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

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

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

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

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

### INNER JOIN Syntax

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

## **JOIN Three Tables**

The following SQL statement selects all orders with customer and shipper information:

### Example

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.

### LEFT JOIN Syntax

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

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



## **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.

### RIGHT JOIN Syntax

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

## **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.

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

### FULL OUTER JOIN Syntax

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



## **SQL Self JOIN**

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

### Self JOIN Syntax

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

### Example

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;

## **The SQL 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.

### GROUP BY Syntax

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

## **The SQL HAVING Clause**

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

### HAVING Syntax

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

## **The 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.

### EXISTS Syntax

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

1. CREATE DATABASE databasename;
2. DROP DATABASE databasename;
3. CREATE TABLE table\_name (  
       column1 datatype,  
       column2 datatype,  
       column3 datatype,  
      ....  
   );
4. ALTER TABLE table\_name  
   ADD column\_name datatype;
5. ALTER TABLE table\_name  
   DROP COLUMN column\_name;
6. ALTER TABLE table\_name  
   MODIFY COLUMN column\_name datatype;

## **SQL Create Constraints**

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.

### Syntax

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

## **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:

* [**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 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.

## **SQL PRIMARY KEY Constraint**

The PRIMARY KEY constraint uniquely identifies each record in a database table.

Primary keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only one primary key, which may consist of single or multiple fields.

## **SQL 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

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

The FOREIGN KEY constraint also prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.

## **SQL FOREIGN KEY on CREATE TABLE**

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

|  |  |  |  |
| --- | --- | --- | --- |
| Person ID | Last Name | First Name | Age |
| 1 | hans | Buhlar | 30 |
| 2 | tan | Stephan | 32 |
| 3 | Joerg | Peter | 33 |

|  |  |  |
| --- | --- | --- |
| OrderID | OrderNumber | PersonID |
| 1 | 343345 | 3 |
| 2 | 535455 | 2 |
| 3 | 777421 | 1 |

**MySQL:**

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

## **SQL FOREIGN KEY on ALTER TABLE**

To create a FOREIGN KEY constraint on the "PersonID" column when the "Orders" table is already created, use the following SQL:

**MySQL / SQL Server / Oracle / MS Access:**

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

## **DROP a FOREIGN KEY Constraint**

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

**MySQL:**

ALTER TABLE Orders  
DROP FOREIGN KEY FK\_PersonOrder;

## **SQL 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 single column it allows 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.

## **SQL CHECK on CREATE TABLE**

The following SQL creates a CHECK constraint on the "Age" column when the "Persons" table is created. The CHECK constraint ensures that you can not have any person below 18 years:

**MySQL:**

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