* 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 Syntax**

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

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

SELECT \* FROM table\_name;

syntax:

🡪Here, column1, column2, ... are the field names of the table you want to select data from.

SELECT column1, column2, ...

FROM table\_name;

**SELECT DISTINCT syntax**

The SELECT DISTINCT statement is used to return only distinct (different) values.

Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

SELECT DISTINCT column1, column2, ...

FROM table\_name;

### **WHERE Syntax**

The WHERE clause is used to filter records.

It is used to extract only those records that fulfill a specified condition.

Note: The WHERE clause is not only used in SELECT statements, it is also used in UPDATE, DELETE, etc.!

syntax:

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

## The SQL AND, OR and NOT Operators

The WHERE clause can be combined with 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.

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

### **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;

## The SQL 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.

### **ORDER BY Syntax**

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

## The SQL INSERT INTO Statement

The INSERT INTO statement is used to insert new records in a table.

### **INSERT INTO Syntax**

It is possible to write the INSERT INTO statement in two ways:

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

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

2. 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. Here, the INSERT INTO syntax would be as follows:

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

## 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.

**Note:** A NULL value is different from a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation!

### **IS NULL Syntax**

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

### **IS NOT NULL Syntax**

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

## The SQL UPDATE Statement

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

### **UPDATE Syntax**

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

## The SQL DELETE Statement

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

### **DELETE Syntax**

DELETE FROM table\_name WHERE condition;

### **MIN() Syntax**

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

### **MAX() Syntax**

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

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

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

### **COUNT() Syntax**

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

The AVG() function returns the average value of a numeric column.

### **AVG() Syntax**

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

The SUM() function returns the total sum of a numeric column.

### **SUM() Syntax**

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

## The SQL LIKE Operator

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

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

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

**Note:** MS Access uses an asterisk (\*) instead of the percent sign (%), and a question mark (?) instead of the underscore (\_).

The percent sign and the underscore can also be used in combinations!

### **LIKE Syntax**

SELECT column1, column2, ...  
FROM table\_name  
WHERE columnN LIKE pattern;

## 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, ...);

or:

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

## The 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.

### **BETWEEN Syntax**

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

## SQL Aliases

### **Alias Column Syntax**

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

### **Alias Table Syntax**

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

## SQL JOIN

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

Let's look at a selection from the "Orders" table:

|  |  |  |
| --- | --- | --- |
| **OrderID** | **CustomerID** | **OrderDate** |
| 10308 | 2 | 1996-09-18 |
| 10309 | 37 | 1996-09-19 |
| 10310 | 77 | 1996-09-20 |

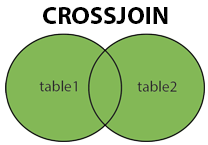
Then, look at a selection from the "Customers" table:

|  |  |  |  |
| --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mexico |

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

|  |  |  |
| --- | --- | --- |
| **OrderID** | **CustomerName** | **OrderDate** |
| 10308 | Ana Trujillo Emparedados y helados | 9/18/1996 |
| 10365 | Antonio Moreno Taquería | 11/27/1996 |
| 10383 | Around the Horn | 12/16/1996 |
| 10355 | Around the Horn | 11/15/1996 |
| 10278 | Berglunds snabbköp | 8/12/1996 |

* INNER JOIN: Returns records that have matching values in both tables
* LEFT JOIN: Returns all records from the left table, and the matched records from the right table
* RIGHT JOIN: Returns all records from the right table, and the matched records from the left table
* CROSS JOIN: Returns all records from both tables

### **INNER JOIN Syntax**

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

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

### **LEFT JOIN Syntax**

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

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.

### **CROSS JOIN Syntax**

SELECT column\_name(s)  
FROM table1  
CROSS JOIN table2 WHERE condition;

### **Example**

SELECT Customers.CustomerName, Orders.OrderID  
FROM Customers  
CROSS JOIN Orders  
WHERE Customers.CustomerID=Orders.CustomerID;

### **RIGHT JOIN Syntax**

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

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.

### **Self Join Syntax**

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

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

## The SQL UNION Operator

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

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

### **UNION Syntax**

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

### **UNION ALL Syntax**

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

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

## The SQL GROUP BY Statement

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

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 cannot 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);

## The SQL ANY and ALL Operators

The ANY and ALL operators allow you to perform a comparison between a single column value and a range of other values.

## The SQL ANY Operator

The ANY operator:

* returns a boolean value as a result
* returns TRUE if ANY of the subquery values meet the condition

ANY means that the condition will be true if the operation is true for any of the values in the range.

### **ANY Syntax**

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

**Note:** The operator must be a standard comparison operator (=, <>, !=, >, >=, <, or <=).

## The SQL ALL Operator

The ALL operator:

* returns a boolean value as a result
* returns TRUE if ALL of the subquery values meet the condition
* is used with SELECT, WHERE and HAVING statements

ALL means that the condition will be true only if the operation is true for all values in the range.

### **ALL Syntax With SELECT**

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

### **ALL Syntax With WHERE or HAVING**

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

**Note:** The operator must be a standard comparison operator (=, <>, !=, >, >=, <, or <=).

## The SQL SELECT INTO Statement

The SELECT INTO statement copies data from one table into a new table.

### **SELECT INTO Syntax**

Copy all columns into a new table:

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

Copy only some columns into a new table:

SELECT column1, column2, column3, ...  
INTO newtable [IN externaldb]  
FROM oldtableWHERE condition;

## The SQL INSERT INTO SELECT Statement

The INSERT INTO SELECT statement copies data from one table and inserts it into another table.

The INSERT INTO SELECT statement requires that the data types in source and target tables match.

**Note:** The existing records in the target table are unaffected.

### **INSERT INTO SELECT Syntax**

Copy all columns from one table to another table:

INSERT INTO table2  
SELECT \* FROM table1WHERE condition;

Copy only some columns from one table into another table:

INSERT INTO table2 (column1, column2, column3, ...)  
SELECT column1, column2, column3, ...  
FROM table1  
WHERE condition;

## The SQL CREATE DATABASE Statement

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

### **Syntax**

CREATE DATABASE databasename;

## The SQL DROP DATABASE Statement

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

### **Syntax**

DROP DATABASE databasename;

## The SQL 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,  
   ....  
);

## The SQL DROP TABLE Statement

The DROP TABLE statement is used to drop an existing table in a database.

### **Syntax**

DROP 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

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

ALTER TABLE *table\_name*  
ADD *column\_name datatype*;

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*;

ALTER TABLE - RENAME COLUMN

To rename a column in a table, use the following syntax:

ALTER TABLE *table\_name*  
RENAME COLUMN *old\_name* to *new\_name*;

## ALTER TABLE - ALTER/MODIFY DATATYPE

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) - Prevents actions that would destroy links between tables
* [CHECK](https://www.w3schools.com/sql/sql_check.asp) - Ensures that the values in a column satisfies a specific condition
* [DEFAULT](https://www.w3schools.com/sql/sql_default.asp) - Sets a default value for a column if no value is specified
* [CREATE INDEX](https://www.w3schools.com/sql/sql_create_index.asp) - Used to create and retrieve data from the database very quickly

## SQL NOT NULL on CREATE TABLE

The following SQL ensures that the "ID", "LastName", and "FirstName" columns will NOT accept NULL values when the "Persons" table is created:

### **Example**

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

## SQL NOT NULL on ALTER TABLE

To create a NOT NULL constraint on the "Age" column when the "Persons" table is already created, use the following SQL:

ALTER TABLE Persons  
MODIFY COLUMN Age int NOT NULL;

## SQL UNIQUE Constraint on CREATE TABLE

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

### **Example**

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

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

### **Example**

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

SQL PRIMARY KEY on CREATE TABLE

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

### **Example**

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:

### **Example**

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

## SQL 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.

SQL FOREIGN KEY on CREATE TABLE

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

**MySQL:**

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:

**MySQL**

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));

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**

ALTER TABLE Orders  
ADD 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:

**MySQL**

ALTER TABLE Orders  
ADD CONSTRAINT FK\_PersonOrder  
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 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 the age of a person must be 18, or older:

**MySQL:**

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

### CREATE INDEX Syntax

Creates an index on a table. Duplicate values are allowed:

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

### 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, ...);

## 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

## SQL 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.

### **CREATE VIEW Syntax**

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