# **MySQL Derived Tables**

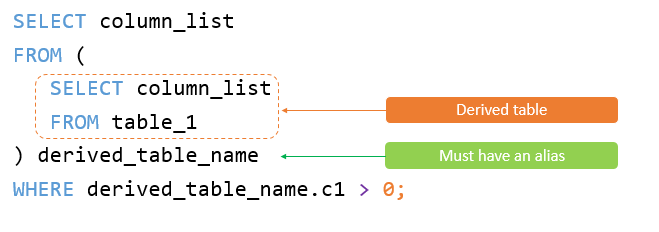
**Summary**: in this tutorial, you will learn about MySQL derived tables and how to use them to simplify complex queries.

## **Introduction to MySQL derived tables**

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

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

The following illustrates a query that uses a derived table:

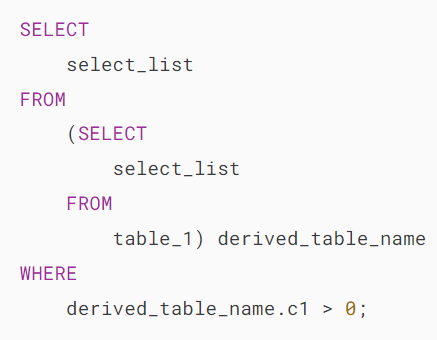


Note that a stand-alone subquery is a subquery that can execute independently of the outer query.

Unlike a subquery, a derived table must have an [alias](https://www.mysqltutorial.org/mysql-basics/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:

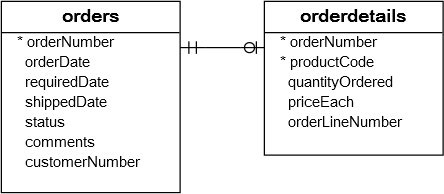


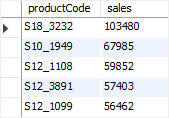
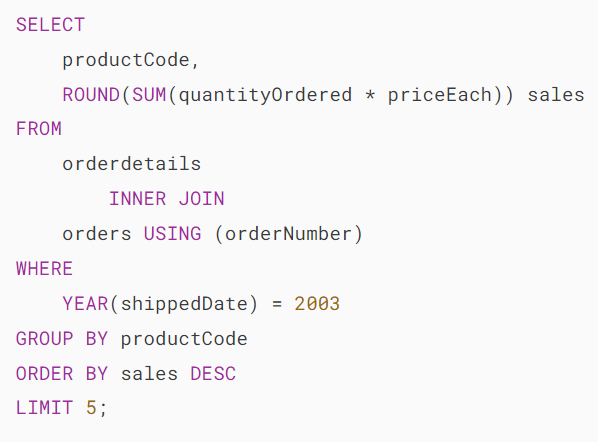
The following illustrates the syntax of a query that uses a derived table:



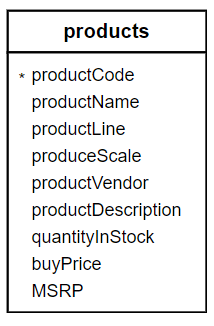
## **Basic MySQL Derived Table example**

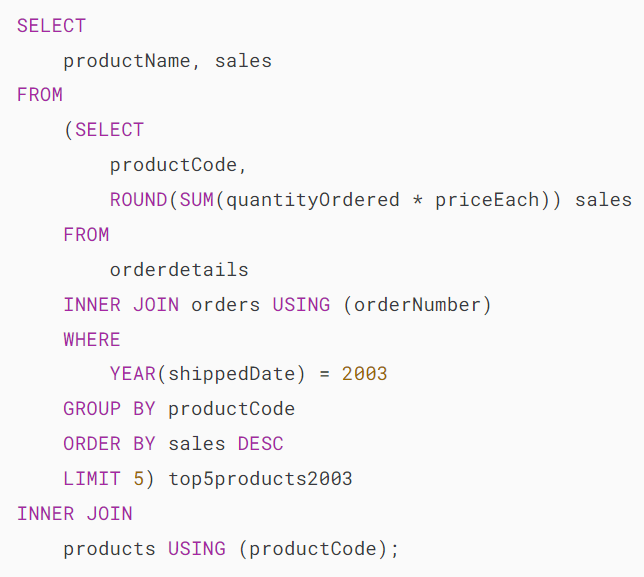
The following query gets the top five products by sales revenue in 2003 from the orders and orderdetails tables in the [sample database](https://www.mysqltutorial.org/getting-started-with-mysql/mysql-sample-database/):

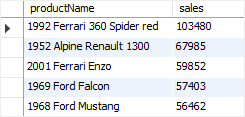




You can use the result of this query as a derived table and join it with the products table as follows:







In this example:

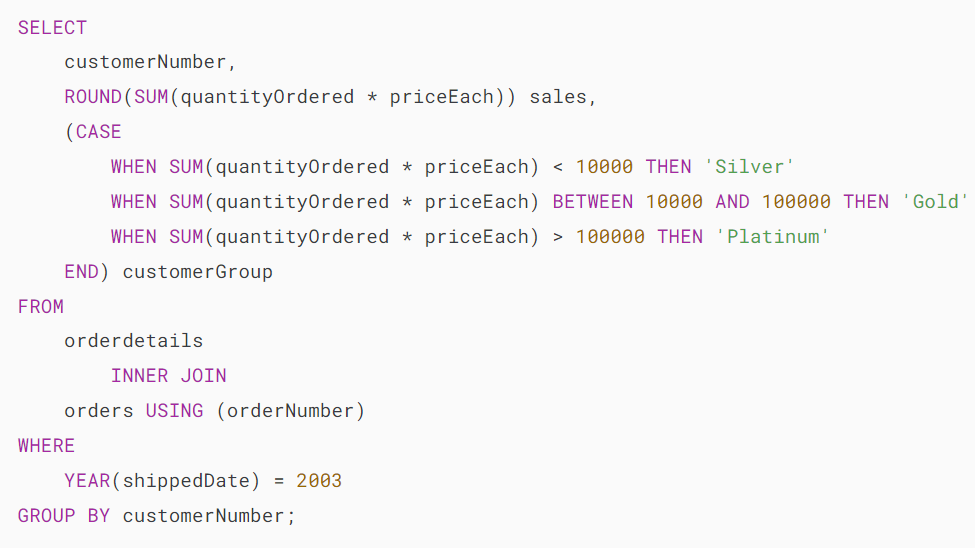
1. First, the subquery is executed to create a result set or derived table.
2. Then, the outer query is executed that joins the top5product2003 derived table with the products table using the productCode column.

## **A more complex MySQL Derived Table example**

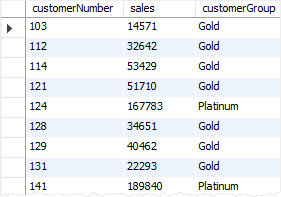
Suppose you have to classify the customers who bought products in 2003 into 3 groups: platinum, gold, and silver. And you need to know the number of customers in each group with the following conditions:

* Platinum customers who have orders with a volume greater than 100K.
* Gold customers who have orders with a volume between 10K and 100K.
* Silver customers who have orders with a volume of less than 10K.

To form this query, you first need to put each customer into the respective group using [CASE](https://www.mysqltutorial.org/mysql-control-flow-functions/mysql-case-function/) expression and [GROUP BY](https://www.mysqltutorial.org/mysql-basics/mysql-group-by/) clause as follows:



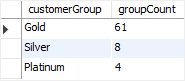
The following is the output of the query:



Then, you can use this query as the derived table and perform grouping as follows:



The query returns the customer groups and the number of customers in each.



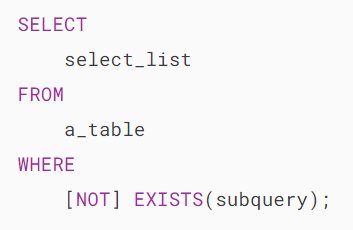
# **MySQL EXISTS**

**Summary**: in this tutorial, you will learn how to use the MySQL EXISTS operator and when to use it to improve the performance of the queries.

## **Introduction to MySQL EXISTS operator**

The EXISTS operator is a [boolean](https://www.mysqltutorial.org/mysql-basics/mysql-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-basics/mysql-subquery/).

The following illustrates the basic syntax of the EXISTS operator:



If the subquery returns at least one row, the EXISTS operator returns true, otherwise, it returns false.

In addition, the EXISTS operator terminates further processing immediately once it finds a matching row, which can help improve the performance of the query.

The NOT operator negates the EXISTS operator. In other words, the NOT EXISTS returns true if the subquery returns no row, otherwise it returns false.

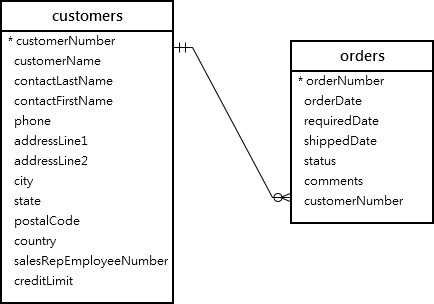
Note that you can use SELECT \*, SELECT column, SELECT a\_constant, or anything in the subquery. The results are the same because MySQL ignores the select list that appeared in the [SELECT](https://www.mysqltutorial.org/mysql-basics/mysql-select-from/) clause.

## **MySQL EXISTS operator examples**

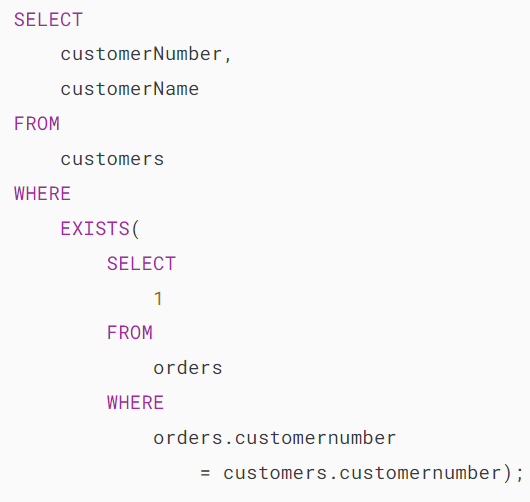
Let’s take some examples of using the EXISTS operator to understand how it works.

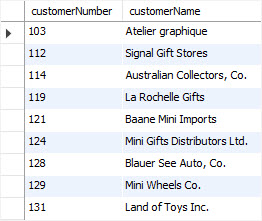
### **MySQL SELECT EXISTS examples**

Consider the following customers and orders tables in the [sample database](https://www.mysqltutorial.org/getting-started-with-mysql/mysql-sample-database/).



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

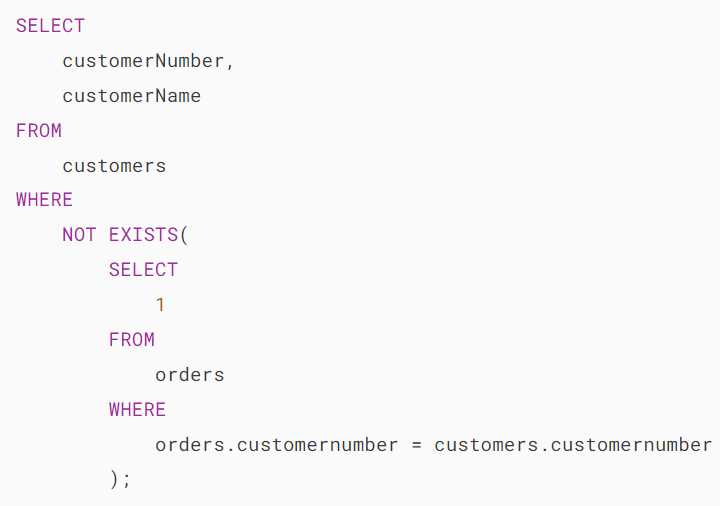


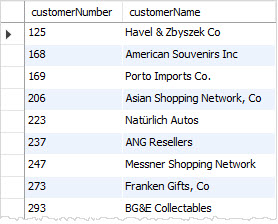


In this example, for each row in the customers table, the query checks the customerNumber in the orders table. If the customerNumber, which appears in the customers table, exists in the orders table, the subquery returns the first matching row.

As a result, the EXISTS operator returns true and stops examining the orders table. Otherwise, the subquery returns no row, and the EXISTS operator returns false.

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



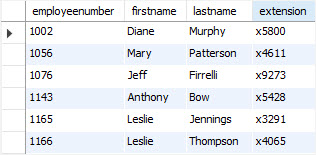


### **MySQL UPDATE EXISTS examples**

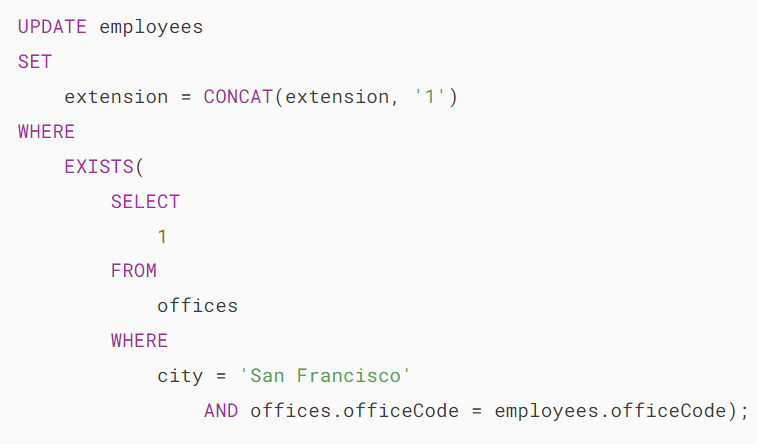
Suppose that you have to update the phone extensions of the employees who work at the office in San Francisco.

The following statement finds employees who work at the office in San Franciso:





This example adds the number 1 to the phone extension of employees who work at the office in San Francisco:



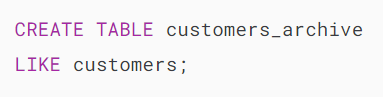
How it works.

* First, the [EXISTS](https://www.mysqltutorial.org/mysql-basicshttps:/www.mysqltutorial.org/mysql-basics/mysql-exists/) operator in the [WHERE](https://www.mysqltutorial.org/mysql-basics/mysql-where/) clause gets only employees who work at the office in San Fransisco.
* Second, the [CONCAT()](https://www.mysqltutorial.org/mysql-string-functions/mysql-concat/) function concatenate the phone extension with the number 1.

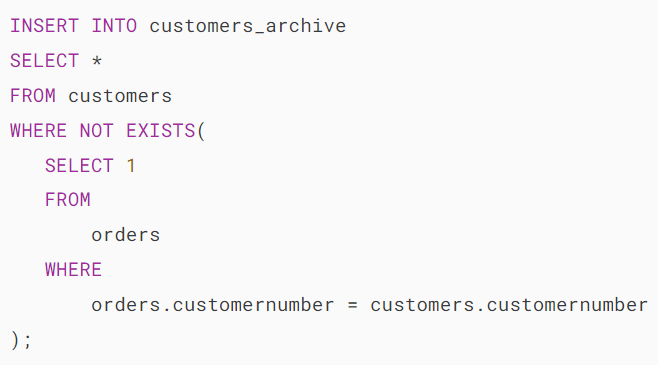
### **MySQL INSERT EXISTS example**

Suppose that you want to archive customers who don’t have any sales orders in a separate table. To do this, you use these steps:

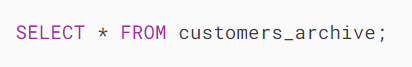
First, [create a new table](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) for archiving the customers by [copying](https://www.mysqltutorial.org/mysql-basics/mysql-copy-table/) the structure from the customers table:



Second, insert customers who do not have any sales orders into the customers\_archive table using the following [INSERT](https://www.mysqltutorial.org/mysql-basics/mysql-insert/) statement.



Third, [query data](https://www.mysqltutorial.org/mysql-basics/mysql-select-from/) from the customers\_archive table to verify the insert operation.



### **MySQL DELETE EXISTS example**

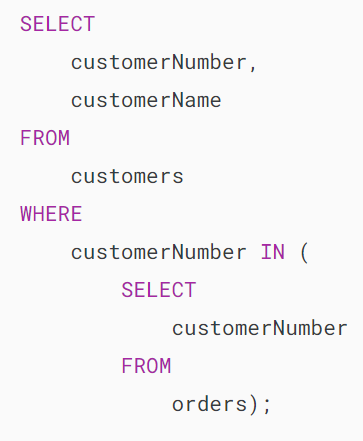
One final task in archiving the customer data is to delete the customers that exist in the customers\_archive table from the customers table.

To do this, you use the EXISTS operator in WHERE clause of the [DELETE](https://www.mysqltutorial.org/mysql-basics/mysql-delete/) statement as follows:

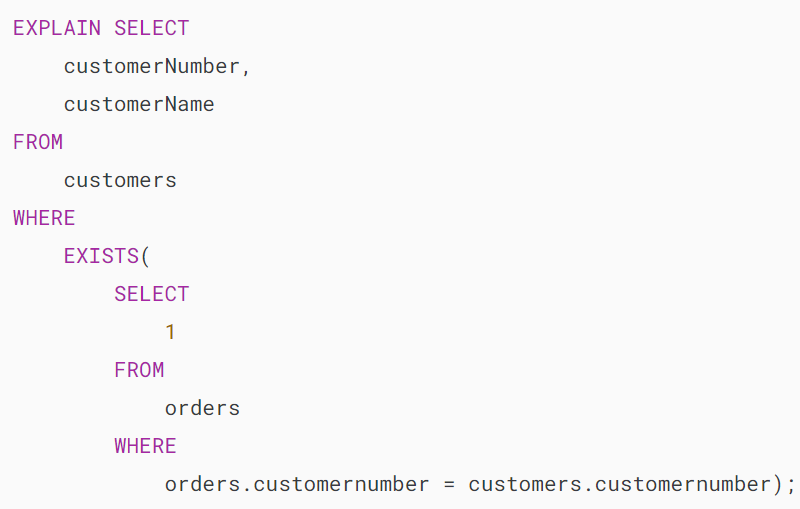


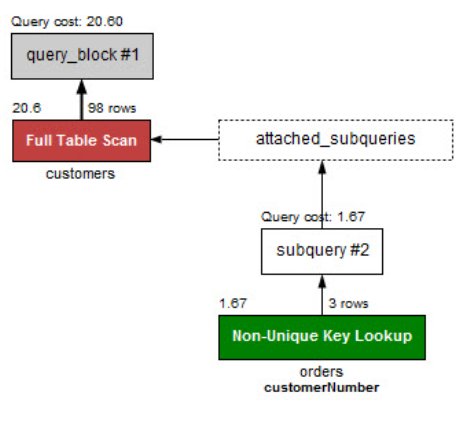
## **MySQL EXISTS operator vs. IN operator**

To find the customer who has placed at least one order, you can use the [IN](https://www.mysqltutorial.org/mysql-basics/mysql-in/) operator as shown in the following query:

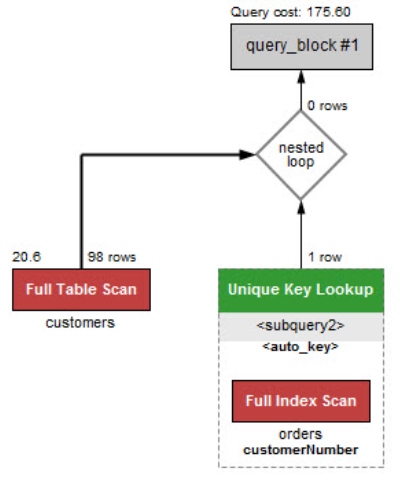
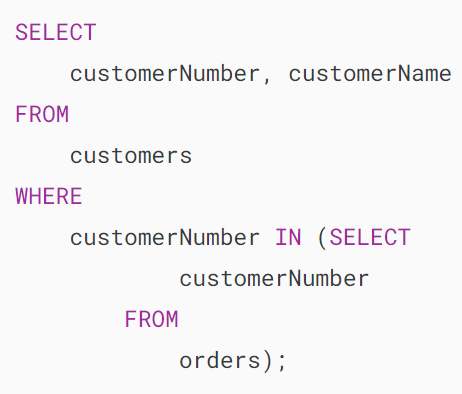


Let’s compare the query that uses the IN operator with the one that uses the EXISTS operator by using the EXPLAIN statement.





Now, check the performance of the query that uses the [IN](https://www.mysqltutorial.org/mysql-basics/mysql-in/) operator.



The query that uses the EXISTS operator is much faster than the one that uses the IN operator.

The reason is that the EXISTS operator works based on the “at least found” principle. The EXISTS stops scanning the table when a matching row is found.

On the other hand, when the IN operator is combined with a subquery, MySQL must process the subquery first and then use the result of the subquery to process the whole query.

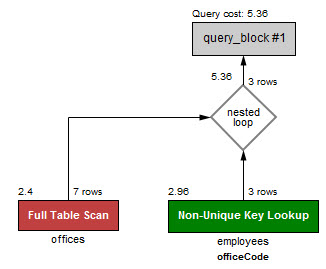
The general rule of thumb is that if the subquery contains a large volume of data, the EXISTS operator provides better performance.

However, the query that uses the IN operator will perform faster if the result set returned from the subquery is very small.

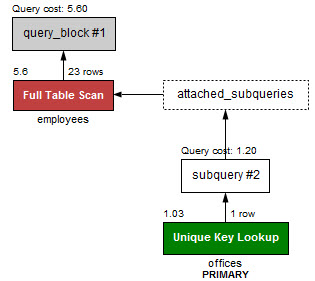
For example, the following statement uses the IN operator to select all employees who work at the office in San Francisco.



Let’s check the performance of the query.



It is a little bit faster than the query that uses the EXISTS operator that we mentioned in the first example. See the performance of the query that uses the EXIST operator below:

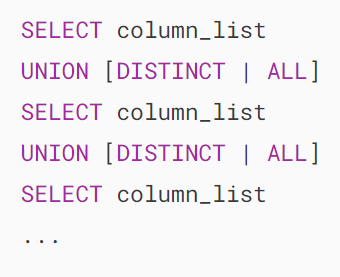


# **MySQL UNION**

**Summary**: in this tutorial, you will learn how to use MySQL UNION operator to combine two or more result sets from multiple SELECT statements into a single result set.

## **MySQL UNION operator**

MySQL UNION operator allows you to combine two or more result sets of queries into a single result set. The following illustrates the syntax of the UNION operator:

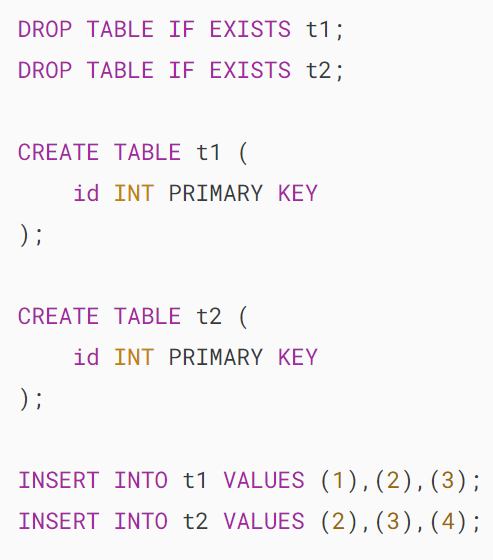


To combine result set of two or more queries using the UNION operator, these are the basic rules that you must follow:

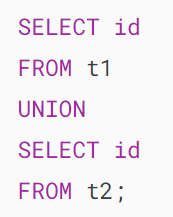
* First, the number and the orders of columns that appear in all [SELECT](https://www.mysqltutorial.org/mysql-basics/mysql-select-from/) statements must be the same.
* Second, the [data types](https://www.mysqltutorial.org/mysql-basics/mysql-data-types/) of columns must be the same or compatible.

By default, the UNION operator removes [duplicate rows](https://www.mysqltutorial.org/mysql-basics/mysql-find-duplicate-values/) even if you don’t specify the [DISTINCT](https://www.mysqltutorial.org/mysql-basics/mysql-distinct/) operator explicitly.

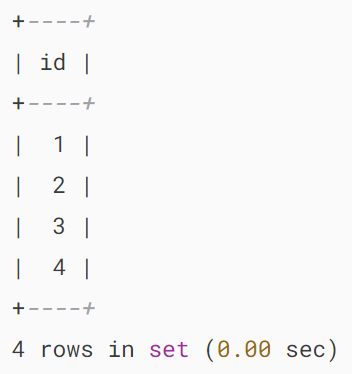
Let’s see the following sample tables: t1 and t2:



The following statement combines result sets returned from t1 and t2 tables:

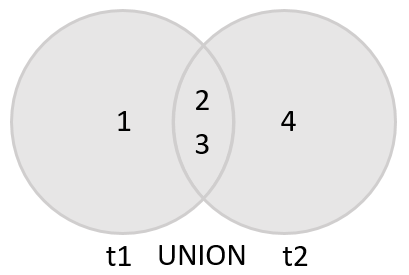


The final result set contains the distinct values from separate result sets returned by the queries:

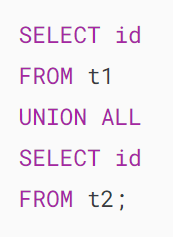


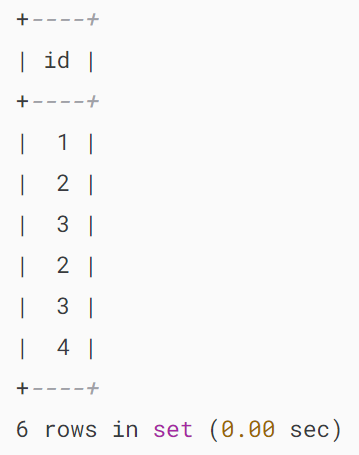
Because the rows with value 2 and 3 are duplicates, the UNION removed them and kept only unique values.

The following Venn diagram illustrates the union of two result sets that come from t1 and t2 tables:



If you use the UNION ALL explicitly, the duplicate rows, if available, remain in the result. Because UNION ALL does not need to handle duplicates, it performs faster than UNION DISTINCT .

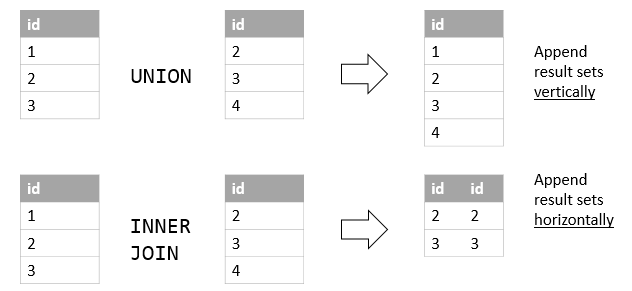




As you can see, the duplicates appear in the combined result set because of the UNION ALL operation.

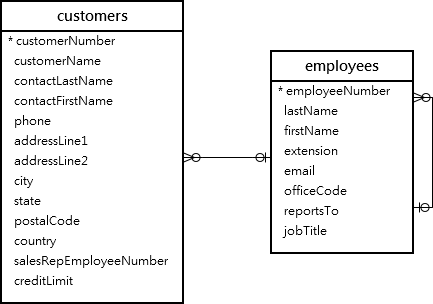
## **UNION vs. JOIN**

A JOIN combines result sets horizontally, a UNION appends result set vertically. The following picture illustrates the difference between UNION and JOIN:

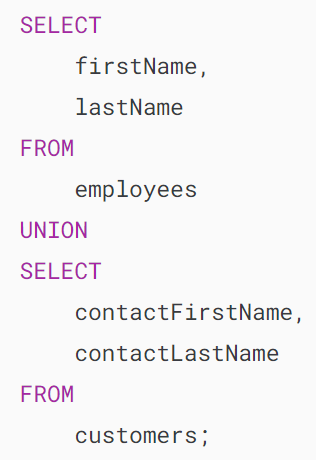


## **MySQL UNION and column alias examples**

We’ll use the customers and employees tables in the [sample database](https://www.mysqltutorial.org/getting-started-with-mysql/mysql-sample-database/) for the demonstration:



Suppose that you want to combine the first name and last name of employees and customers into a single result set, you can use the UNION operator as follows:



As you can see from the output, the MySQL UNION uses the column names of the first SELECT statement for the column headings of the output.

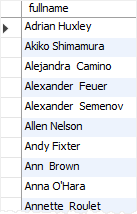
If you want to use other column headings, you need to use [column aliases](https://www.mysqltutorial.org/mysql-basics/mysql-alias/) explicitly in the first SELECT statement as shown in the following example:

****

This example uses the column heading of the first query for the output. It uses the CONCAT() function to concatenate first name, space, and last name into a full name.

## **MySQL UNION and ORDER BY**

If you want to sort the result set of a union, you use an [ORDER BY](https://www.mysqltutorial.org/mysql-basics/mysql-order-by/) clause in the last SELECT statement as shown in the following example:



Notice that if you place the ORDER BY clause in each SELECT statement, it will not affect the order of the rows in the final result set.

To differentiate between employees and customers, you can add a column as shown in the following query:



MySQL also provides you with an alternative option to sort a result set based on column position using ORDER BY clause as follows:



However, it is not a good practice to sort the result set by column position.

# **MySQL EXCEPT**

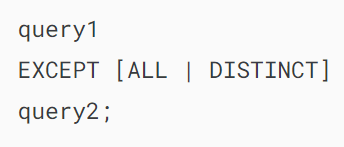
**Summary**: in this tutorial, you will learn how to use the MySQL EXCEPT operator to find the set difference between two sets of data.

Note that the EXCEPT operator is supported in MySQL starting from version 8.0.31. If you use a lower version, you can [emulate the EXCEPT (or MINUS) operator](https://www.mysqltutorial.org/mysql-basics/mysql-minus/).

## **Introduction to the MySQL EXCEPT operator**

The MySQL EXCEPT operator allows you to retrieve rows from one query that do not appear in another query.

Here’s the basic syntax of the MySQL EXCEPT operator:

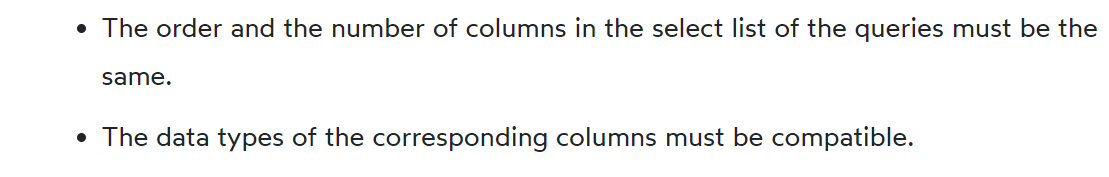


In this syntax, the EXCEPT will compare the result of query1 with the result set of query2 and return the rows of the result set of query1 that do not appear in the result set of query2.

By default, the EXCEPT operator uses the DISTINCT option if you omit it. The EXCEPT DISTINCT removes duplicate rows in the result set.

If you want to retain the duplicate rows, you need to specify the ALL option explicitly.

To use the EXCEPT operator, the query1 and query2 need to follow these rules:



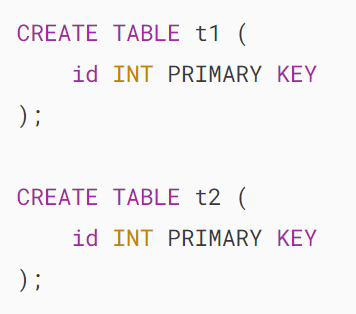
The EXCEPT operator returns a query set with column names derived from the column names of the first query (query1).

## **MySQL EXCEPT operator examples**

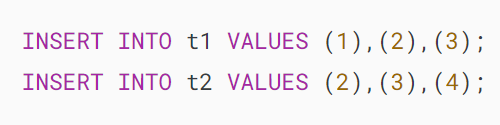
Let’s take some examples of using the MySQL EXCEPT operator.

### **1) Simple MySQL EXCEPT operator example**

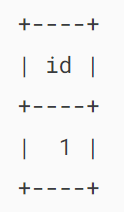
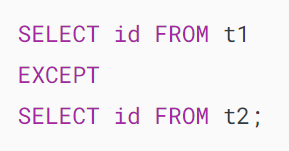
First, create two tables t1 and t2:



Second, insert rows into the t1 and t2 tables:



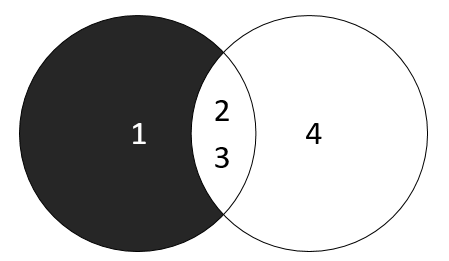
Third, use the EXCEPT operator to find rows that appear in the table t1 but do not appear in the table t2:



In this example, the first query returns a result set (1,2,3) and the second query produces a result set (2,3,4).

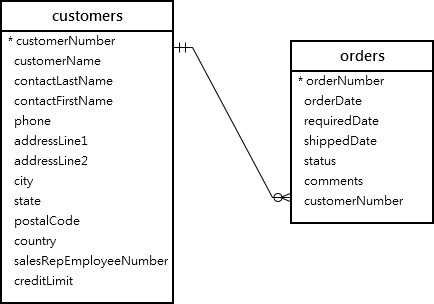
The EXCEPT operator returns the row 1 that appears in the first result set but does not appear in the second result set.

The following Venn diagram illustrates the EXCEPT operation:



### **2) Practical MySQL EXCEPT operator example**

We’ll use the employees and customers tables from the sample database to illustrate the EXCEPT operator:



The following query uses the EXCEPT operator to find the first names that appear in the customers table but do not appear in the employees table:



In this example, the result set uses the firstName column of the first query for its column.

### **3) Using the EXCEPT operator with the ORDER BY clause example**

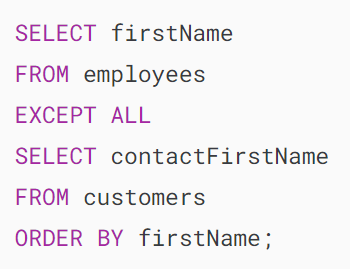
To sort the result set returned by the EXCEPT operator, you use the [ORDER BY](https://www.mysqltutorial.org/mysql-basics/mysql-order-by/) clause.

For example, the following query uses the EXCEPT operator to get the first names that appear in the employees table but do not appear in the customers table and sort the first names alphabetically:

### 

### **4) Using the EXCEPT operator with the ALL option**

The following example uses the EXCEPT operator with the ALL option to retain duplicate first names in the result set:



In the result set, the first name Gerard appears twice.

## **Summary**

* Use the MySQL EXCEPT operator to retrieve rows from one result set that do not appear in another result set.
* EXCEPT DISTINCT removes duplicates while the EXCEPT ALL retains the duplicates.
* The EXCEPT operator uses the DISTINCT option by default.

# **MySQL INTERSECT**

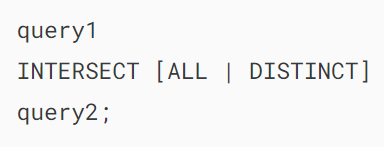
**Summary**: in this tutorial, you will learn about the MySQL INTERSECT operator and how to use it to find the common rows of multiple queries.

Notice that MySQL supported INTERSECT operator in MySQL 8.0.31.

## **Introduction to the MySQL INTERSECT operator**

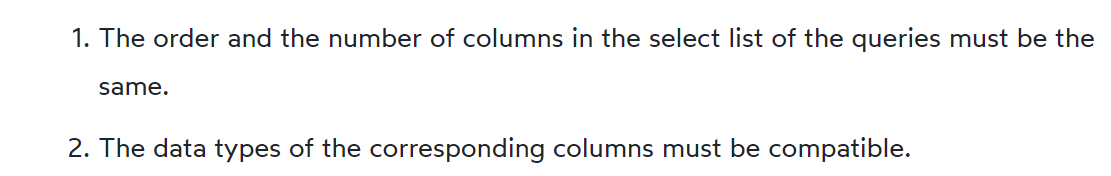
The INTERSECT operator is a set operator that returns the common rows of two or more queries.

Here’s the syntax of the INTERSECT operator:



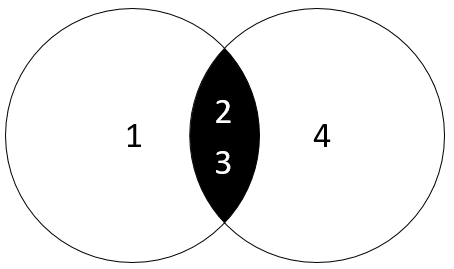
The INTERSECT operator compares the result sets of two queries and returns the common rows.

To use the INTERSECT operator for the queries, follow these rules:



The INTERSECT operator uses the DISTINCT by default. This means that the DISTINCT removes duplicates from either side of the intersection. If you want to retain duplicates, you explicitly specify the ALL option.

The following diagram illustrates the INTERSECT operator.



In this diagram, the left query produces a result set of (1,2,3) and the right query returns a result set of (2,3,4). The INTERSECT operator returns the common rows, which in this case are (2,3).

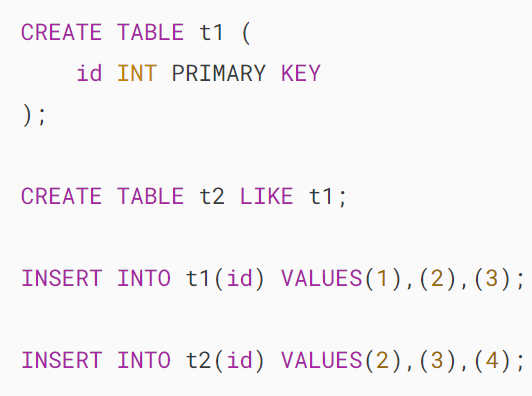
Unlike the [UNION](https://www.mysqltutorial.org/mysql-basics/mysql-union/) operator, which combines the rows of two queries, the INTERSECT operator returns the common rows between queries.

## **MySQL INTERSECT operator examples**

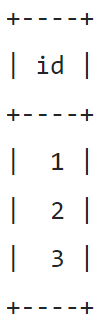
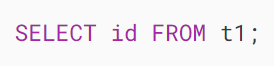
Let’s take an example of using the INTERSECT operator.

### **Setting up sample tables**

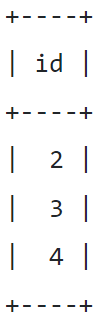
First, [create tables](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) t1 and t2, and [insert data](https://www.mysqltutorial.org/mysql-basics/mysql-insert/) into both tables:



Second, query the data from the t1 table:

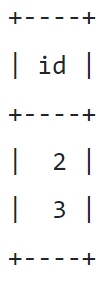
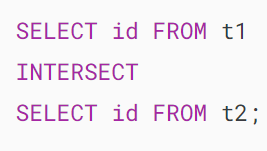


Third, query the data from the t2 table:



### **1) MySQL INTERSECT operator example**

The following example uses the INTERSECT operator to get a result set that contains the rows common to both t1 and t2 tables:

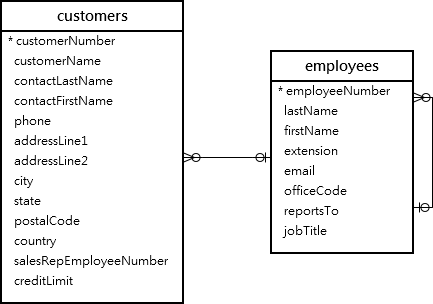


The query is equivalent to the following query that uses the DISTINCT explicitly:

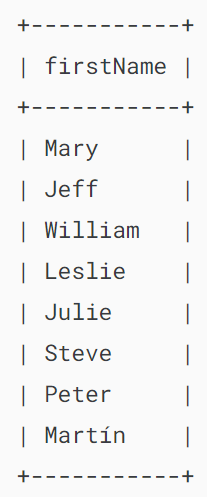
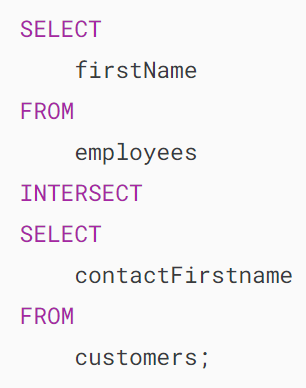
### 

### **2) A practical MySQL INTERSECT example**

We’ll use the customers and employees tables from the [sample database](https://www.mysqltutorial.org/getting-started-with-mysql/mysql-sample-database/) for the demonstration:



The following query uses the INTERSECT operator to find the common first names of customers and employees:



### **3) Using INTERSECT operator with ORDER BY clause**

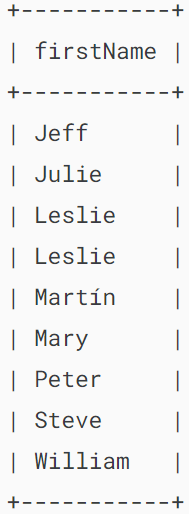
To sort the result set returned by the INTERSECT operator, you use the [ORDER BY](https://www.mysqltutorial.org/mysql-basics/mysql-order-by/) clause. MySQL uses the columns of the first query as the column name of the result set.

For example, the following query uses the INTERSECT operator to find the common first names of employees and customers and sort them in ascending order:

### 

### **4) Using INTERSECT operator with ALL option example**

The following example uses the INTERSECT operator with the ALL option that finds the common first names of employees and customers and retains the duplicates:



## **Summary**

* Use the MySQL INTERSECT operator to find the rows that are common to multiple query results.
* Use INTERSECT DISTINCT to remove the duplicates from the result sets and INTERSECT ALL to retain the duplicates.
* The INTERSECT operator uses DISTINCT by default.

# **MySQL CTE**

**Summary**: in this tutorial, you will learn how to use MySQL CTE or common table expression to construct complex queries in a more readable manner.

MySQL introduced the common table expression or CTE  feature since version 8.0 so you should have MySQL 8.0+ to practice with the statements in this tutorial.

## **Introduction to the common table expression (CTE)**

A common table expression is a named temporary result set that exists solely within the execution scope of a single SQL statement, such as [SELECT](https://www.mysqltutorial.org/mysql-basics/mysql-select-from/), [INSERT](https://www.mysqltutorial.org/mysql-basics/mysql-insert/), [UPDATE](https://www.mysqltutorial.org/mysql-basics/mysql-update/), or [DELETE](https://www.mysqltutorial.org/mysql-basics/mysql-delete/).

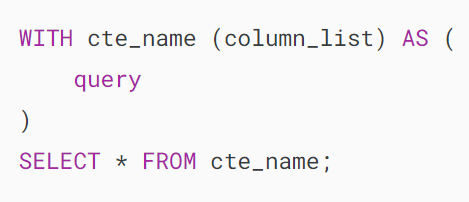
Similar to a [derived table](https://www.mysqltutorial.org/mysql-basics/mysql-derived-table/), a common table expression (CTE) is not stored as an object and lasts only during the query execution.

Unlike a derived table, a common table expression (CTE) can be self-referencing (in the case of a [recursive CTE](https://www.mysqltutorial.org/mysql-basics/mysql-recursive-cte/)) or referenced multiple times within the same query. Moreover, a CTE offers enhanced readability and performance compared to a derived table.

## **MySQL CTE syntax**

The structure of a CTE includes the name, an optional column list, and a query that defines the CTE. After you define a CTE, you can use like a view in the SELECT, INSERT, UPDATE, DELETE, or CREATE VIEW statement.

The following illustrates the basic syntax of a CTE:



In this syntax:

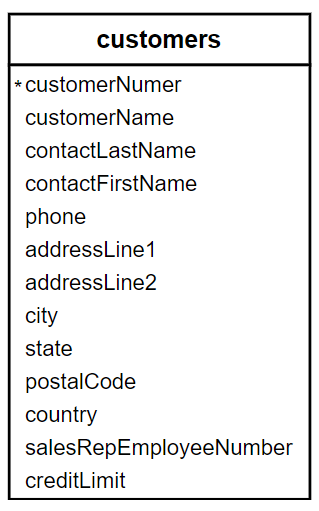
* WITH cte\_name (column\_list) AS: define a CTE with the name cte\_name and a list of columns (column\_list) that the CTE will have. The column\_list is optional if you don’t specify column\_list, the CTE will inherit the column names from the result of the query.
* query: This is the query that defines the CTE. MySQL will store the result of the query in the CTE.
* SELECT \* FROM cte\_name: This is an example of how you can use the CTE. In this case, it is a simple SELECT statement that retrieves all columns from the CTE.

## **MySQL CTE examples**

Let’s explore some examples of using MySQL CTE.

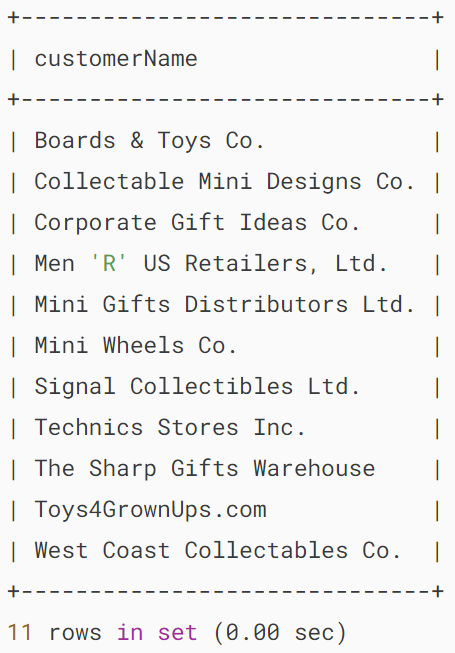
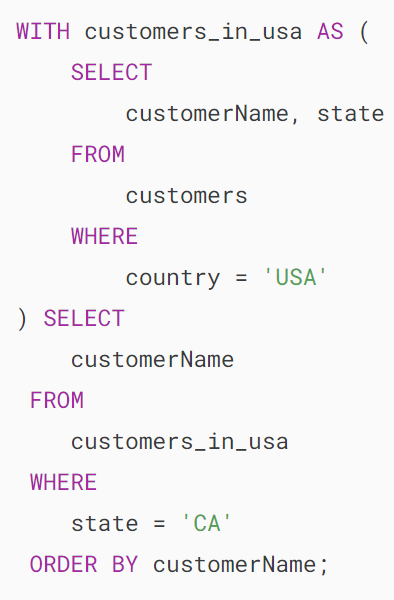
### **1) Basic MySQL CTE example**

We’ll use the customers table from the [sample database](https://www.mysqltutorial.org/getting-started-with-mysql/mysql-sample-database/) for demonstration:



The following example illustrates how to use a CTE for querying data from the customers table in the [sample database](https://www.mysqltutorial.org/getting-started-with-mysql/mysql-sample-database/).

Note that this example is only for demonstration purposes to make it easy for you to understand the CTE concept.



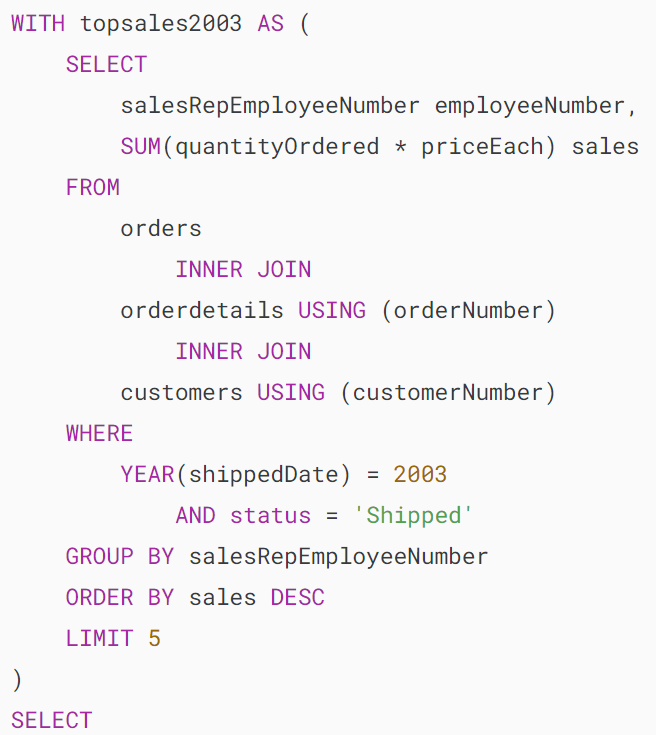
How it works.

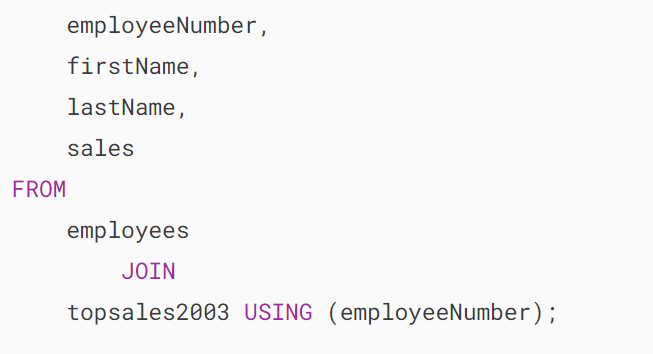
* First, define a CTE with the name customers\_in\_usa that stores the customer name and state of customers in the USA. The defining query retrieves data from the customers table.
* Second, select the customers located in California from the CTE.

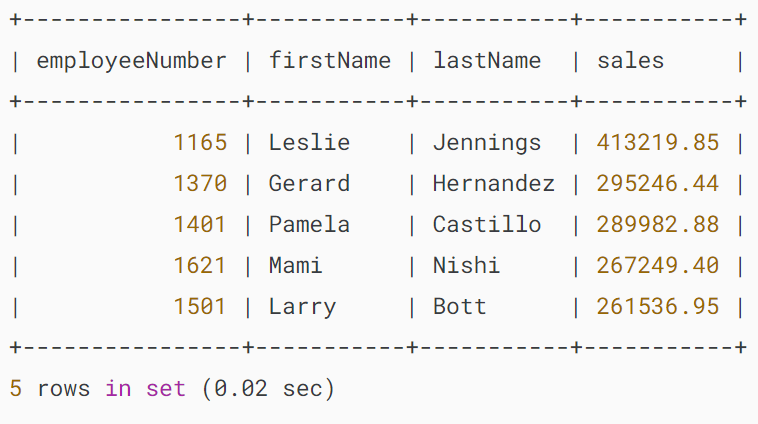
### **2) Getting top sales using a CTE**

We’ll use the orders, orderdetails, and employees from the sample database:

The following example uses a CTE to retrieve the top 5 sales representatives based on their total sales in the year 2003:





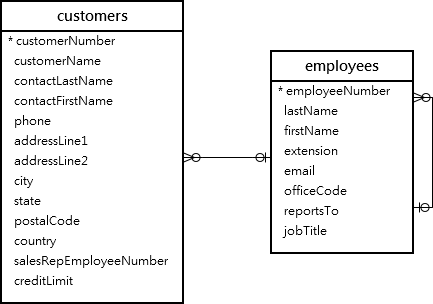


How it works.

* First, define a CTE that retrieves the top 5 employees with their total sales in 2003.
* Second, join the CTE with the employees table to include the first and last names of the sales representatives.

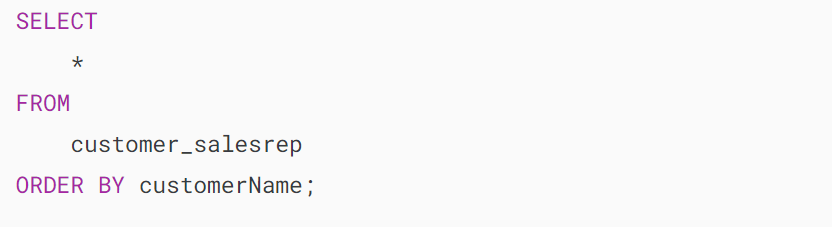
### **3) Using multiple CTEs**

We’ll use the customers and employees from the sample database:



The following example uses multiple CTEs to map the customers with their respective sales representatives:





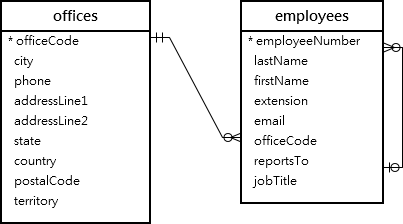


How it works.

* CTE salesrep: Select employeeNumber and concatenate the firstName and lastName columns to create a column named salesrepName, and include only employees with the job title 'Sales Rep'.
* CTE customer\_salesrep: selects customerName and salesrepName by joining the customers table with the salesrep CTE based on the common column employeeNumber.
* Main query: Select all columns from the customer\_salesrep CTE.

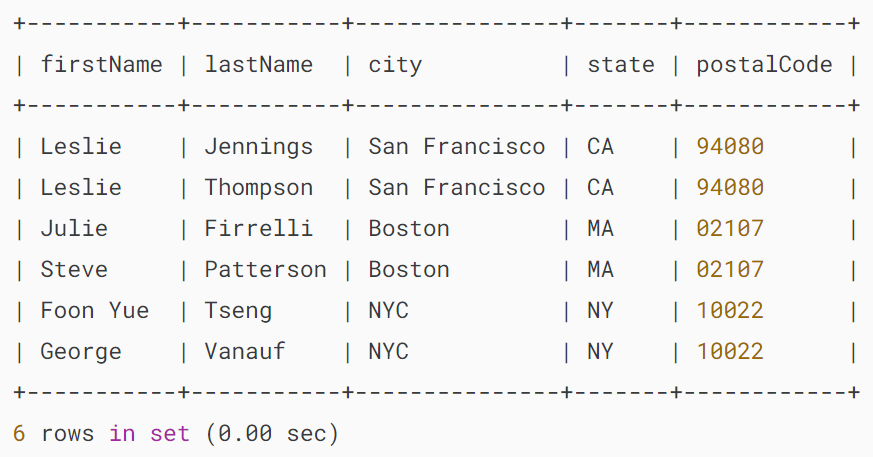
### **4) Joining two CTEs example**

We’ll use the offices and employees tables from the sample database:



The following example is creating two CTEs and joining them to get the Sales Representatives located in the USA, including their office information:





How it works.

* CTE e: Retrieve employees whose job title is Sales Rep.
* CTE o: Retrieve offices located in the USA.
* Main query: Joins the CTE e and o using the officeCode column.

## **Summary**

* Use MySQL CTEs to break down complex queries into simpler, more manageable queries. Each CTE represents a temporary result set that can be referenced within the main query.

# **MySQL Recursive CTE**

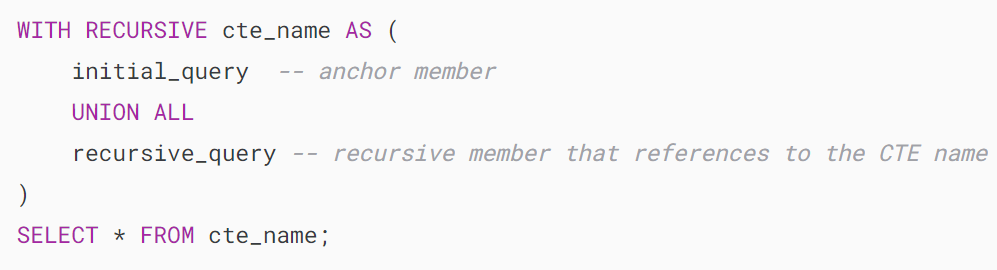
**Summary**: in this tutorial, you will learn about MySQL recursive CTE and how to use it to traverse hierarchical data.

Notice that a [common table expression](https://www.mysqltutorial.org/mysql-basics/mysql-cte/) (CTE) is only available in MySQL version 8.0 or later. Therefore, ensure that you have the right version of MySQL installed to use the statements in this tutorial.

## **Introduction to MySQL recursive CTE**

In MySQL, a recursive Common Table Expression (CTE) is a named temporary result set that references itself in the recursive member, enabling the hierarchical traversal or iteration over data until a specified termination condition is met.

The following illustrates the syntax of a recursive CTE:



A recursive CTE consists of three main parts:

* An initial [query](https://www.mysqltutorial.org/mysql-basics/mysql-select-from/) that forms the base result set of the CTE structure. The initial query part is referred to as an anchor member.
* A recursive query part is a query that references the CTE name, therefore, it is called a recursive member. The recursive member is joined with the anchor member by a [UNION ALL](https://www.mysqltutorial.org/mysql-basics/mysql-union/) or UNION DISTINCT operator.
* A termination condition that ensures the recursion stops when the recursive member returns no row.

The execution order of a recursive CTE is as follows:

1. First, separate the members into two: anchor and recursive members.
2. Next, execute the anchor member to form the base result set ( R0) and use this base result set for the next iteration.
3. Then, execute the recursive member with Ri result set as an input and make Ri+1 as an output.
4. After that, repeat the third step until the recursive member returns an empty result set, in other words, the termination condition is met.
5. Finally, combine result sets from R0 to Rn using UNION ALL operator.

## **Recursive member restrictions**

The recursive member must not contain the following constructs:

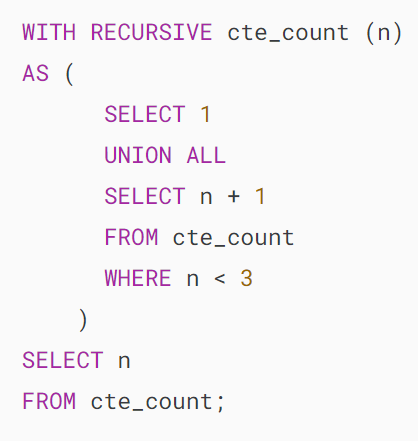
* Aggregate functions e.g., [MAX](https://www.mysqltutorial.org/mysql-aggregate-functions/mysql-max-function/), [MIN](https://www.mysqltutorial.org/mysql-aggregate-functions/mysql-min/), [SUM](https://www.mysqltutorial.org/mysql-aggregate-functions/mysql-sum/), [AVG](https://www.mysqltutorial.org/mysql-aggregate-functions/mysql-avg/), [COUNT](https://www.mysqltutorial.org/mysql-aggregate-functions/mysql-count/), etc.
* [GROUP BY](https://www.mysqltutorial.org/mysql-basics/mysql-group-by/) clause
* [ORDER BY](https://www.mysqltutorial.org/mysql-basics/mysql-order-by/) clause
* [LIMIT](https://www.mysqltutorial.org/mysql-basics/mysql-limit/)clause
* [DISTINCT](https://www.mysqltutorial.org/mysql-basics/mysql-distinct/)

Note that the above constraint does not apply to the anchor member. Furthermore, the restriction on using DISTINCT only applies when you use UNION operator. If you use the UNION DISTINCT operator, the DISTINCT is permitted.

In addition, the recursive member can reference the CTE name only once in its FROM clause and not in any [subquery](https://www.mysqltutorial.org/mysql-basics/mysql-subquery/).

## **Basic MySQL recursive CTE example**

See the following simple recursive CTE example:

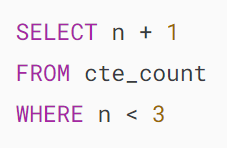


In this example, the following query:



is the anchor member that returns 1 as the base result set.

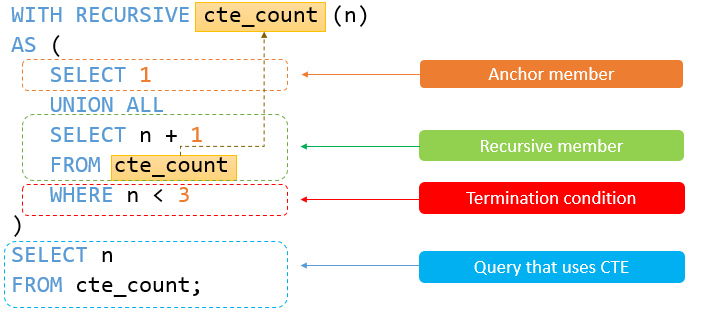
The following query



is the recursive member because it references the name of the CTE which is cte\_count.

The expression n < 3 in the recursive member is the termination condition. Once n equals 3, the recursive member returns an empty set that will stop the recursion.

The following picture illustrates the elements of CTE above:



The recursive CTE returns the following output:

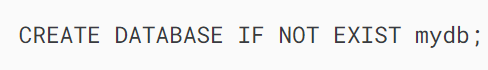


The execution steps of the recursive CTE are as follows:

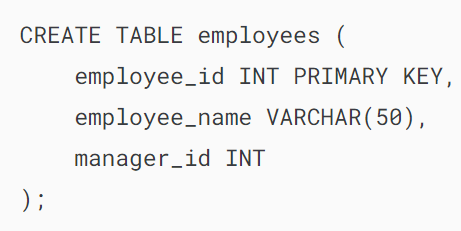
* First, separate the anchor and recursive members.
* Next, the anchor member forms the initial row ( SELECT 1) therefore the first iteration produces 1 + 1 = 2 with n = 1.
* Then, the second iteration operates on the output of the first iteration (2) and produces 2 + 1 = 3 with n = 2.
* After that, before the third operation ( n = 3), the termination condition ( n < 3) is met therefore the query stops.
* Finally, combine all result sets 1, 2, and 3 using the UNION ALL operator

## **Using MySQL recursive CTE to traverse the hierarchical data**

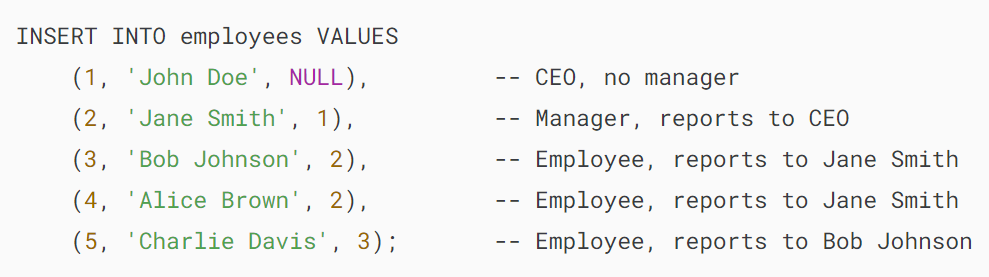
First, create a new database called mydb:



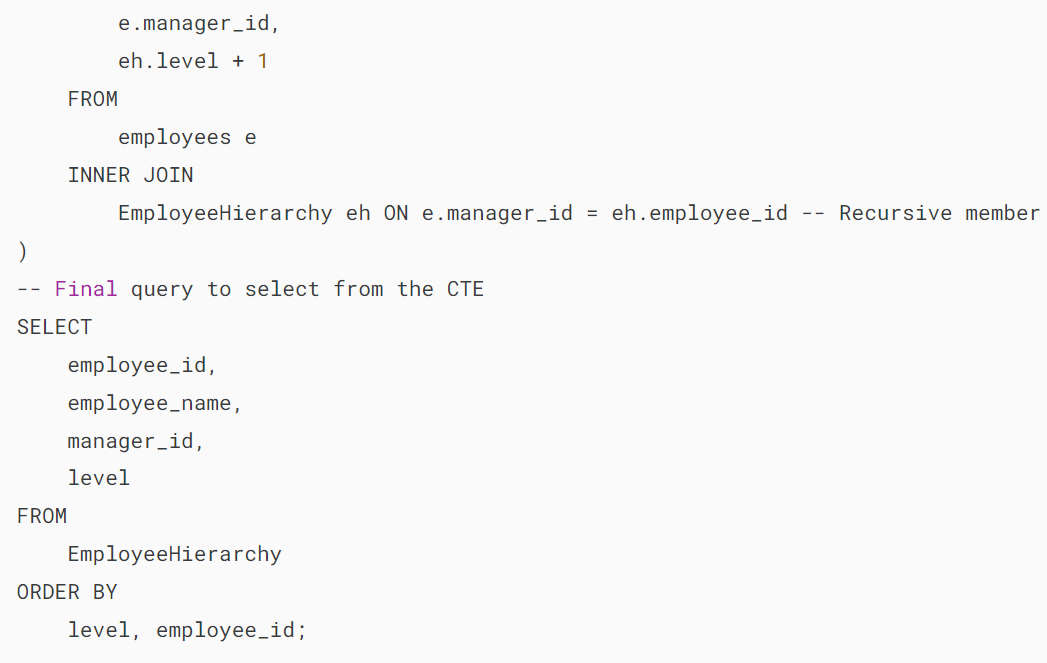
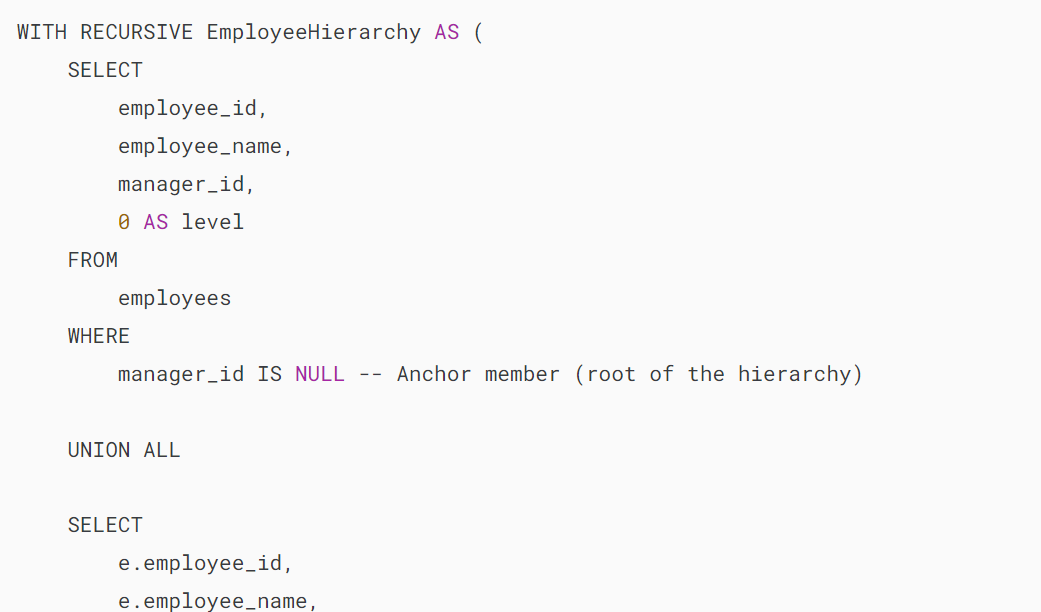
Second, change the current database to mydb:

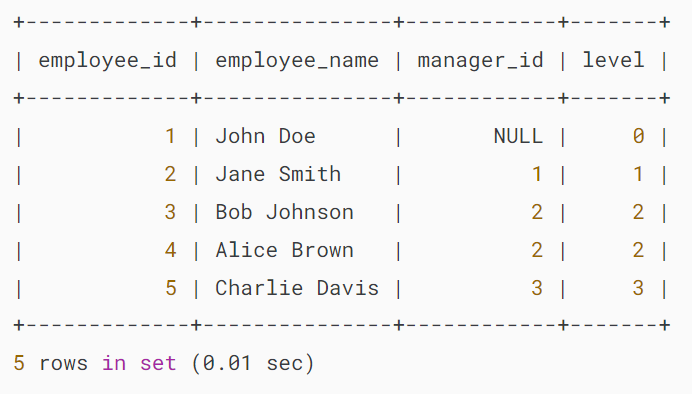


Third, [insert some rows](https://www.mysqltutorial.org/mysql-basics/mysql-insert-multiple-rows/) into the employees table:



Finally, traverse the hierarchical data in the employees table using a recursive CTE:





How it works.

* Define The CTE with the name EmployeeHierarchy.
* Define an anchor member that selects employees who do not have a manager (manager\_id IS NULL), starting with the root of the hierarchy (CEO).
* Use a recursive member to join the employees table with the CTE on the condition that the manager\_id in the employees table matches the employee\_id in the CTE, effectively traversing the hierarchy.
* Select information from the CTE, including the employee’s ID, name, manager’s ID, and the level in the hierarchy in the final query. And sort the result set by level and employee ID.

## **Summary**

* Use MySQL recursive CTE to traverse hierarchical data.