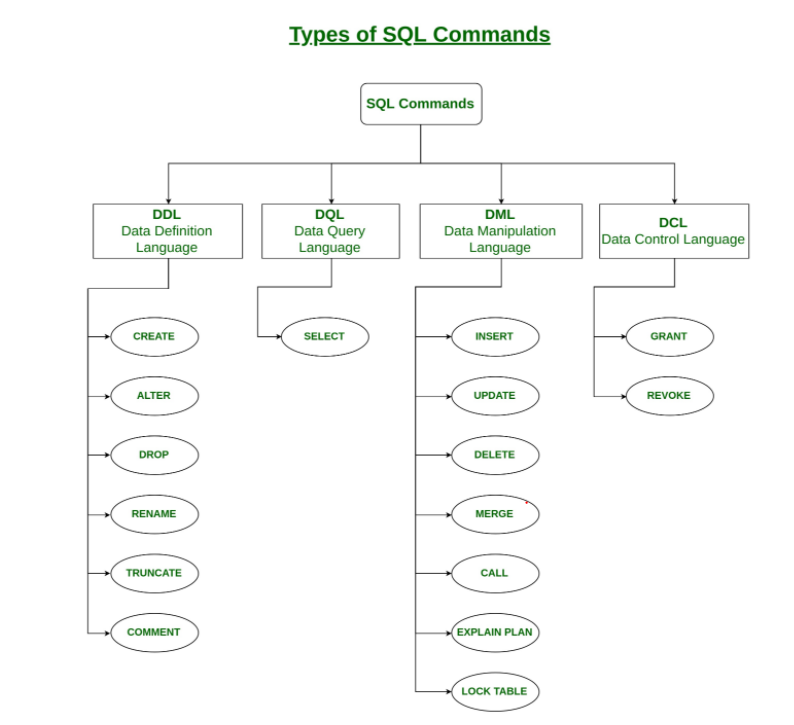
## https://www.mysqltutorial.org/basic-mysql-tutorial.aspx/

## SQL | DDL, DQL, DML, DCL and TCL Commands



1. **DDL(Data Definition Language) :**DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database.

**Examples of DDL commands:**

* + [**CREATE**](https://www.geeksforgeeks.org/sql-create/) – is used to create the database or its objects (like table, index, function, views, store procedure and triggers).
  + [**DROP**](https://www.geeksforgeeks.org/sql-drop-truncate/) – is used to delete objects from the database.
  + [**ALTER**](https://www.geeksforgeeks.org/sql-alter-add-drop-modify/)-is used to alter the structure of the database.
  + [**TRUNCATE**](https://www.geeksforgeeks.org/sql-drop-truncate/)–is used to remove all records from a table, including all spaces allocated for the records are removed.
  + [**COMMENT**](https://www.geeksforgeeks.org/sql-comments/) –is used to add comments to the data dictionary.
  + [**RENAME**](https://www.geeksforgeeks.org/sql-alter-rename/)–is used to rename an object existing in the database.

1. **DQL (Data Query Language) :**

DML statements are used for performing queries on the data within schema objects. The purpose of DQL Command is to get some schema relation based on the query passed to it.

**Example of DQL:**

* + [**SELECT**](https://www.geeksforgeeks.org/sql-select-clause/) – is used to retrieve data from the a database.

1. **DML(Data Manipulation Language) :**The SQL commands that deals with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements.

**Examples of DML:**

* + [**INSERT**](https://www.geeksforgeeks.org/sql-insert-statement/) – is used to insert data into a table.
  + [**UPDATE**](https://www.geeksforgeeks.org/sql-update-statement/) – is used to update existing data within a table.
  + [**DELETE**](https://www.geeksforgeeks.org/sql-delete-statement/) – is used to delete records from a database table.

1. **DCL(Data Control Language) :**DCL includes commands such as GRANT and REVOKE which mainly deals with the rights, permissions and other controls of the database system.

**Examples of DCL commands:**

* + **GRANT**-gives user’s access privileges to database.
  + **REVOKE**-withdraw user’s access privileges given by using the GRANT command.

1. **TCL(transaction Control Language) :**TCL commands deals with the [transaction within the database](https://www.geeksforgeeks.org/sql-transactions/).

**Examples of TCL commands:**

* + **COMMIT**– commits a Transaction.
  + [**ROLLBACK**](https://www.geeksforgeeks.org/sql-transactions/)– rollbacks a transaction in case of any error occurs.
  + **SAVEPOINT**–sets a savepoint within a transaction.
  + **SET TRANSACTION**–specify characteristics for the transaction.

## MySQL SELECT

SELECT lastName FROM employees;

SELECT lastname, firstname, jobtitle FROM employees;

SELECT \* FROM employees;

## MySQL ORDER BY

When you use the [SELECT](https://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) statement to query data from a table, the result set is not sorted. It means that the rows in the result set can be in any order.

To sort the result set, you add the ORDER BY clause

The ASC stands for ascending and the DESC stands for descending. You use ASC to sort the result set in ascending order and DESC to sort the result set in descending order

By default, the ORDER BY clause uses ASC if you don’t explicitly specify any option.

If you want to sort the result set by multiple columns, you specify a comma-separated list of columns in the ORDER BY clause:

ORDER BY

column1,

column2;

It is possible to sort the result by a column in ascending order, and then by another column in descending order:

ORDER BY

column1 ASC,

column2 DESC;

In this case, the ORDER BY clause:

* First, sort the result set by the values in the column1 in ascending order.
* Then, sort the sorted result set by the values in the column2

SELECT contactLastname, contactFirstname FROM customers ORDER BY contactLastname;

SELECT contactLastname, contactFirstname FROM customers ORDER BY contactLastname DESC;

SELECT contactLastname, contactFirstname FROM customers ORDER BY contactLastname DESC, contactFirstname ASC;

SELECT orderNumber, orderlinenumber, quantityOrdered \* priceEach FROM orderdetails ORDER BY quantityOrdered \* priceEach DESC;

To make the query more readable, you can assign the expression in the SELECT clause a [column alias](https://www.mysqltutorial.org/mysql-alias/) and use that column alias in the ORDER BY clause as shown in the following query:

SELECT orderNumber, orderLineNumber, quantityOrdered \* priceEach AS subtotal FROM orderdetails

ORDER BY subtotal DESC;

Suppose that you want to sort the sales orders based on their statuses in the following order:

* In Process
* On Hold
* Canceled
* Resolved
* Disputed
* Shipped

To do this, you can use the FIELD() function to map each order status to a number and sort the result by the result of the FIELD() function:

SELECT orderNumber, status FROM orders

ORDER BY

FIELD(status,

'In Process',

'On Hold',

'Cancelled',

'Resolved',

'Disputed',

'Shipped');

The following expression:

FIELD(status, 'In Process', 'On Hold', 'Cancelled', 'Resolved', 'Disputed', 'Shipped');

returns the index of the status in the list 'In Process', 'On Hold', 'Cancelled', 'Resolved', 'Disputed', 'Shipped'.

For example, if the status is In Process, the function will return 1. If the status is On Hold, the function will return 2, and so on.

## MySQL WHERE

The WHERE clause allows you to specify a search condition for the rows returned by a query

SELECT lastname, firstname, jobtitle FROM employees WHERE jobtitle = 'Sales Rep';

SELECT lastname, firstname, jobtitle, officeCode FROM employees WHERE jobtitle = 'Sales Rep' AND

officeCode = 1;

SELECT lastname, firstname, jobtitle, officeCode FROM employees WHERE jobtitle = 'Sales Rep' OR

officeCode = 1;

SELECT firstName, lastName, officeCode FROM employees WHERE officeCode BETWEEN 1 AND 3

ORDER BY officeCode;

The [LIKE](https://www.mysqltutorial.org/mysql-like/) operator evaluates to TRUE if a value matches a specified pattern. To form a pattern, you use % and \_ wildcards. The % wildcard matches any string of zero or more characters while the \_ wildcard matches any single character.

This query finds employees whose last names end with the string 'son':

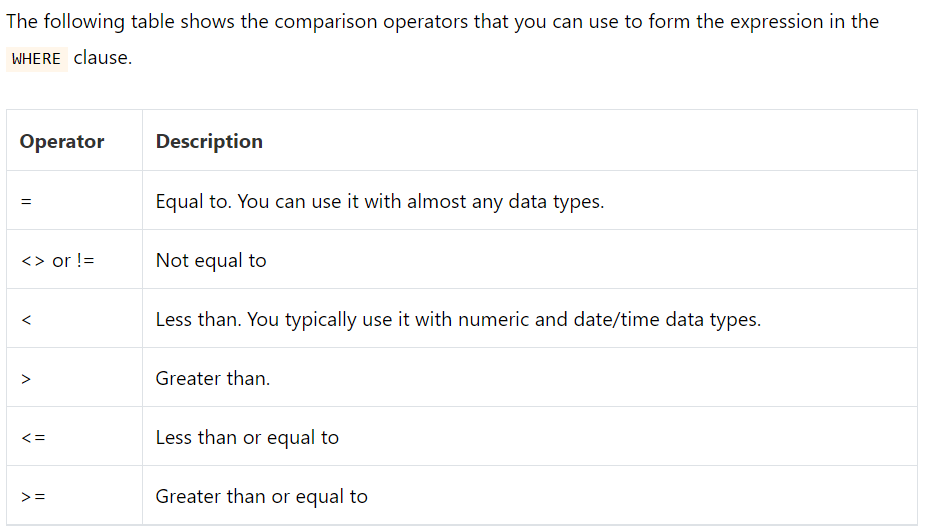
SELECT firstName, lastName FROM employees WHERE lastName LIKE '%son' ORDER BY firstName;

SELECT firstName, lastName FROM employees WHERE firstName LIKE '%Ma\_\_' ORDER BY firstName;

SELECT firstName, lastName, officeCode FROM employees WHERE officeCode IN (1 , 2, 3) ORDER BY

officeCode;

SELECT lastName, firstName, reportsTo FROM employees WHERE reportsTo IS NULL;



SELECT lastname, firstname, jobtitle FROM employees WHERE jobtitle <> 'Sales Rep';

SELECT lastname, firstname, officeCode FROM employees WHERE officecode > 5;

SELECT lastname, firstname, officeCode FROM employees WHERE officecode <= 4;

## MySQL DISTINCT

When querying data from a table, you may get duplicate rows. In order to remove these duplicate rows, you use the DISTINCT clause in the [SELECT](https://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) statement.

SELECT DISTINCT lastname FROM employees ORDER BY lastname;

You can use the DISTINCT clause with more than one column. In this case, MySQL uses the combination of values in these columns to determine the uniqueness of the row in the result set.

SELECT DISTINCT state, city FROM customers WHERE state IS NOT NULL ORDER BY state, city;

If you use the [GROUP BY](https://www.mysqltutorial.org/mysql-group-by.aspx) clause in the SELECT statement without using [aggregate functions](https://www.mysqltutorial.org/mysql-aggregate-functions.aspx), the GROUP BY clause behaves like the DISTINCT clause.

The following statement uses the GROUP BY clause to select the unique states of customers from the customers table.

SELECT state FROM customers GROUP BY state;

Generally speaking, the DISTINCT clause is a special case of the GROUP BY clause. The difference between DISTINCT clause and GROUP BY clause is that the GROUP BY clause [sorts the result set](https://www.mysqltutorial.org/mysql-order-by/) whereas the DISTINCT clause does not.

## **MySQL DISTINCT and aggregate functions**

SELECT COUNT(DISTINCT state) FROM customers WHERE country = 'USA';

SELECT DISTINCT state FROM customers WHERE state IS NOT NULL LIMIT 5;

## MySQL AND Operator

SELECT customername, country, state FROM customers WHERE

country = 'USA' AND state = 'CA';

SELECT customername, country, state, creditlimit FROM customers

WHERE country = 'USA' AND state = 'CA' AND creditlimit > 100000;

## MySQL OR Operator

SELECT customername, country FROM customers WHERE country = 'USA' OR

country = 'France';

SELECT customername, country, creditLimit FROM customers WHERE

(country = 'USA' OR country = 'France') AND creditlimit > 100000;

## MySQL IN

SELECT officeCode, city, phone, country FROM offices WHERE country IN ('USA' , 'France');

SELECT officeCode, city, phone FROM offices WHERE country NOT IN ('USA' , 'France');

SELECT orderNumber, customerNumber, status, shippedDate FROM orders

WHERE orderNumber IN

(

SELECT orderNumber FROM orderDetails GROUP BY orderNumber

HAVING SUM(quantityOrdered \* priceEach) > 60000

);

## MySQL BETWEEN

SELECT productCode, productName, buyPrice FROM products

WHERE buyPrice BETWEEN 90 AND 100;

SELECT productCode, productName, buyPrice FROM products WHERE

buyPrice >= 90 AND buyPrice <= 100;

SELECT productCode, productName, buyPrice FROM products WHERE

buyPrice NOT BETWEEN 20 AND 100;

When you use the BETWEEN operator with date values, to get the best result, you should use the [type cast](https://www.mysqltutorial.org/mysql-cast/) to explicitly convert the type of column or expression to the [DATE type](https://www.mysqltutorial.org/mysql-date/).

The following example returns the orders which have the required dates between 01/01/2003 to 01/31/2003:

SELECT orderNumber, requiredDate, status FROM orders WHERE

requireddate BETWEEN CAST('2003-01-01' AS DATE) AND

CAST('2003-01-31' AS DATE);

## MySQL LIKE

The LIKE operator is a logical operator that tests whether a string contains a specified pattern or not. Here is the syntax of the LIKE operator:

find employees whose first names start with a

SELECT employeeNumber, lastName, firstName FROM employees

WHERE firstName LIKE 'a%';

find employees whose last names end with on e.g., Patterson, Thompson:

SELECT employeeNumber, lastName, firstName FROM employees

WHERE lastName LIKE '%on';

to find all employees whose last names contain on , you use the following query with the pattern %on%

SELECT employeeNumber, lastName, firstName FROM employees

WHERE lastname LIKE '%on%';

SELECT employeeNumber, lastName, firstName FROM employees

WHERE firstname LIKE 'T\_m';

SELECT employeeNumber, lastName, firstName FROM employees WHERE

lastName NOT LIKE 'B%';

Sometimes the pattern, which you want to match, contains wildcard character e.g., 10%, \_20, etc. In this case, you can use the ESCAPE clause to specify the escape character so that MySQL will interpret the wildcard character as a literal character. If you don’t specify the escape character explicitly, the backslash character \ is the default escape character.

For example, if you want to find products whose product codes contain the string \_20 , you can use the pattern %\\_20% as shown in the following query:

SELECT productCode, productName FROM products

WHERE productCode LIKE '%\\_20%';

Or you can specify a different escape character e.g., $ by using the ESCAPE clause:

SELECT productCode, productName FROM products

WHERE productCode LIKE '%$\_20%' ESCAPE '$';

## MySQL LIMIT

The LIMIT clause is used in the [SELECT](https://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) statement to constrain the number of rows to return

he following illustrates the LIMIT clause syntax with two arguments:

SELECT

select\_list

FROM

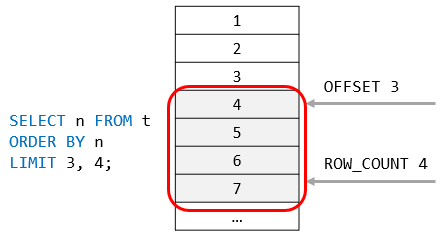
table\_name

LIMIT [offset,] row\_count;

In this syntax:

* The offset specifies the offset of the first row to return. The offset of the first row is 0, not 1.
* The row\_count specifies the maximum number of rows to return.

The following picture illustrates the LIMIT clause:



This statement uses the LIMIT clause to get the top five customers who have the highest credit:

SELECT customerNumber, customerName, creditLimit FROM customers

ORDER BY creditLimit DESC LIMIT 5;

this example uses the LIMIT clause to find 5 customers who have the lowest credits

SELECT customerNumber, customerName, creditLimit FROM customers

ORDER BY creditLimit LIMIT 5;

### Using MySQL LIMIT for pagination

This query uses the LIMIT clause to get rows of page 1 which contains the first 10 customers sorted by the customer name:

SELECT customerNumber, customerName FROM customers ORDER BY customerName LIMIT 10;

This query uses the LIMIT clause to get the rows of the second page that include row 11 – 20:

SELECT customerNumber, customerName FROM customers ORDER BY customerName LIMIT 10, 10;

### Using MySQL LIMIT to get the nth highest or lowest value

SELECT select\_list FROM table\_name ORDER BY sort\_expression

LIMIT n-1, 1;

The clause LIMIT n-1, 1 returns 1 row starting at the row n.

For example, the following finds the customer who has the second-highest credit:

SELECT customerName, creditLimit FROM customers ORDER BY

creditLimit DESC LIMIT 1,1;

## MySQL IS NULL

SELECT customerName, country, salesrepemployeenumber FROM customers

WHERE salesrepemployeenumber IS NULL ORDER BY customerName;

SELECT customerName, country, salesrepemployeenumber FROM customers

WHERE salesrepemployeenumber IS NOT NULL ORDER BY customerName;

## MySQL Alias

Sometimes, column names are so technical that make the query’s output very difficult to understand. To give a column a descriptive name, you can use a column alias.

SELECT CONCAT\_WS(', ', lastName, firstname) AS `Full name`

FROM employees;

SELECT orderNumber `Order no.`, SUM(priceEach \* quantityOrdered) total FROM orderDetails GROUP BY `Order no.` HAVING total > 60000;

This query shows how to assign the employees table alias as e:

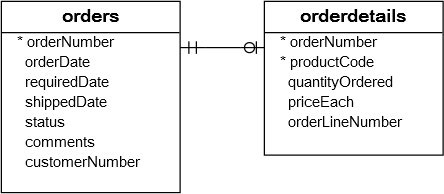
SELECT \* FROM employees e;

SELECT e.firstName, e.lastName FROM employees e ORDER BY e.firstName;

## MySQL Join

A relational database consists of multiple related tables linking together using common columns which are known as [foreign key](http://www.mysqltutorial.org/mysql-foreign-key/) columns. Because of this, data in each table is incomplete from the business perspective.

For example, in the [sample database](http://www.mysqltutorial.org/mysql-sample-database.aspx), we have the orders and  orderdetails tables that are linked using the orderNumber column:



To get complete orders’ information, you need to query data from both orders and  orderdetails tables.

That’s why joins come into the play.

MySQL supports the following types of joins:

1. [Inner join](https://www.mysqltutorial.org/mysql-inner-join.aspx)
2. [Left join](https://www.mysqltutorial.org/mysql-left-join.aspx)
3. [Right join](https://www.mysqltutorial.org/mysql-right-join/)
4. [Cross join](https://www.mysqltutorial.org/mysql-cross-join/)

## MySQL INNER JOIN clause

The [inner join](https://www.mysqltutorial.org/mysql-inner-join.aspx) clause joins two tables based on a condition which is known as a join predicate.

The inner join clause compares each row from the first table with every row from the second table. If values in both rows cause the join condition evaluates to true, the inner join clause creates a new row whose column contains all columns of the two rows from both tables and include this new row in the final result set. In other words, the inner join clause includes only rows whose values match.

The following shows the basic syntax of the inner join clause that joins two tables table\_1 and table\_2:

SELECT column\_list

FROM table\_1

INNER JOIN table\_2 ON join\_condition;

If the join condition uses the equal operator (=) and the column names in both tables used for matching are the same, you can use the USING clause instead:

SELECT column\_list

FROM table\_1

INNER JOIN table\_2 USING (column\_name);

The following statement finds members who are also the committee members:

SELECT

m.member\_id,

m.name member,

c.committee\_id,

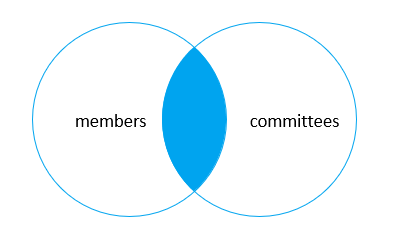
c.name committee

FROM

members m

INNER JOIN committees c

ON c.name = m.name;



## MySQL LEFT JOIN clause

Similar to an inner join, a [left join](https://www.mysqltutorial.org/mysql-left-join.aspx) also requires a join-predicate. When joining two tables using a left join, the concepts of left and right tables are introduced.

The left join selects data starting from the left table. For each row in the left table, the left join compares with every row in the right table. If the values in the two rows cause the join condition evaluates to true, the left join creates a new row whose columns contain all columns of the rows in both tables and includes this row in the result set.

If the values in the two rows are not matched, the left join clause still creates a new row whose columns contain columns of the row in the left table and NULL for columns of the row in the right table.

In other words, the left join selects all data from the left table whether there are matching rows exist in the right table or not. In case there is no matching rows from the right table found, NULLs are used for columns of the row from the right table in the final result set.

SELECT column\_list

FROM table\_1

LEFT JOIN table\_2 ON join\_condition;

SELECT

m.member\_id,

m.name member,

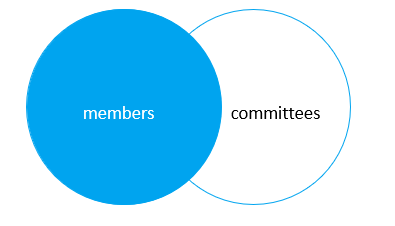
c.committee\_id,

c.name committee

FROM

members m

LEFT JOIN committees c USING(name);



To find members who are not the committee members, you add a [WHERE](https://www.mysqltutorial.org/mysql-where/) clause and [IS NULL](https://www.mysqltutorial.org/mysql-is-null/) operator as follows:

SELECT

m.member\_id,

m.name member,

c.committee\_id,

c.name committee

FROM

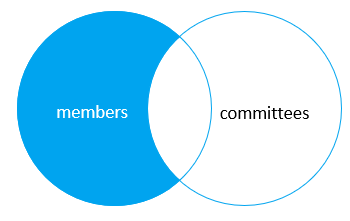
members m

LEFT JOIN committees c USING(name)

WHERE c.committee\_id IS NULL;

Generally, this query pattern can find rows in the left table that does not have corresponding rows in the right table.

This Venn diagram illustrates how to use the left join to select rows that only exist in the left table:



## MySQL RIGHT JOIN clause

The [right join](https://www.mysqltutorial.org/mysql-right-join/) clause is similar to the left join clause except that the treatment of tables is reversed. The right join starts selecting data from the right table instead of the left table.

The right join clause selects all rows from the right table and matches rows in the left table. If a row from the right table does not have matching rows from the left table, the column of the left table will have NULL in the final result set.

Here is the syntax of the right join:

SELECT column\_list

FROM table\_1

RIGHT JOIN table\_2 ON join\_condition;

Similar to the left join clause, the right clause also supports the USING syntax:

SELECT column\_list

FROM table\_1

RIGHT JOIN table\_2 USING (column\_name);

To find rows in the right table that does not have corresponding rows in the left table, you also use a [WHERE](https://www.mysqltutorial.org/mysql-where/) clause with the [IS NULL](https://www.mysqltutorial.org/mysql-is-null/) operator:

SELECT column\_list

FROM table\_1

RIGHT JOIN table\_2 USING (column\_name)

WHERE column\_table\_1 IS NULL;

This statement uses the right join to join the members and committees tables:

SELECT

m.member\_id,

m.name member,

c.committee\_id,

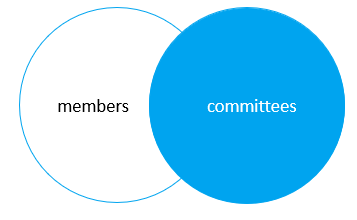
c.name committee

FROM

members m

RIGHT JOIN committees c on c.name = m.name;

This Venn diagram illustrates the right join:



The following statement uses the right join clause with the USING syntax:

SELECT

m.member\_id,

m.name member,

c.committee\_id,

c.name committee

FROM

members m

RIGHT JOIN committees c USING(name);

To find the committee members who are not in the members table, you use this query:

SELECT

m.member\_id,

m.name member,

c.committee\_id,

c.name committee

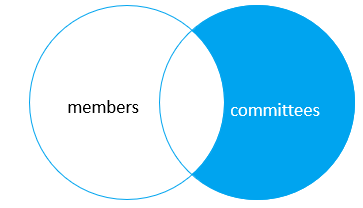
FROM

members m

RIGHT JOIN committees c USING(name)

WHERE m.member\_id IS NULL;

This Venn diagram illustrates how to use the right join to select data that exists only in the right table:



Unlike the [inner join](https://www.mysqltutorial.org/mysql-inner-join.aspx), [left join](https://www.mysqltutorial.org/mysql-left-join.aspx), and [right join](https://www.mysqltutorial.org/mysql-right-join/), the [cross join](https://www.mysqltutorial.org/mysql-cross-join/) clause does not have a join condition.

The cross join makes a Cartesian product of rows from the joined tables. The cross join combines each row from the first table with every row from the right table to make the result set.

Suppose the first table has n rows and the second table has m rows. The cross join that joins the first with the second table will return nxm rows.

The following shows the basic syntax of the cross join clause:

SELECT select\_list

FROM table\_1

CROSS JOIN table\_2;

This example uses the cross join clause to join the members with the committees tables:

SELECT

m.member\_id,

m.name member,

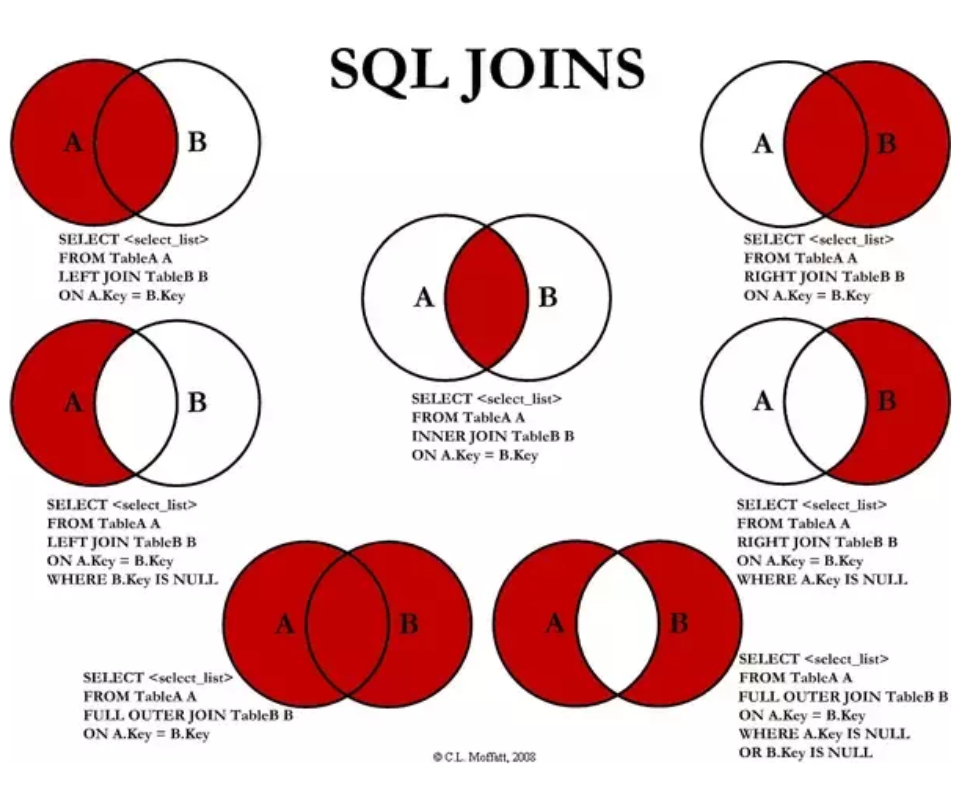
c.committee\_id,

c.name committee

FROM

members m

CROSS JOIN committees c;



## MySQL GROUP BY

SELECT status FROM orders GROUP BY status;

SELECT status, COUNT(\*) FROM orders GROUP BY status;

In addition to columns, you can group rows by expressions. The following query gets the total sales for each year.

SELECT YEAR(orderDate) AS year,SUM(quantityOrdered \* priceEach) AS total FROM orders INNER JOIN orderdetails USING (orderNumber)WHERE status = 'Shipped' GROUP BY YEAR(orderDate);

SELECT YEAR(orderDate) AS year, SUM(quantityOrdered \* priceEach) AS total FROM orders INNER JOIN orderdetails USING (orderNumber)WHERE status = 'Shipped' GROUP BY year HAVING year > 2003;

## MySQL HAVING

The  HAVING clause is used in the [SELECT](https://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) statement to specify filter conditions for a group of rows or aggregates.

The HAVING clause is often used with the [GROUP BY](https://www.mysqltutorial.org/mysql-group-by.aspx) clause to filter groups based on a specified condition. If the GROUP BY clause is omitted, the HAVING clause behaves like the [WHERE](https://www.mysqltutorial.org/mysql-where/) clause.

SELECT ordernumber, SUM(quantityOrdered) AS itemsCount, SUM(priceeach\*quantityOrdered) AS total FROM orderdetails GROUP BY ordernumber HAVING total > 1000;

SELECT ordernumber,SUM(quantityOrdered) AS itemsCount, SUM(priceeach\*quantityOrdered) AS total FROM orderdetails GROUP BY ordernumber HAVING total > 1000 AND itemsCount > 600;

## MySQL ROLLUP

<https://www.mysqltutorial.org/mysql-rollup/>

## MySQL Subquery

SELECT lastName, firstName FROM employees WHERE officeCode IN (SELECT officeCode FROM offices WHERE country = 'USA');

the following query returns the customer who has the maximum payment.

SELECT customerNumber, checkNumber, amount FROM payments WHERE amount = (SELECT MAX(amount) FROM payments);

you can find customers whose payments are greater than the average payment using a subquery:

SELECT customerNumber, checkNumber, amount FROM payments WHERE amount > (SELECT

AVG(amount) FROM payments);

you can use a subquery with NOT IN operator to find the customers who have not placed any orders as follows:

SELECT customerName FROM customers WHERE customerNumber NOT IN (SELECT DISTINCT customerNumber FROM orders);

SELECT MAX(items), MIN(items), FLOOR(AVG(items)) FROM (SELECT orderNumber, COUNT(orderNumber) AS items FROM orderdetails GROUP BY orderNumber) AS lineitems;

In the following query, we select products whose buy prices are greater than the average buy price of all products in eachproduct line.

SELECT productname, buyprice FROM products p1 WHERE buyprice > (SELECT AVG(buyprice) FROM products WHERE productline = p1.productline)

You can use the query above as a correlated subquery to find customers who placed at least one sales order with the total value greater than 60K by using the EXISTS operator:

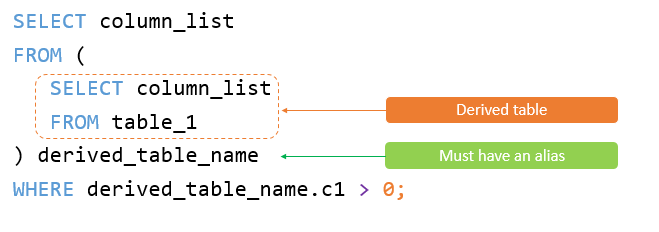
SELECT customerNumber, customerName FROM customers WHERE EXISTS( SELECT orderNumber, SUM(priceEach \* quantityOrdered) FROM orderdetails INNER JOIN orders USING (orderNumber) WHERE customerNumber = customers.customerNumber GROUP BY orderNumber HAVING SUM(priceEach \* quantityOrdered) > 60000);

## An Essential Guide to MySQL Derived Table

A derived table is a virtual table returned from a [SELECT](https://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) statement. A derived table is similar to a [temporary table](https://www.mysqltutorial.org/mysql-temporary-table/), but using a derived table in the SELECT statement is much simpler than a temporary table because it does not require steps of creating the temporary table.

The term derived table and [subquery](https://www.mysqltutorial.org/mysql-subquery/) is often used interchangeably. When a stand-alone subquery is used in the FROM clause of a SELECT statement, it is called a derived table.

The following illustrates a query that uses a derived table:



Unlike a subquery, a derived table must have an [alias](https://www.mysqltutorial.org/mysql-alias/) so that you can reference its name later in the query. If a derived table does not have an alias, MySQL will issue the following error

## A Definitive Guide To MySQL Recursive CTE

<https://www.mysqltutorial.org/mysql-recursive-cte/>

## MySQL EXISTS

The EXISTS operator is a Boolean operator that returns either true or false. The EXISTS operator is often used to test for the existence of rows returned by the [subquery](https://www.mysqltutorial.org/mysql-subquery/).

The following statement uses the EXISTS operator to find the customer who has at least one order:

**SELECT** customerNumber, customerName **FROM** customers **WHERE** **EXISTS**(

**SELECT** 1 **FROM** orders **WHERE** orders.customernumber = customers.customernumber);

The following example uses the NOT EXISTS operator to find customers who do not have any orders:

**SELECT** customerNumber, customerName **FROM** customers **WHERE** **NOT** **EXISTS**( **SELECT** 1 **FROM** orders **WHERE** orders.customernumber = customers.customernumber);