATE TABLE Persons ( ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),Age int, CHECK (Age>=18)  
);

* CHECK constraint on multiple columns, use the following SQL syntax:

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')  
);

1. **CHECK constraint on ALTER TABLE :**

* ADD CHECK constraint on single columns, use the following SQL syntax:

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

* ADD CHECK constraint on multiple columns, use the following SQL syntax:

ALTER TABLE Persons ADD CONSTRAINT CHK\_PersonAge

CHECK (Age>=18 AND City='Sandnes');

* To drop a CHECK constraint, use the following SQL Syntax:

ALTER TABLE Persons DROP CHECK CHK\_PersonAge;

6. DEFAULT Constraint :

* The DEFAULT constraint is used to set a default value for a column.
* The default value will be added to all new records, if no other value is specified.

1. **DEFAULT constraint on CREATE TABLE :**

* DEFAULT constraint on single columns, use the following SQL syntax:

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    City varchar(255) DEFAULT 'Sandnes'  
);

* The DEFAULT constraint can also be used to insert system values, by using functions like [GETDATE()](https://www.w3schools.com/sql/func_sqlserver_getdate.asp):

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

1. **DEFAULT constraint on ALTER TABLE :**

* ADD DEFAULT constraint on single columns, use the following SQL syntax:

ALTER TABLE Persons ALTER City SET DEFAULT 'Sandnes';

* To drop a DEFAULT constraint, use the following SQL Syntax:

ALTER TABLE Persons ALTER City DROP DEFAULT;

7. CREATE INDEX Constraint :

* The CREATE INDEX statement is used to create indexes in tables.
* Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.
* CREATE INDEX Syntax :
* Creates an index on a table. Duplicate values are allowed:

CREATE INDEX index\_name  
 ON table\_name (column1, column2, ...);

**e.g.**

1.CREATE INDEX idx\_lastname ON Persons (LastName);

2.CREATE INDEX idx\_pname ON Persons (LastName, FirstName);

### CREATE UNIQUE INDEX Syntax :

* Creates a unique index on a table. Duplicate values are not allowed:

CREATE UNIQUE INDEX index\_name  
 ON table\_name (column1, column2, ...);

## DROP INDEX Statement

* The DROP INDEX statement is used to delete an index in a table.

ALTER TABLE table\_name DROP INDEX index\_name;

❖**AUTO INCREMENT Field :**

* Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.
* Often this is the primary key field that we would like to be created automatically every time a new record is inserted.
* **Syntax :**

CREATE TABLE Persons (  
    Personid int NOT NULL AUTO\_INCREMENT,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    PRIMARY KEY (Personid)  
);

❖**CREATE VIEW Statement :**

* In SQL, a view is a virtual table based on the result-set of an SQL statement.
* A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.
* You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.
* A view is created with the CREATE VIEW statement.
* **Syntax :**

CREATE VIEW view\_name AS SELECT column1, column2, ...  
 FROM table\_name WHERE condition;

* **Examples :**

1. CREATE VIEW [Brazil Customers] AS SELECT CustomerName,

ContactName FROM Customers WHERE Country = 'Brazil';

We can also write it as :

SELECT \* FROM [Brazil Customers];

1. CREATE VIEW [Products Above Average Price] AS SELECT

ProductName, Price FROM Products WHERE Price > (SELECT AVG(Price) FROM Products);

We can also write it as :

SELECT \* FROM [Products Above Average Price];

## SQL Updating a View

A view can be updated with the **CREATE OR REPLACE VIEW** statement.

**Syntax :**

CREATE OR REPLACE VIEW view\_name AS SELECT column1, column2, ...  
 FROM table\_name WHERE condition;

**e.g. :**

CREATE OR REPLACE VIEW [Brazil Customers] AS  
 SELECT CustomerName, ContactName, City FROM Customers  
 WHERE Country = 'Brazil';

## SQL Dropping a View

A view is deleted with the **DROP VIEW** statement.

**Syntax :** DROP VIEW view\_name;

**e.g. :** DROP VIEW [Brazil Customers];