MYSQL

```sql
SELECT CustomerName, City, Country FROM Customers;
```

```sql
SELECT DISTINCT Country FROM Customers;
```

```sql
SELECT COUNT(DISTINCT Country) FROM Customers;
```

```sql
SELECT * FROM Customers
WHERE Country = 'Mexico';
```

```sql
SELECT * FROM Products
WHERE Price BETWEEN 50 AND 60;
```

```sql
SELECT * FROM Customers
WHERE City LIKE 's%';
```

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

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

```sql
SELECT * FROM Customers
WHERE City = 'Berlin' OR City = 'Stuttgart';
```

```sql
SELECT * FROM Customers
WHERE NOT Country = 'Germany';
```

```sql
SELECT * FROM Customers
WHERE Country = 'Germany' AND (City = 'Berlin' OR City = 'Stuttgart');
```

```sql
SELECT * FROM Customers
ORDER BY Country;
```

```sql
SELECT * FROM Customers
ORDER BY Country DESC;
```

The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" and the "CustomerName" column. This means that it orders by Country, but if some rows have the same Country, it orders them by CustomerName:

Example

```sql
SELECT * FROM Customers
ORDER BY Country, CustomerName;
```

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

```sql
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');
```

```sql
INSERT INTO Customers (CustomerName, City, Country)
VALUES ('Cardinal', 'Stavanger', 'Norway');
```

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

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

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

Note: Be careful when updating records in a table! Notice the `WHERE` clause in the `UPDATE` statement. The `WHERE` clause specifies which record(s) that should be updated. If you omit the `WHERE` clause, all records in the table will be updated!

```
UPDATE Customers
SET PostalCode = 00000
WHERE Country = 'Mexico';
```

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

```
UPDATE Customers
SET PostalCode = 00000;
```

```
DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
```

```
DELETE FROM Customers;
```

The `LIMIT` clause is used to specify the number of records to return.

The `LIMIT` clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

```
SELECT * FROM Customers
LIMIT 3;
```

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

The `MIN()` function returns the smallest value of the selected column.

The `MAX()` function returns the largest value of the selected column.

```
SELECT MIN(Price) AS SmallestPrice
FROM Products;
```

```sql
SELECT MAX(Price) AS LargestPrice
FROM Products;
```

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

```sql
SELECT COUNT(ProductID)
FROM Products;
```

```sql
SELECT AVG(Price)
FROM Products;
```

```sql
SELECT SUM(Quantity)
FROM OrderDetails;
```

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

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

Here are some examples showing different `LIKE` operators with '%' and '_' wildcards:

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

```sql
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%';
```

```sql
SELECT * FROM Customers
WHERE CustomerName LIKE '%a';
```

```sql
SELECT * FROM Customers
WHERE CustomerName LIKE '%or%';
```

```sql
SELECT * FROM Customers
WHERE CustomerName LIKE 'a__%';
```

```sql
SELECT * FROM Customers
WHERE CustomerName NOT LIKE 'a%';
```

WHERE CustomerName LIKE 'a_%_%'	Finds any values that starts with "a" and are at least 3 characters in length

```sql
SELECT * FROM Customers
WHERE City LIKE 'ber%';
```

```sql
SELECT * FROM Customers
WHERE City LIKE '_ondon';
```

```sql
SELECT * FROM Customers
WHERE City LIKE 'L_n_on';
```

The `IN` operator allows you to specify multiple values in a `WHERE` clause.

The `IN` operator is a shorthand for multiple `OR` conditions.

The following SQL statement selects all customers that are located in "Germany", "France" or "UK":

Example

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

```sql
SELECT * FROM Customers
WHERE Country NOT IN ('Germany', 'France', 'UK');
```

The following SQL statement selects all customers that are from the same countries as the suppliers:

Example

```
SELECT * FROM Customers
WHERE Country IN (SELECT Country FROM Suppliers);
```

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.

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20;
```

```
SELECT * FROM Products
WHERE Price NOT BETWEEN 10 AND 20;
```

The following SQL statement selects all products with a price between 10 and 20. In addition; do not show products with a CategoryID of 1,2, or 3:

Example

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20
AND CategoryID NOT IN (1,2,3);
```

The following SQL statement selects all products with a ProductName between "Carnarvon Tigers" and "Mozzarella di Giovanni":

Example

```
SELECT * FROM Products
WHERE ProductName BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni'
ORDER BY ProductName;
```

```
SELECT * FROM Products
WHERE ProductName NOT BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni'
ORDER BY ProductName;
```

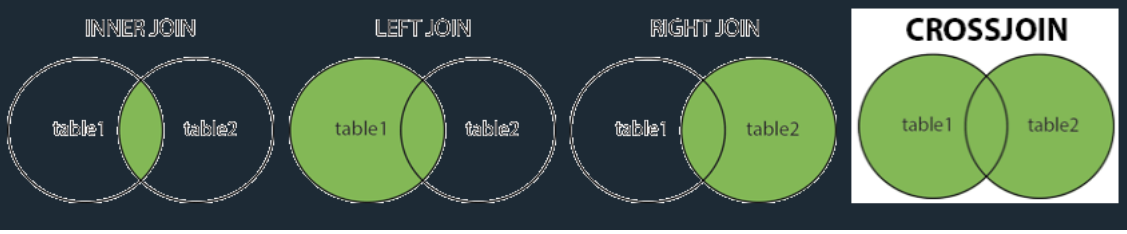
```
SELECT * FROM Orders
WHERE OrderDate BETWEEN '1996-07-01' AND '1996-07-31';
```

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

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

Supported Types of Joins in MySQL

- `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




```
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);
```

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

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

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
CROSS JOIN Orders;
```

If you add a `WHERE` clause (if table1 and table2 has a relationship), the `CROSS JOIN` will produce the same result as the `INNER JOIN` clause:

Example

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

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

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

T1 and *T2* are different table aliases for the same table.

```
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 `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;
```

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;
```

```
SELECT City FROM Customers
UNION
SELECT City FROM Suppliers
ORDER BY City;
```

The following SQL statement returns the cities (duplicate values also) from both the "Customers" and the "Suppliers" table:

Example

```
SELECT City FROM Customers
UNION ALL
SELECT City FROM Suppliers
ORDER BY City;
```

The following SQL statement returns the German cities (only distinct values) from both the "Customers" and the "Suppliers" table:

Example

```
SELECT City, Country FROM Customers
WHERE Country='Germany'
UNION
SELECT City, Country FROM Suppliers
WHERE Country='Germany'
ORDER BY City;
```

```
SELECT 'Customer' AS Type, ContactName, City, Country
FROM Customers
UNION
SELECT 'Supplier', ContactName, City, Country
FROM Suppliers;
```

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);
```

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

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

```
SELECT Shippers.ShipperName, COUNT(Orders.OrderID) AS NumberOfOrders FROM Orders
LEFT JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID
GROUP BY ShipperName;
```

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 condition
ORDER BY column_name(s);
```

The following SQL statement lists the number of customers in each country. Only include countries with more than 5 customers:

Example

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

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM (Orders
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID)
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 10;
```

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM Orders
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
WHERE LastName = 'Davolio' OR LastName = 'Fuller'
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 25;
```

```
CREATE DATABASE databasename;
```

Tip: Make sure you have admin privilege before creating any database. Once a database is created, you can check it in the list of databases with the following SQL command: `SHOW DATABASES`;

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

Syntax

```
DROP DATABASE databasename;
```

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

A copy of an existing table can also be created using `CREATE TABLE`.

The new table gets the same column definitions. All columns or specific columns can be selected.

If you create a new table using an existing table, the new table will be filled with the existing values from the old table.

Syntax

```sql
CREATE TABLE new_table_name AS
    SELECT column1, column2,...
    FROM existing_table_name
    WHERE ....;
```

The following SQL creates a new table called "TestTables" (which is a copy of the "Customers" table):

Example

```sql
CREATE TABLE TestTable AS
SELECT customername, contactname
FROM customers;
```

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

Syntax

```sql
DROP TABLE table_name;
```

The `TRUNCATE TABLE` statement is used to delete the data inside a table, but not the table itself.

Syntax

```
TRUNCATE TABLE table_name;
```

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.

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

```
ALTER TABLE table_name
ADD column_name datatype;
```

The following SQL adds an "Email" column to the "Customers" table:

Example

```
ALTER TABLE Customers
ADD Email varchar(255);
```

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;
```

The following SQL deletes the "Email" column from the "Customers" table:

Example

```
ALTER TABLE Customers
DROP COLUMN Email;
```

To change the data type of a column in a table, use the following syntax:

```
ALTER TABLE table_name
MODIFY COLUMN column_name datatype;
```