Some of The Most Important SQL Commands

* **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 Statement

The SELECT statement is used to select data from a database.

SELECT column1, column2, FROM table\_name;

If you want to select all the fields available in the table, use the following syntax

SELECT \* FROM *table\_name*;

If you want to select two column like( name, city ) from customer table

SELECT Name, City FROM Customers;

SELECT DISTINCT Statement

SELECT DISTINCT column1, column2, FROM table\_name;

Select statement only the distinct value from the “Country ” column in the “Customer” table

SELECT DISTINCT Country FROM Customers;

WHERE Clause

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

statement selects all the customers from the country "Mexico", in the "Customers" table

SELECT \* FROM Customers WHERE Country='Mexico';

SELECT \* FROM Customers WHERE CustomerID=1;

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

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

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

AND, OR and NOT Operators

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

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

### **AND Syntax**

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

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

### **OR Syntax**

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

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

SELECT \* FROM Customers WHERE Country='Germany' OR Country='Spain';

### **NOT Syntax**

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

SELECT \* FROM Customers WHERE NOT Country='Germany';

(Not print Country Germany)

## 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;

### **MAX() Syntax**

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

## MIN() Example

The following SQL statement finds the price of the cheapest product:

SELECT MIN(Price) AS SmallestPrice FROM Products;

## MAX() Example

The following SQL statement finds the price of the most expensive product:

SELECT MAX(Price) AS LargestPrice FROM Products;

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

The COUNT() function returns the number of rows that matches a specified criteria.

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;

**AVG() Syntax**

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

### **SUM() Syntax**

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

## COUNT() Example

The following SQL statement finds the number of products:

SELECT COUNT(ProductID) FROM Products;

**Note:** NULL values are not counted.

## AVG() Example

The following SQL statement finds the average price of all products:

SELECT AVG(Price) FROM Products;

**Note:** NULL values are ignored.

## SUM() Example

The following SQL statement finds the sum of the "Quantity" fields in the "OrderDetails" table:

SELECT SUM(Quantity) FROM OrderDetails;

**ORDER BY Keyword**

SELECT \* FROM Customers ORDER BY Country;

SELECT \* FROM CustomersORDER BY Country DESC;

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

**INSERT INTO Statement**

The first way specifies both the column names and the values to be inserted:

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

If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. The INSERT INTO syntax would be as follows:

INSERT INTO *table\_name* VALUES (*value1*,*value2*,*value3*, ...);

## UPDATE Statement

## UPDATE Table

The following SQL statement updates the first customer (CustomerID = 1) with a new contact person and a new city.

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

The following SQL statement will update the contactname to "Juan" for all records where country is "Mexico":

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

## DELETE Statement

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

## DELETE FROM table\_name WHERE condition;

The following SQL statement deletes the customer "Alfreds Futterkiste" from the "Customers" table:

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:

DELETE FROM *table\_name*;

The following SQL statement deletes all rows in the "Customers" table, without deleting the table:

DELETE FROM Customers;

## JOIN

Notice that the "CustomerID" column in the "Orders" table refers to the "CustomerID" in the "Customers" table. The relationship between the two tables above is the "CustomerID" column.

Then, we can create the following SQL statement (that contains an INNER JOIN), that selects records that have matching values in both tables:

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

## 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

      

## INNER JOIN Keyword

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

### INNER JOIN Syntax

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

The following SQL statement selects all orders (table) with customer (table) information:

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

## 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.

### LEFT JOIN Syntax

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

**Note:** In some databases LEFT JOIN is called LEFT OUTER JOIN.

## LEFT JOIN Example

The following SQL statement will select all customers (table), and any orders(table) they might have:

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

**Note:** The LEFT JOIN keyword returns all records from the left table (Customers), even if there are no matches in the right table (Orders).

## 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.

### RIGHT JOIN Syntax

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

**Note:** In some databases RIGHT JOIN is called RIGHT OUTER JOIN.

## RIGHT JOIN Example

The following SQL statement will return all employees (table), and any orders (table) they might have placed:

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

## 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.

**Note:** FULL OUTER JOIN can potentially return very large result-sets!

**Tip:** FULL OUTER JOIN and FULL JOIN are the same.

### **FULL OUTER JOIN Syntax**

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

## FULL OUTER JOIN Example

The following SQL statement selects all customers, and all orders:

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

**Note:** The FULL OUTER JOIN keyword returns all matching records from both tables whether the other table matches or not. So, if there are rows in "Customers" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Customers", those rows will be listed as well.

## CREATE DATABASE Statement

The CREATE DATABASE statement is used to create a new SQL database.

CREATE DATABASE databasename;

## DROP DATABASE Statement

The DROP DATABASE statement is used to drop an existing SQL database.

DROP DATABASE databasename;

## CREATE TABLE Statement

The CREATE TABLE statement is used to create a new table in a database.

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.

DROP TABLE table\_name;

## 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,  
    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)  
);

## 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);

## 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).

## PRIMARY KEY on CREATE TABLE

The following SQL creates a PRIMARY KEY on the "ID" column when the "Persons" table is created:

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

To allow naming of a PRIMARY KEY constraint, and for defining a 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).

## FOREIGN KEY Constraint

A FOREIGN KEY is a key used to link two tables together.

A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

## FOREIGN KEY on CREATE TABLE

The following SQL creates a FOREIGN KEY on the "PersonID" column when the "Orders" table is created:

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

To allow naming of a FOREIGN KEY constraint, and for defining a 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)  
);