

The SQL Statement

Q: How to see all the existing databases?  
A: SHOW DATABASES;.

Q: How to select a database to work with?  
A: USE database\_name;.

Q: How to see all the tables present in a database?  
A: SHOW TABLES;.

Q: How can you see the structure or details of a MySQL table?  
A: DESC table\_name;.

SELECT CustomerName, City FROM Customers;

SELECT COUNT(DISTINCT Country) FROM Customers;

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

SELECT \* FROM Customers   
WHERE Country = 'Spain' AND (CustomerName LIKE 'G%' OR CustomerName LIKE 'R%');

The SQL INSERT INTO Statement

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

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

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

How to Test for NULL Values?

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

IS NULL, IS NOT NULL;

The SQL UPDATE Statement

If you omit the WHERE clause, all records in the table will be updated!

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

The SQL DELETE Statement

DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';

LIMIT

Select the first 3 records of the Customers table:

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

SELECT \* FROM Customers  
ORDER BY CustomerName DESC  
LIMIT 3;

SQL Aggregate Functions

MIN(), MAX(), COUNT(), SUM(), AVG()

The SQL MIN() and MAX() Functions

SELECT MIN(Price) AS SmallestPrice, CategoryID  
FROM Products  
GROUP BY CategoryID;

The SQL COUNT() Function

Find the total number of rows in the Products table:

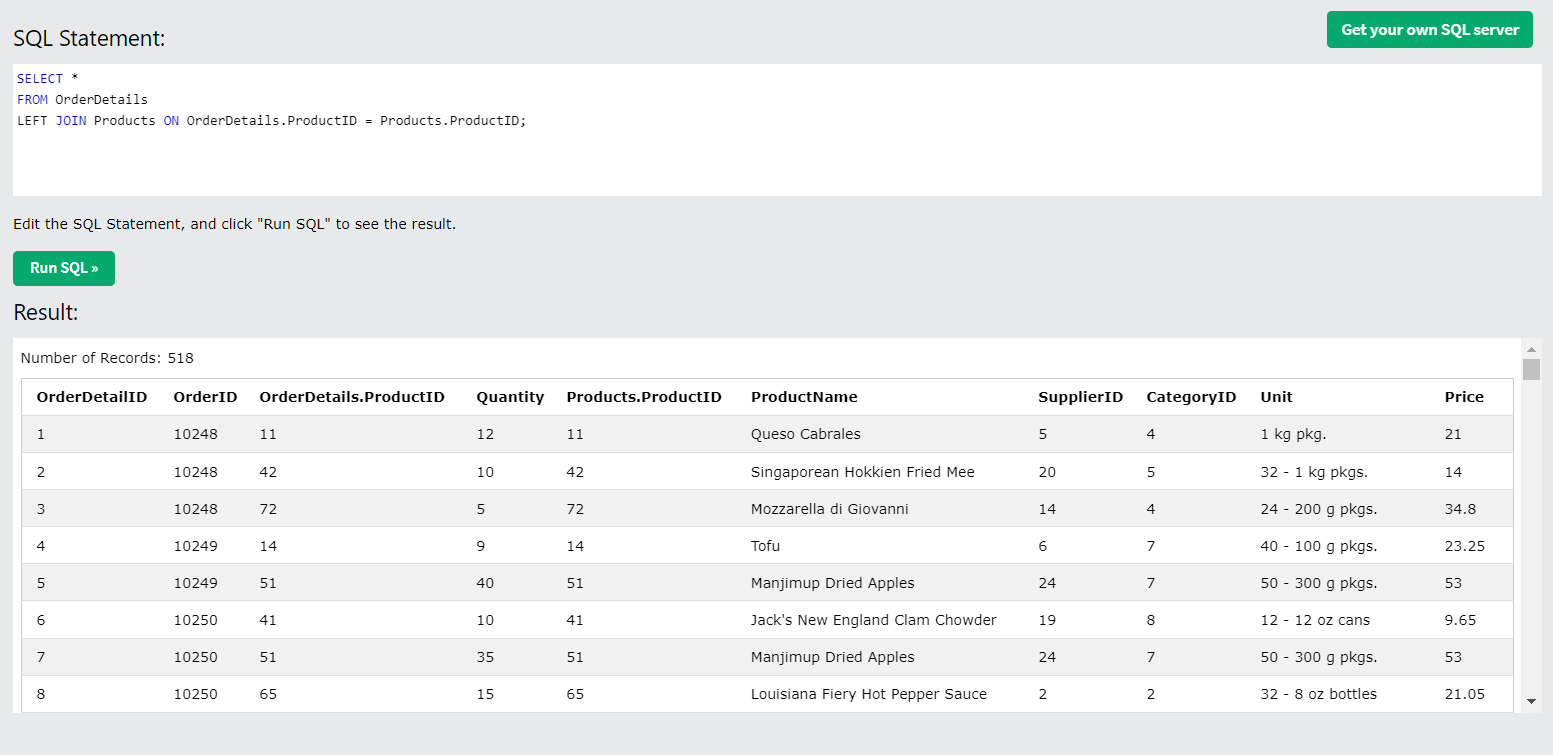
SELECT COUNT(\*)  
FROM Products;

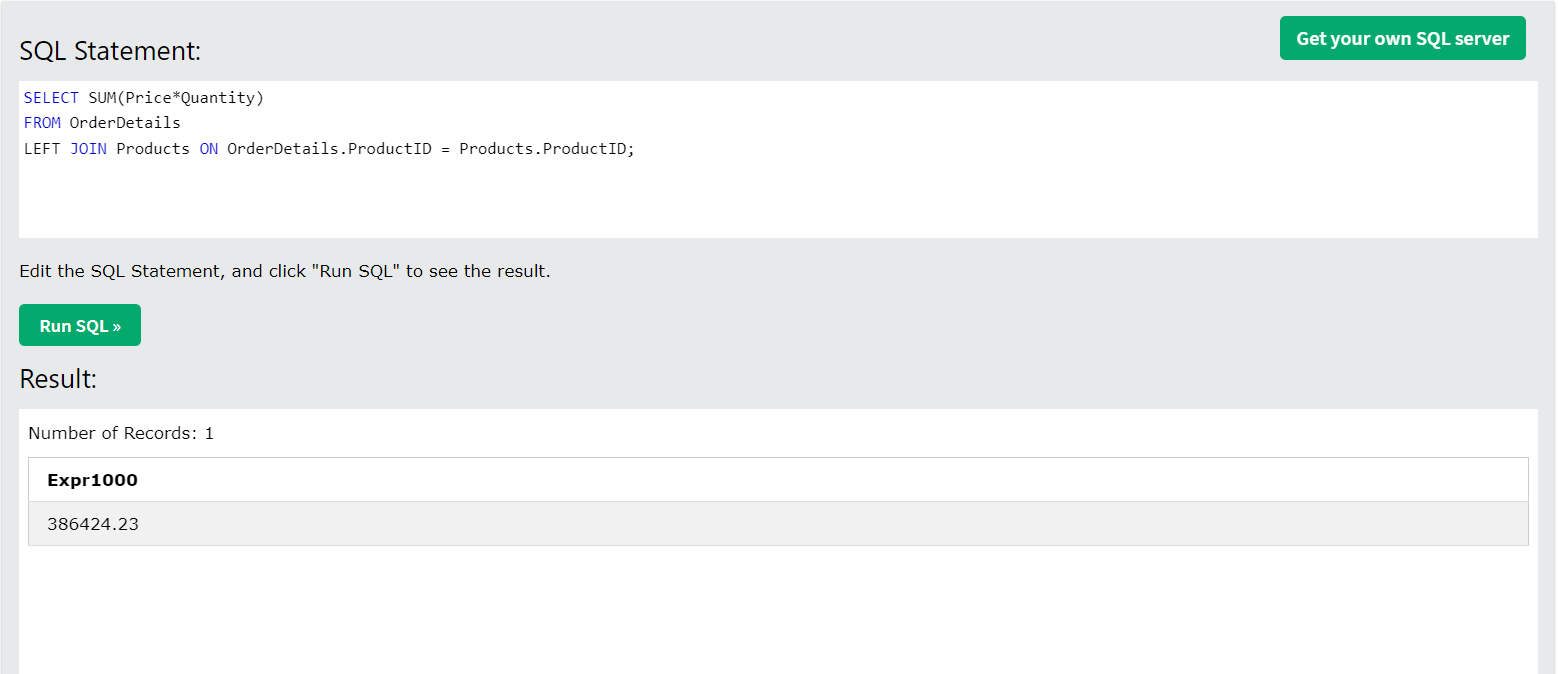
SELECT COUNT(\*) AS [Number of records], CategoryID  
FROM Products  
GROUP BY CategoryID;

The SQL SUM() Function

SELECT OrderID, SUM(Quantity)   
FROM OrderDetails  
GROUP BY OrderID;

Join OrderDetails with Products, and use SUM() to find the total amount:





The SQL AVG() Function

**Note:** NULL values are ignored.

SELECT \* FROM Products  
WHERE price > (SELECT AVG(price) FROM Products);

The average price for each category in the Products table:

SELECT AVG(Price) AS AveragePrice, CategoryID  
FROM Products  
GROUP BY CategoryID;

The SQL LIKE Operator

SELECT \* FROM Customers  
WHERE city LIKE 'L\_nd%';

LIKE, NOT LIKE

The SQL IN Operator

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

NOT IN, IN

Return all customers that have an order in the [**Orders**](https://www.w3schools.com/sql/trysql.asp?filename=trysql_orders) table:

SELECT \* FROM Customers  
WHERE CustomerID IN (SELECT CustomerID FROM Orders);

The SQL BETWEEN Operator

SELECT \* FROM Products  
WHERE Price BETWEEN 10 AND 20  
AND CategoryID IN (1,2,3);

NOT BETWEEN, NOT BETWEEN

**SQL Joins**

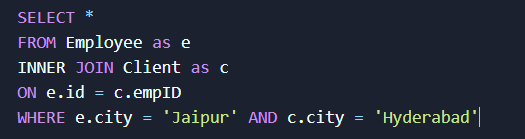
Here are all the main types of SQL joins:

1. **INNER JOIN**: Returns records with matching values in both tables.
2. **LEFT JOIN (or LEFT OUTER JOIN)**: Returns all records from the left table and matching records from the right table; if no match, returns NULL for right table.
3. **RIGHT JOIN (or RIGHT OUTER JOIN)**: Returns all records from the right table and matching records from the left table; if no match, returns NULL for left table.
4. **FULL JOIN (or FULL OUTER JOIN)**: Returns all records when there's a match in either table; non-matching rows in both tables are filled with NULL.
5. **CROSS JOIN**: Returns the Cartesian product of both tables (all possible combinations).
6. **SELF JOIN**: Joins a table to itself, useful for comparing rows within the same table.

INNER JOIN

SELECT ProductID, ProductName, CategoryName  
FROM Products  
INNER JOIN Categories ON Products.CategoryID = Categories.CategoryID;



SQL LEFT JOIN Keyword

A **left outer join** returns all rows from the left table and the matching rows from the right table. If no match is found in the right table, NULL values are returned.

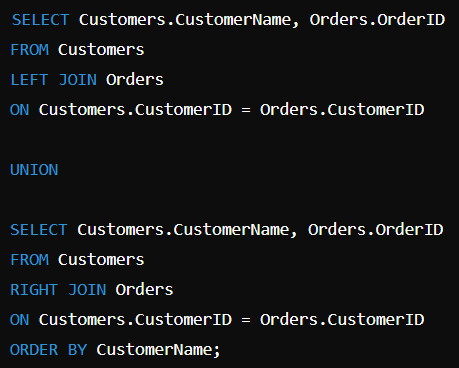
SELECT Customers.CustomerName, Orders.OrderID  
FROM Customers  
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID  
ORDER BY Customers.CustomerName;

SQL RIGHT JOIN Keyword

A **right outer join** returns all rows from the right table and the matching rows from the left table. If no match is found in the left table, NULL values are returned.

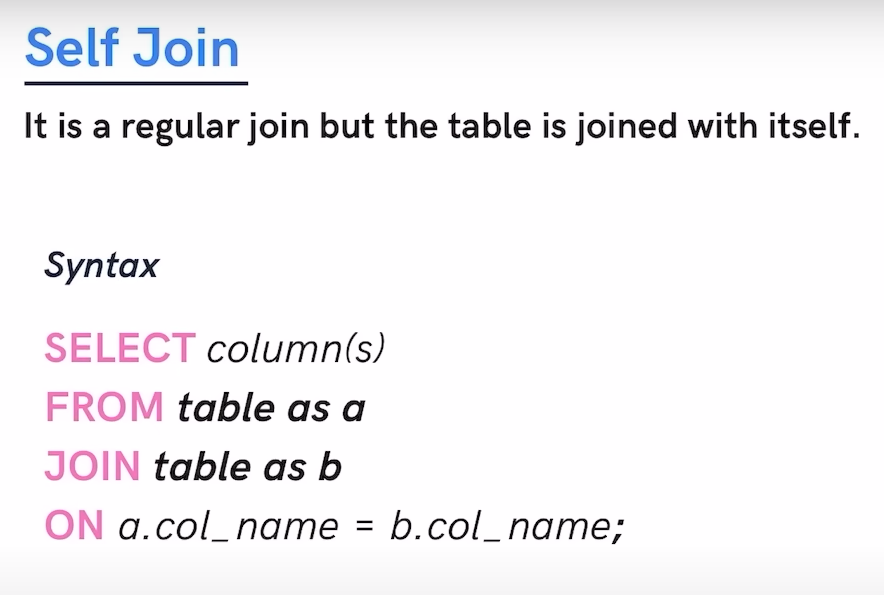
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName  
FROM Orders  
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID  
ORDER BY Orders.OrderID;

SQL FULL OUTER JOIN Keyword

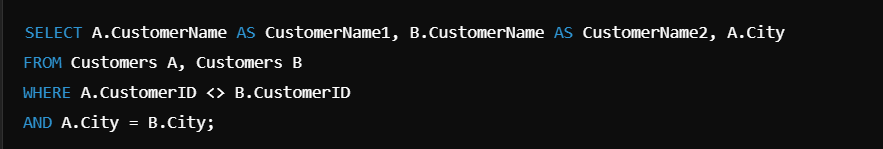
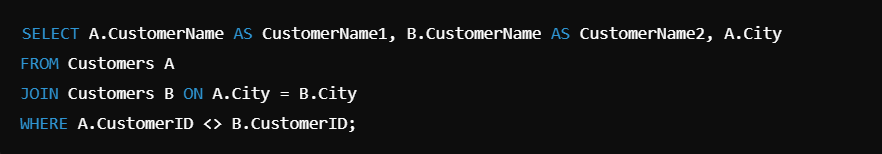


Note: The FULL OUTER JOIN returns all records from both tables, including unmatched rows from either table.

SQL Self Join



Customers that are from the same city:



CROSS JOIN: Returns the Cartesian product of both tables (i.e., all possible combinations of rows). No condition is applied between the two tables.

SELECT columns

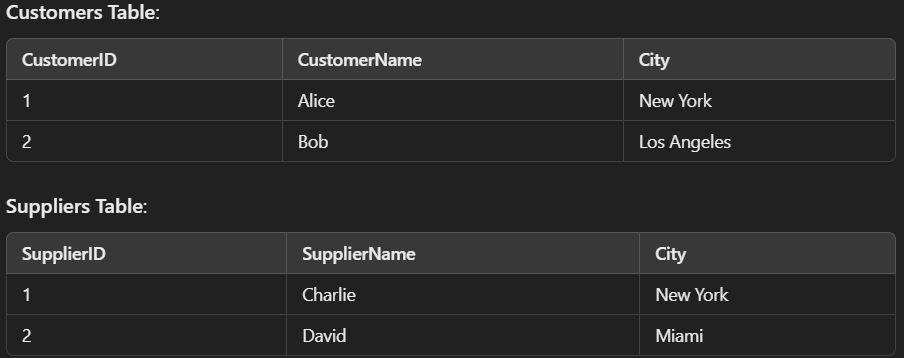
FROM table1

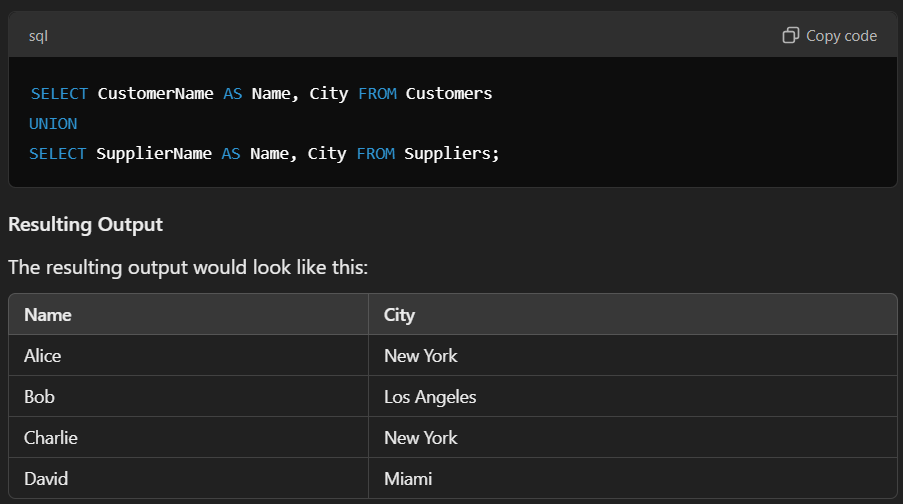
CROSS JOIN table2;

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





To allow duplicate values, use UNION ALL:

The SQL GROUP BY Statement

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;

The SQL HAVING Clause

The HAVING clause was added because WHERE keyword cannot be used with aggregate functions. WHERE filters records before any groupings are applied, while HAVING filters the grouped results.

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

Create the backup table structure:

CREATE TABLE CustomersBackup AS  
SELECT \*

FROM Customers;

Copy table structure:

CREATE TABLE CustomersBackup AS  
SELECT \*

FROM Customers;

WHERE 1=0;

SQL Database

CREATE DATABASE testDB;

DROP DATABASE testDB;

BACKUP DATABASE testDB  
TO DISK = 'D:\backups\testDB.bak';

## The SQL CREATE TABLE Statement

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

SQL Query Used To Create A Table With Same Structure Of Another Table.



SQL Query Used To Create A Table With Same Data Of Another Table



SQL DROP TABLE Example

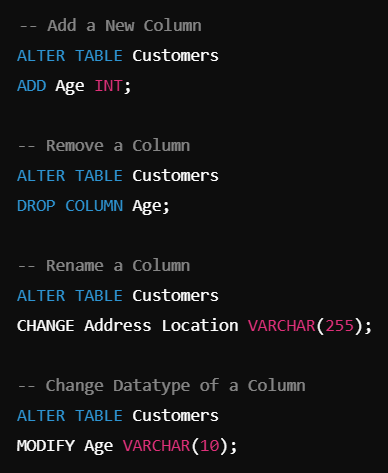
DROP TABLE Shippers;

SQL TRUNCATE TABLE

Delete the data inside a table, but not the table itself.

TRUNCATE TABLE table\_name;

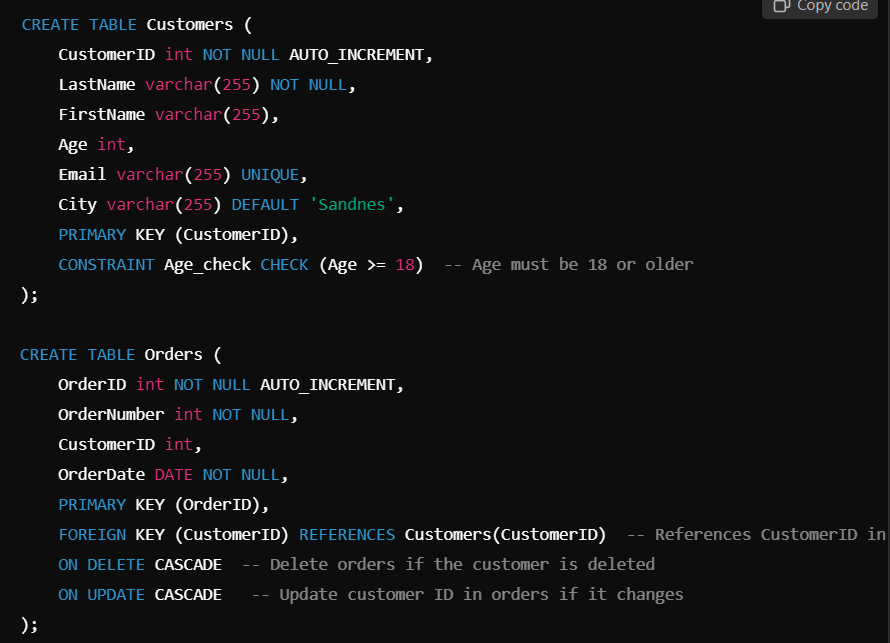
ALTER TABLE

****

SQL Constraints

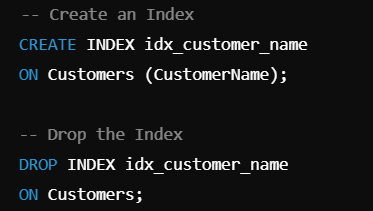
Constraints in SQL enforce rules on data in tables to ensure data accuracy, validity, and consistency. They prevent invalid data by restricting actions (e.g., setting unique values, ensuring non-null fields, enforcing relationships) and help maintain data integrity in the database.

[NOT NUL](https://www.w3schools.com/sql/sql_notnull.asp)L, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK, DEFAULT, CREATE INDEX



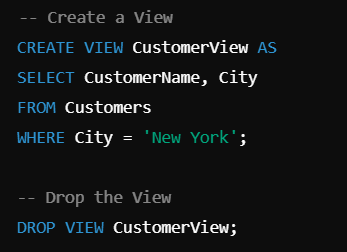
SQL CREATE INDEX Statement

An **index** improves data retrieval speed in SQL tables. It’s created on one or more columns and enhances performance for search, filter, and sort operations, but may slow down updates.



SQL CREATE VIEW Statement

A view is a virtual table that presents the result set of a stored SQL query.



SQL Injection

SQL injection is a type of cyberattack where malicious SQL code is inserted into an input field, often resulting in unauthorized access or data manipulation. To prevent SQL injection, you can:

* Use parameterized queries or prepared statements.
* Implement input validation and sanitization.

SELECT \* FROM Users WHERE UserId = 105 OR 1=1;

Input: 105 OR 1=1

Input: 105; DROP TABLE Suppliers;

### Prevention with Parameterized Queries

sql = "SELECT \* FROM Customers WHERE CustomerId = @0"; command.Parameters.AddWithValue("@0", txtUserId);

