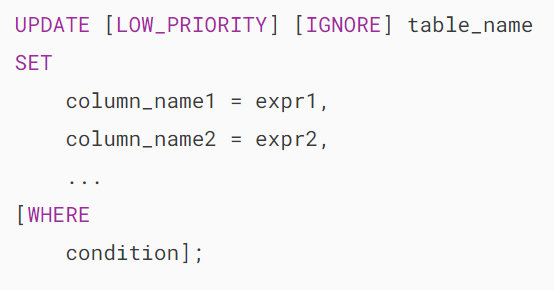
# **MySQL UPDATE**

**Summary**: updating data is one of the most important tasks when you work with the database. In this tutorial, you will learn how to use the MySQL UPDATE statement to update data in a table.

## **Introduction to MySQL UPDATE statement**

The UPDATE statement updates data in a table. It allows you to change the values in one or more columns of a single row or multiple rows.

The following illustrates the basic syntax of the UPDATE statement:



In this syntax:

* First, specify the name of the table that you want to update data after the UPDATE keyword.
* Second, specify which column you want to update and the new value in the SET clause. To update values in multiple columns, you use a list of comma-separated assignments by supplying a value in each column’s assignment in the form of a literal value, an expression, or a [subquery](https://www.mysqltutorial.org/mysql-basics/mysql-subquery/).
* Third, specify which rows to be updated using a condition in the [WHERE](https://www.mysqltutorial.org/mysql-basics/mysql-where/) clause. The WHERE clause is optional. If you omit it, the UPDATE statement will modify all rows in the table.

Notice that the WHERE clause is so important that you should not forget. Sometimes, you may want to update just one row; However, you may forget the WHERE clause and accidentally update all rows of the table.

MySQL supports two modifiers in the UPDATE statement.

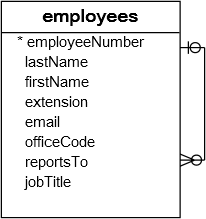
1. The LOW\_PRIORITY modifier instructs the UPDATE statement to delay the update until there is no connection reading data from the table. The LOW\_PRIORITY takes effect for the [storage engines](https://www.mysqltutorial.org/mysql-administration/mysql-storage-engines/) that use table-level [locking](https://www.mysqltutorial.org/mysql-basics/mysql-table-locking/) only such as MyISAM, MERGE, and MEMORY.
2. The IGNORE modifier enables the UPDATE statement to continue updating rows even if errors occurred. The rows that cause errors such as duplicate-key conflicts are not updated.

## **MySQL UPDATE examples**

Let’s practice the UPDATE statement.

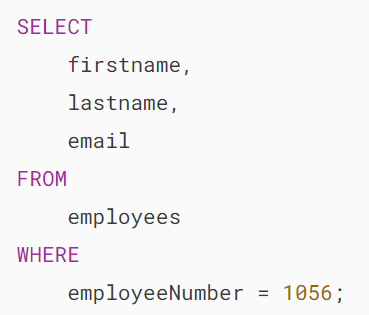
### **1) Using MySQL UPDATE to modify values in a single column example**

See the following employees table from the [sample database](https://www.mysqltutorial.org/getting-started-with-mysql/mysql-sample-database/).



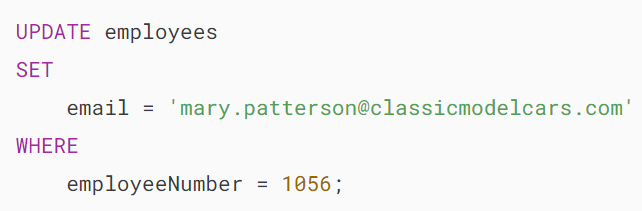
In this example, we will update the email of Mary Patterson to the new email mary.patterso@classicmodelcars.com.

First, find Mary’s email from the employees table using the following [SELECT](https://www.mysqltutorial.org/mysql-basics/mysql-select-from/) statement:

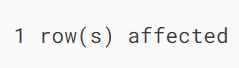


MySQL Update example

Second, update the email address of Mary to the new email mary.patterson@classicmodelcars.com :



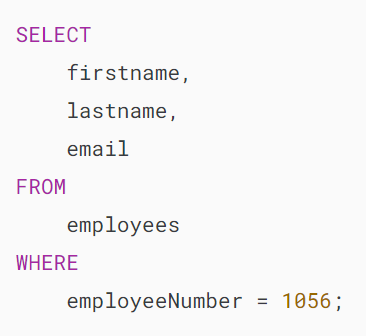
MySQL issued the number of rows affected:



In this UPDATE statement:

* The [WHERE](https://www.mysqltutorial.org/mysql-basics/mysql-where/) clause specifies the row with employee number 1056 will be updated.
* The SET clause sets the value of the email column to the new email.

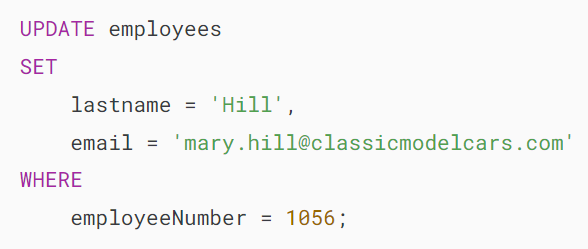
Third,  execute the SELECT statement again to verify the change:



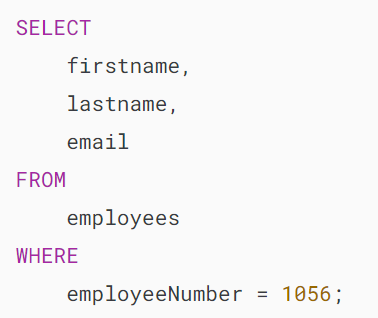
MySQL UPDATE table example

### **2) Using MySQL UPDATE to modify values in multiple columns**

To update values in the multiple columns, you need to specify the assignments in the SET clause. For example, the following statement updates both last name and email columns of employee number 1056:



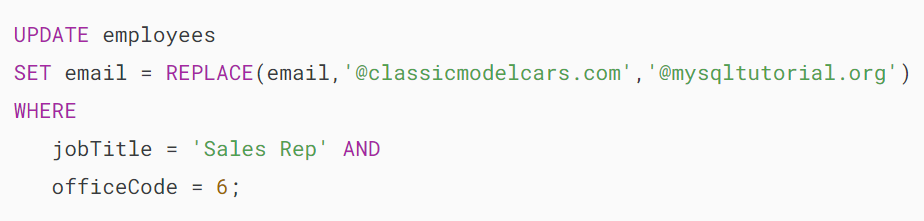
Let’s verify the changes:



MySQL UPDATE multiple columns

### **3) Using MySQL UPDATE to replace string example**

The following example updates the domain parts of emails of all Sales Reps with office code 6:

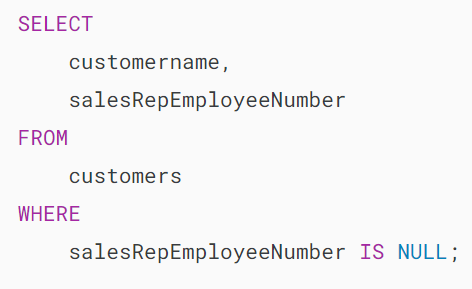


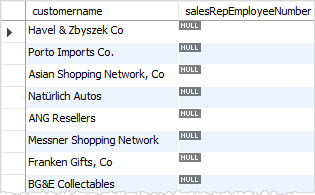
In this example, the [REPLACE](https://www.mysqltutorial.org/mysql-string-functions/mysql-replace-function/)() function replaces @classicmodelcars.com in the email column with @mysqltutorial.org.

### **4) Using MySQL UPDATE to update rows returned by a SELECT statement example**

You can supply the values for the SET clause from a SELECT statement that queries data from other tables.

For example, in the customers table, some customers do not have any sale representative. The value of the column saleRepEmployeeNumber is NULL as follows:

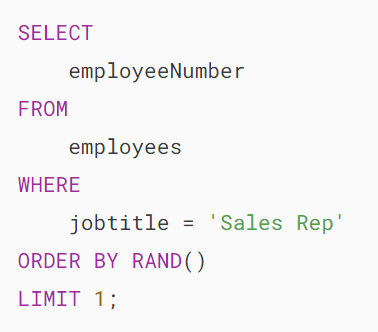




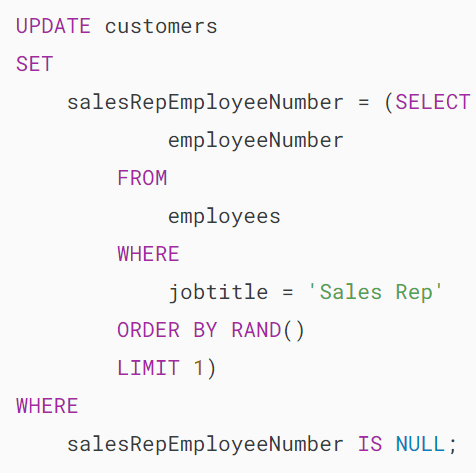
We can take a sale representative and update for those customers.

To do this, we can select a random employee whose job title is Sales Rep from the  employees table and update it for the  employees table.

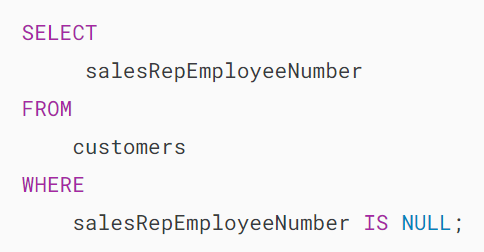
This query [selects a random](https://www.mysqltutorial.org/mysql-basics/mysql-select-random/) employee from the table employees whose job title is the Sales Rep.



To update the sales representative employee number  column in the customers table, we place the query above in the SET clause of the UPDATE statement as follows:



If you query data from the  employees table, you will see that every customer has a sales representative. In other words, the following query returns no row.



# **MySQL UPDATE JOIN**

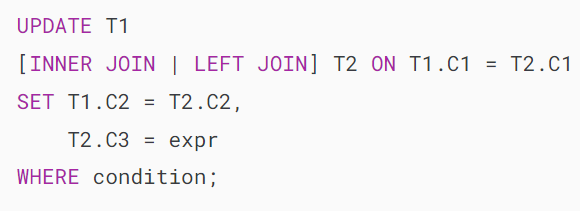
**Summary**: in this tutorial, you will learn how to use the **MySQL UPDATE JOIN** statement to perform the cross-table update.

## **Introduction to MySQL UPDATE JOIN statement**

You often use [joins](https://www.mysqltutorial.org/mysql-basics/mysql-join/) to query rows from a table that have (in the case of [INNER JOIN](https://www.mysqltutorial.org/mysql-basics/mysql-inner-join/)) or may not have (in the case of [LEFT JOIN](https://www.mysqltutorial.org/mysql-basics/mysql-left-join/)) matching rows in another table.

In MySQL, you can also use the JOIN clauses in the [UPDATE](https://www.mysqltutorial.org/mysql-basics/mysql-update/) statement to update rows in one table based on values from another table. The UPDATE JOIN statement is useful when you need to modify data across related tables.

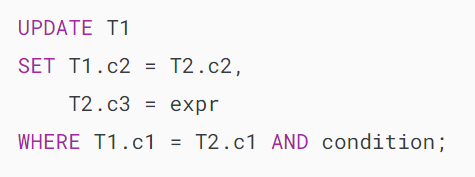
The syntax of the MySQL UPDATE JOIN  is as follows:



How it works:

* First, specify the table that you want to update after the UPDATE keyword (T1).
* Second, use either INNER JOIN  or LEFT JOIN  and a join predicate. The JOIN clause must appear right after the UPDATE clause.
* Third, assign new values to the columns of the T1 table that you want to update data.
* Finally, specify a condition in the [WHERE](https://www.mysqltutorial.org/mysql-basics/mysql-where/) clause to filter the rows for updating.

If you follow the [UPDATE](https://www.mysqltutorial.org/mysql-basics/mysql-update/) statement tutorial, you will notice that there is another way to update the data across tables using the following syntax:



This UPDATE statement works the same as UPDATE JOIN  with an implicit INNER JOIN  clause. It means you can rewrite the above statement as follows:

## 

## **MySQL UPDATE JOIN examples**

Let’s explore some examples of using the UPDATE JOIN  statement to have a better understanding.

We’ll create a new database called hr that consists of two tables:

* The  employees table stores employee data with employee id, name, performance, and salary.
* The merits table stores employee performance and merit percentage.

The following statements create and load data in the hr sample database:

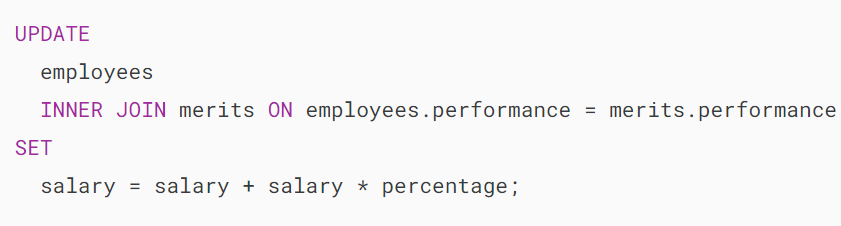
### 

### 

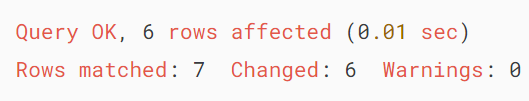
### **MySQL UPDATE JOIN example with INNER JOIN clause**

Suppose you want to increment each employee’s salary by a percentage based on their performance.

The following statement updates the salary column in the employees table by performing an INNER JOIN with the merits table based on matching values in the performance column:



Output:



How the query works:

* **UPDATE employees:** Specifies that we want to update the employees table.
* **INNER JOIN merits ON employees.performance = merits.performance:** Performs an INNER JOIN with the merits table based on the condition that the performance column values match between the employees and merits tables. This means only the rows with matching performance values in both tables will be considered for the update.
* **SET salary = salary + salary \* percentage:** Updates the salary column in the employees table. Each employee’s salary is updated based on the percentage: salary + salary \* percentage.

Because the UPDATE statement does not have the WHERE clause, it updates all rows employees table.

The following statement retrieves data from the employees table to verify the updates:

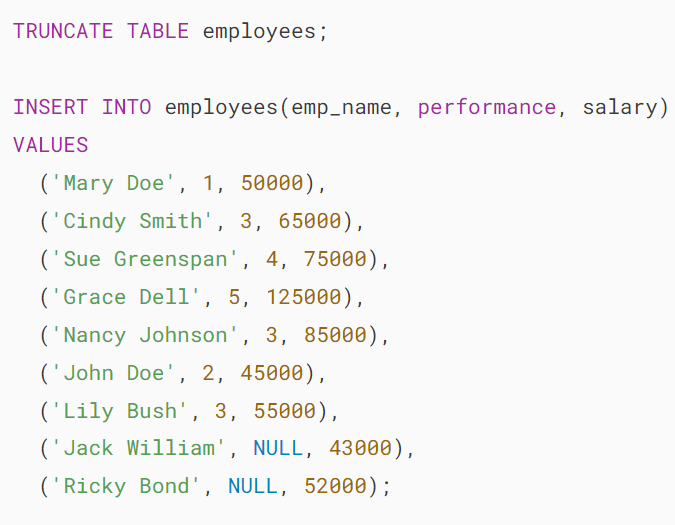


Output:

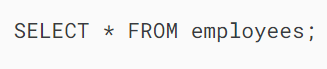
### 

### **MySQL UPDATE JOIN example with LEFT JOIN**

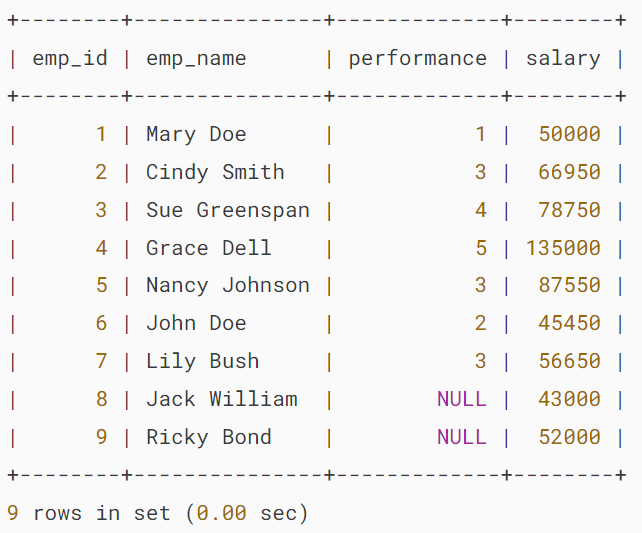
Suppose the company has two new hires employees with the performances are NULL:



Because these employees are new hires so their performance data is not available or NULL:



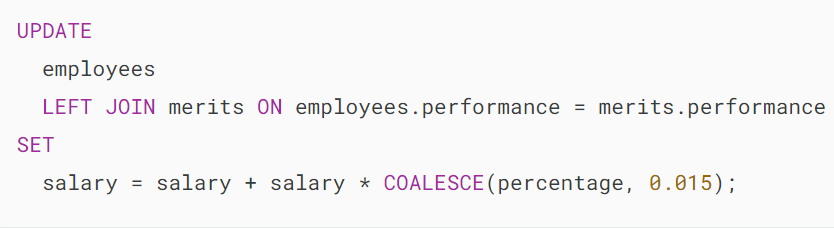
Output:



To raise the salary for all employees including new hires, you cannot use the UPDATE INNER JOIN statement because the performance scores of the new hires are not available in the merits table. This is where the UPDATE LEFT JOIN statement comes to the rescue.

The UPDATE LEFT JOIN  statement updates a row in a table when it does not have a corresponding row in another table.

For example, you can increase the salary for a new hire by 1.5% and other employees based on their performances using the following statement:

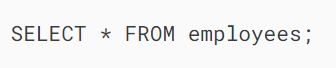


Output:



The salary of the employee (Mary Doe) who has a performance score of 1 was not updated. Therefore, we have 8 rows changed.

The following statement retrieves the data from the employees table to verify the updates:



Output:

## 

## **Summary**

* Use the MySQL UPDATE JOIN  with the INNER JOIN  or LEFT JOIN  clauses to perform cross-table updates.

# **MySQL REPLACE**

**Summary**: in this tutorial, you will learn how to use the MySQL REPLACE statement to insert or update data in database tables.

## **Introduction to MySQL REPLACE statement**

The MySQL REPLACE statement is an extension to the [SQL Standard](http://www.sqltutorial.org/). The MySQL REPLACE statement works as follows:

Step 1. [Insert a new row into the table](https://www.mysqltutorial.org/mysql-basics/mysql-insert/), if a duplicate key error occurs.

Step 2. If the insertion fails due to a duplicate-key error occurs:

* [Delete](https://www.mysqltutorial.org/mysql-basics/mysql-delete/) the conflicting row that causes the duplicate key error from the table.
* Insert the new row into the table again.

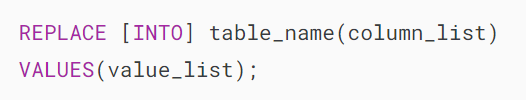
To determine whether the new row that already exists in the table, MySQL uses [PRIMARY KEY](https://www.mysqltutorial.org/mysql-basics/mysql-primary-key/) or [UNIQUE KEY](https://www.mysqltutorial.org/mysql-unique/) index. If the table does not have one of these indexes, the REPLACE statement works like an INSERT statement.

To use the REPLACE statement, you need to have at least both INSERT and DELETE privileges for the table.

Notice that MySQL has the [REPLACE](https://www.mysqltutorial.org/mysql-string-functions/mysql-replace-function/) string function which is not the REPLACE statement covered in this tutorial.

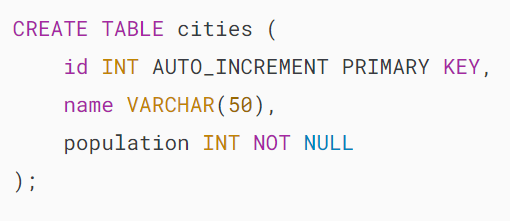
## **Using MySQL REPLACE to insert a new row**

The following illustrates the syntax of the REPLACE statement:



It is similar to the INSERT statement except for the keyword REPLACE. Let’s take a look at the following example of using the REPLACE statement to see how it works.

First, [create a new table](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) named cities as follows:

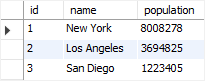


Next, [insert some rows](https://www.mysqltutorial.org/mysql-basics/mysql-insert-multiple-rows/) into the cities table:

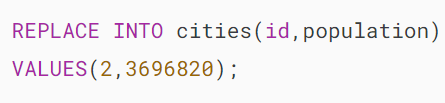


Then, [query data](https://www.mysqltutorial.org/mysql-basics/mysql-select-from/) from the cities table to verify the insert operation.

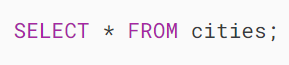


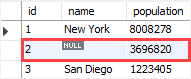


After that, use the REPLACE statement to update the population of the Los Angeles city to 3696820.



Finally, query the data of the cities table again to verify the replacement.





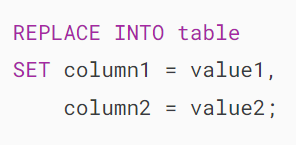
The value of the name column is NULL now.

The REPLACE statement works as follows:

1. First, the REPLACE statement attempted to insert a new row into cities the table. The insertion failed because the id 2 already exists in the cities table.
2. Then, the REPLACE statement deleted the row with id 2 and inserted a new row with the same id 2 and population 3696820. Because no value is specified for the name column, it was set to NULL.

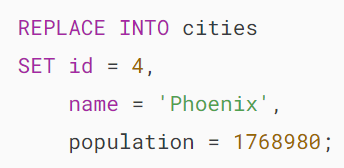
## **Using MySQL REPLACE statement to update a row**

The following illustrates how to use the REPLACE statement to update data:



This statement is like the UPDATE statement except for the REPLACE keyword. In addition, it has no [WHERE](https://www.mysqltutorial.org/mysql-basics/mysql-where/) clause.

This example uses the REPLACE statement to update the population of the city Phoenix to 1768980:

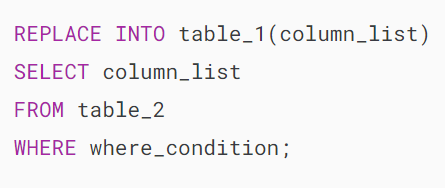


Unlike the UPDATE statement, if you don’t specify the value for the column in the SET clause, the REPLACE statement will use the default value of that column.

## 

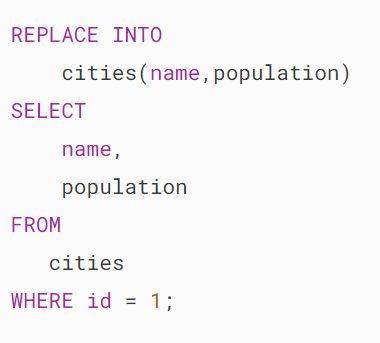
## **Using MySQL REPLACE to insert data from a SELECT statement**

The following illustrates the REPLACE statement that inserts data into a table with the data coming from a query.



Note that this form of the REPLACE statement is similar to [INSERT INTO SELECT](https://www.mysqltutorial.org/mysql-basics/mysql-insert-into-select/) statement.

The following statement uses the REPLACE INTO statement to copy a row within the same table:



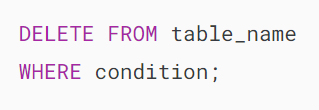
# **MySQL DELETE**

**Summary**: in this tutorial, you will learn how to use the **MySQL DELETE** statement to delete rows from a table and return the number of deleted rows.

## **Introduction to MySQL DELETE statement**

The DELETE statement allows you to delete rows from a table and returns the number of deleted rows.

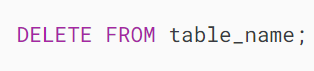
Here’s the basic syntax of the DELETE statement:



In this syntax:

* First, specify the table from which you delete data after the FROM keyword.
* Second, specify a condition to determine which rows to delete in the WHERE clause.

The WHERE clause is optional. If you omit the WHERE clause, the DELETE statement will delete all rows in the table:



Note that to delete data from multiple related tables, you use the [DELETE JOIN](https://www.mysqltutorial.org/mysql-basics/mysql-delete-join/) statement.

When you need to remove all rows from a large table and don’t need to know the exact number of rows deleted, you should use the [TRUNCATE TABLE](https://www.mysqltutorial.org/mysql-basics/mysql-truncate-table/) statement for better performance.

In a table that has a [foreign key](https://www.mysqltutorial.org/mysql-basics/mysql-foreign-key/) constraint, when you delete rows from the parent table, MySQL automatically deletes the rows in the child table if the foreign key uses the [ON DELETE CASCADE](https://www.mysqltutorial.org/mysql-basics/mysql-on-delete-cascade/) option.

## **MySQL DELETE statement examples**

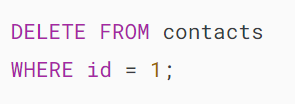
We’ll create a table called contacts with some sample data for the demonstration:

### 

### 

### **1) Delete a row example**

The following example uses the DELETE statement to delete a single row from the contacts table:



Output:

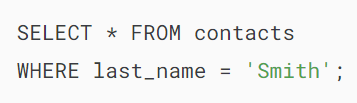


In this example, the DELETE statement deletes the row with id 1. Since the condition returns only one row, the DELETE statement deleted a single row.

The output indicates that one row was deleted.

### **2) Delete multiple rows example**

The following statement retrieves the contacts with the last name Smith:

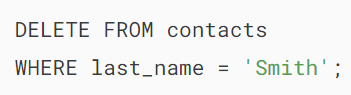


Output:



It returns 6 rows.

To delete these 6 rows, you can use the following DELETE statement:



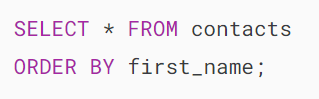
Output:



The output indicates that 6 rows were deleted.

### **3) Using MySQL DELETE statement with LIMIT clause**

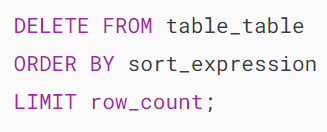
The following statement retrieves the contacts and [sorts them](https://www.mysqltutorial.org/mysql-basics/mysql-order-by/) by first names:



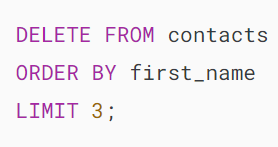
Output:



To delete the first three rows, you can use the DELETE statement with the [ORDER BY](https://www.mysqltutorial.org/mysql-basics/mysql-order-by/) and [LIMIT](https://www.mysqltutorial.org/mysql-basics/mysql-limit/) clauses:



For example, the following example uses the DELETE statement to delete the first three contacts sorted by first names:

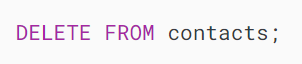


Output:

### 

### **4) Using MySQL DELETE statement to delete all rows**

The following example uses the DELETE statement without a WHERE clause to delete all rows from the contacts table:

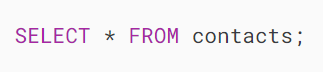


Output:



The statement deleted all rows (10 rows) from the contacts table.

If you retrieve data from the contacts table, you’ll see an empty result set:



Output:

## 

## **Summary**

* Use the DELETE statement to delete one or more rows from a table.
* Use the DELETE statement without a WHERE clause to delete all rows from a table.
* Use the DELETE statement with a LIMIT clause to delete several rows from a table.

# **MySQL DELETE JOIN**

**Summary**: in this tutorial, we will show you how to delete data from multiple tables by using **MySQL DELETE JOIN**statement.

In the previous tutorial, you learned how to delete rows of multiple tables by using:

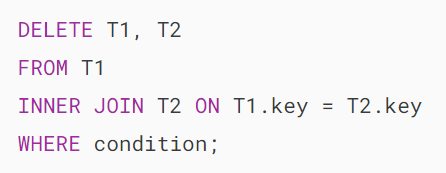
* A single [DELETE](https://www.mysqltutorial.org/mysql-basics/mysql-delete/) statement on multiple tables.
* A single DELETE statement on multiple related tables which the child table has an [ON DELETE CASCADE](https://www.mysqltutorial.org/mysql-basics/mysql-on-delete-cascade/) referential action for the [foreign key](https://www.mysqltutorial.org/mysql-basics/mysql-foreign-key/).

This tutorial introduces to you a more flexible way to delete data from multiple tables using [INNER JOIN](https://www.mysqltutorial.org/mysql-basics/mysql-inner-join/) or [LEFT JOIN](https://www.mysqltutorial.org/mysql-basics/mysql-left-join/) clause with the DELETE statement.

## **MySQL DELETE JOIN with INNER JOIN**

MySQL allows you to use the INNER JOIN clause in the DELETE statement to delete rows from one table that has matching rows in another table.

For example, to delete rows from both T1 and T2 tables that meet a specified condition, you use the following statement:



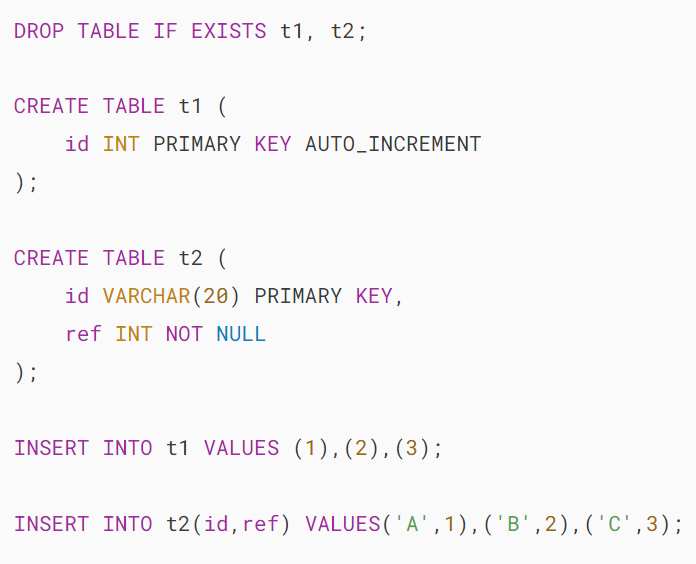
Notice that you place table names T1 and T2 between the DELETE and FROM keywords. If you omit T1 table, the DELETE statement only deletes rows in T2 table. Similarly, if you omit T2 table, the DELETE statement will delete only rows in T1 table.

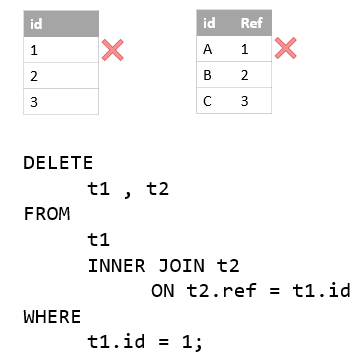
The expression T1.key = T2.key specifies the condition for matching rows between T1 andT2 tables that will be deleted.

The condition of the  WHERE clause determines rows in the T1 and T2 that will be deleted.

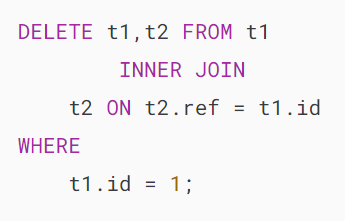
### **MySQL DELETE JOIN with INNER JOIN example**

Suppose, we have two tables t1 and t2 with the following structures and data:

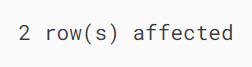




The following statement deletes the row with id 1 in the t1 table and also row with ref 1 in the t2 table using DELETE...INNER JOIN statement:



The statement returned the following message:



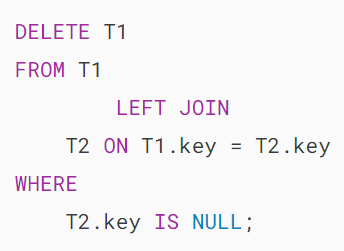
It indicated that two rows had been deleted.

## **MySQL DELETE JOIN with LEFT JOIN**

We often use the [LEFT JOIN](https://www.mysqltutorial.org/mysql-basics/mysql-left-join/) clause in the SELECT statement to find rows in the left table that have or don’t have matching rows in the right table.

We can also use the LEFT JOIN clause in the DELETE statement to delete rows in a table (left table) that does not have matching rows in another table (right table).

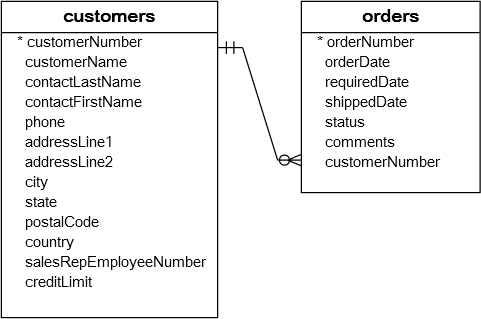
The following syntax illustrates how to use DELETE statement with LEFT JOIN clause to delete rows from T1 table that does not have corresponding rows in the T2 table:



Note that we only put T1 table after the DELETE keyword, not both T1 and T2 tables like we did with the INNER JOIN clause.

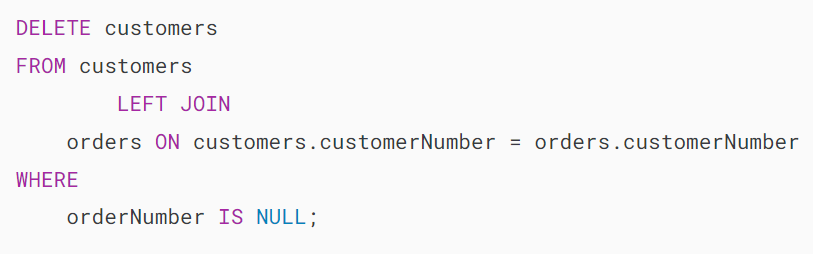
### **MySQL DELETE JOIN with LEFT JOIN example**

See the following customers and orders tables in the sample database:

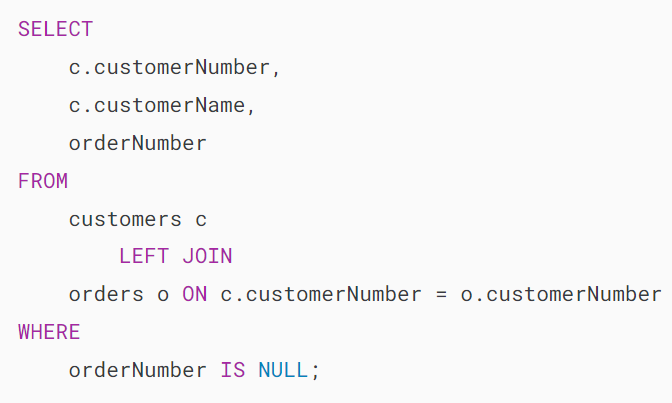


Each customer has zero or more orders. However, each order belongs to one and only one customer.

We can use DELETE statement with LEFT JOIN clause to clean up our customer master data. The following statement removes customers who have not placed any orders:



We can verify the deletion by finding whether customers who do not have any orders exist using the following query:



The query returned an empty result set which is what we expected.

# **MySQL ON DELETE CASCADE**

**Summary**: in this tutorial, you will learn how to use MySQL  ON DELETE CASCADE referential action for a foreign key to delete data from multiple related tables.

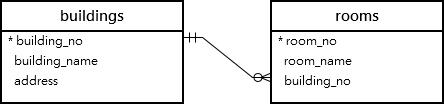
In the previous tutorial, you learned how to delete data from multiple related tables using a single [DELETE](https://www.mysqltutorial.org/mysql-basics/mysql-delete/) statement. However, MySQL provides a more effective way called ON DELETE CASCADE referential action for a [foreign key](https://www.mysqltutorial.org/mysql-basics/mysql-foreign-key/) that allows you to delete data from child tables automatically when you delete the data from the parent table.

## **MySQL ON DELETE CASCADE example**

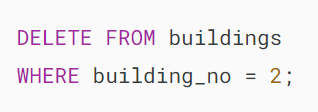
Let’s take a look at an example of using MySQL ON DELETE CASCADE .

Suppose that we have two tables:buildings and rooms . In this database model, each building has one or many rooms. However, each room belongs to one only one building. A room would not exist without a building.

The relationship between the buildings and rooms tables is one-to-many (1:N) as illustrated in the following database diagram:



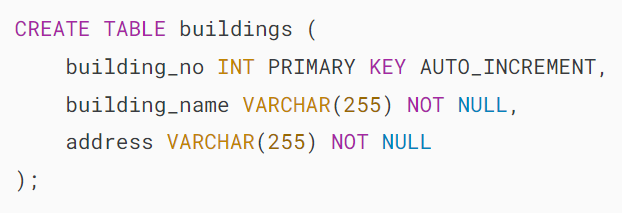
When you delete a row from the buildings table, you also want to delete all rows in the rooms  table that references to the row in the buildings table. For example, when you delete a row with building no. 2 in the buildings  table as the following query:



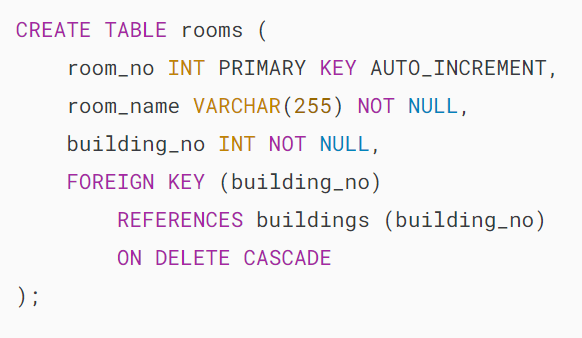
You also want the rows in the rooms table that refers to building number 2 will be also removed.

The following are steps that demonstrate how the ON DELETE CASCADE  referential action works.

**Step 1**. Create the buildings table:

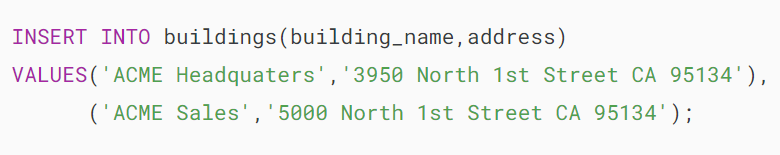


**Step 2**. Create the rooms table:

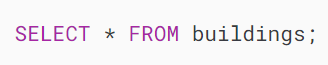


Notice that the ON DELETE CASCADE  clause at the end of the [foreign key constraint](https://www.mysqltutorial.org/mysql-basics/mysql-foreign-key/) definition.

**Step 3**. [Insert rows](https://www.mysqltutorial.org/mysql-basics/mysql-insert-multiple-rows/) into the buildings table:



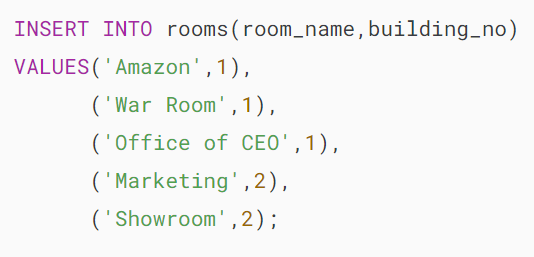
**Step 4**. Query data from the buildings table:



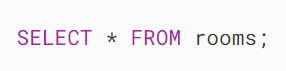
MySQL ON DELETE CASCADE buildings table

We have two rows in the buildings table.

**Step 5**. [Insert rows](https://www.mysqltutorial.org/mysql-basics/mysql-insert-multiple-rows/) into the rooms table:



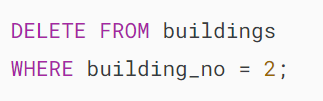
**Step 6**. Query data from the rooms table:



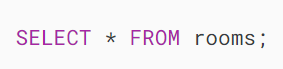


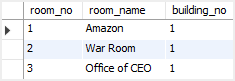
We have three rooms that belong to building no 1 and two rooms that belong to the building no 2.

**Step 7**. [Delete](https://www.mysqltutorial.org/mysql-basics/mysql-delete/) the building with building no. 2:



**Step 8**. Query data from rooms table:





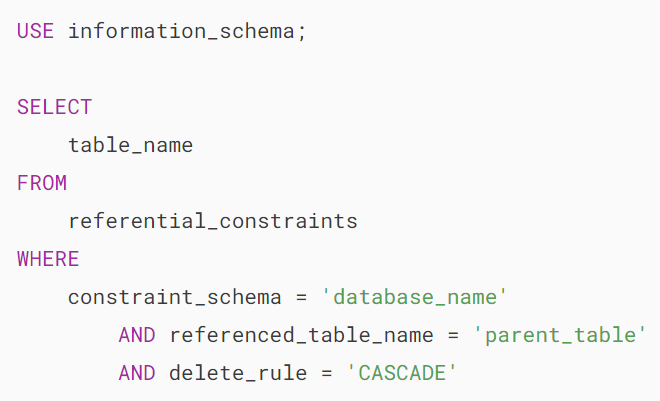
As you can see, all the rows that reference to building\_no 2 were automatically deleted.

Notice that ON DELETE CASCADE  works only with tables with the [storage engines](https://www.mysqltutorial.org/mysql-administration/mysql-storage-engines/) that support foreign keys e.g., InnoDB.

Some table types do not support foreign keys such as MyISAM so you should choose appropriate storage engines for the tables that you plan to use the MySQL ON DELETE CASCADE  referential action.

## **Tips to find tables affected by MySQL ON DELETE CASCADE action**

Sometimes, it is useful to know which table is affected by the ON DELETE CASCADE  referential action when you delete data from a table. You can query this data from the referential\_constraints in the information\_schema  database as follows:



For example, to find tables that associated with the buildings table with the CASCADE  deletion rule  in the classicmodels database, you use the following query:



MySQL ON DELETE CASCADE tips

In this tutorial, you have learned how to use the MySQL ON DELETE CASCADE  referential action for a foreign key to delete data automatically from the child tables when you delete data from the parent table.

# **MySQL TRUNCATE TABLE**

**Summary**: in this tutorial, you will learn how to use the MySQL TRUNCATE TABLE statement to delete all data in a table.

## **Introduction to the MySQL TRUNCATE TABLE statement**

In MySQL, the TRUNCATE TABLE statement allows you to delete all rows from a table.

Here’s the basic syntax of the TRUNCATE TABLE statement:



In this syntax, you specify the name of the table from which you want to delete all rows after the TRUNCATE TABLE keywords.

The TABLE keyword is optional. However, it is a good practice to use the TABLE keyword to distinguish between the TRUNCATE TABLE statement and the [TRUNCATE()](https://www.mysqltutorial.org/mysql-math-functions/mysql-truncate/) function.

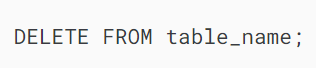
If there is any [FOREIGN KEY](https://www.mysqltutorial.org/mysql-basics/mysql-foreign-key/) constraints from other tables that reference the table that you truncate, the TRUNCATE TABLE statement will fail.

Since a truncate operation causes an implicit commit, it cannot be rolled back. Also, the TRUNCATE TABLE statement does not fire DELETE triggers associated with the table that is being truncated.

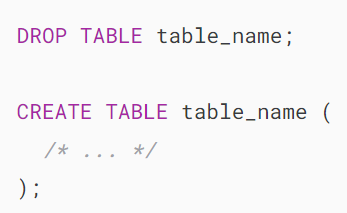
The TRUNCATE TABLE statement [resets the value in the AUTO\_INCREMENT column](https://www.mysqltutorial.org/mysql-basics/mysql-reset-auto-increment-value/)to its initial value if the table has an AUTO\_INCREMENT column.

Unlike a DELETE statement, the number of rows affected by the TRUNCATE TABLE statement is 0, which should be interpreted as no information.

Functionally, the TRUNCATE TABLE statement is like a [DELETE](https://www.mysqltutorial.org/mysql-basics/mysql-delete/) statement without a [WHERE](https://www.mysqltutorial.org/mysql-basics/mysql-where/) clause that deletes all rows from a table:



Or a sequence of [DROP TABLE](https://www.mysqltutorial.org/mysql-drop-table) and [CREATE TABLE](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) statements:

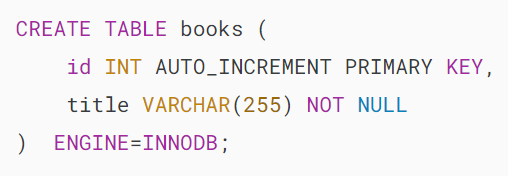


However, the TRUNCATE TABLE statement is more efficient than the DELETE statement because it drops and recreates the table instead of deleting rows one by one.

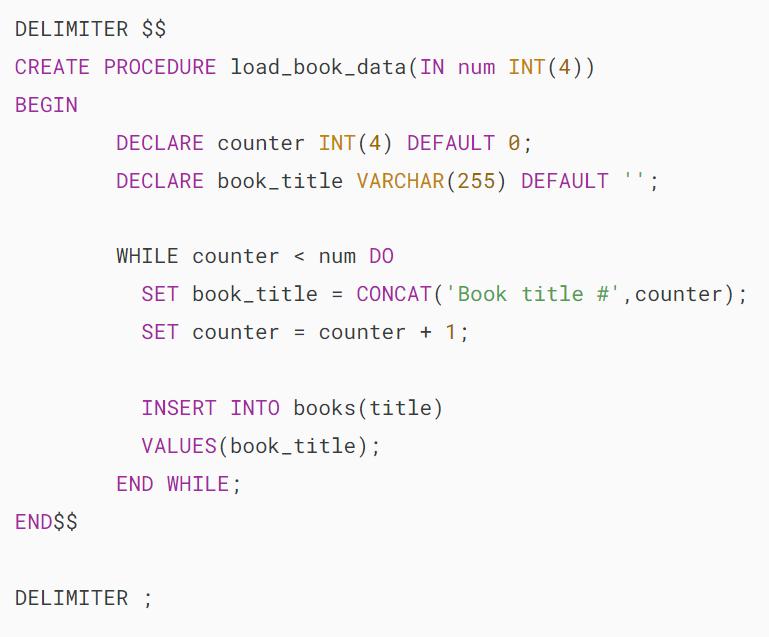
## **MySQL TRUNCATE TABLE example**

Let’s take an example of using the TRUNCATE TABLE statement.

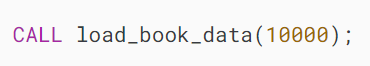
First, [create a new table](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) named books for the demonstration:



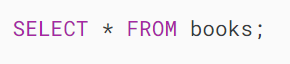
Next, insert dummy data into the books table by using the following [stored procedure](https://www.mysqltutorial.org/mysql-stored-procedure/):



Then, load 10,000 rows into the books table. It will take a while.



After that, check the data in the books table:



Finally, use the TRUNCATE TABLE statement to delete all rows from the books table:



Note that you can compare the performance of the TRUNCATE TABLE with the DELETE statement.

## **Summary**

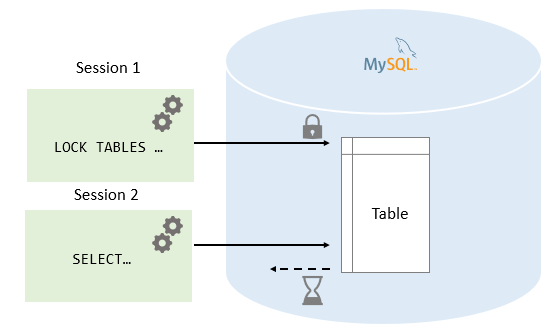
* Use the TRUNCATE TABLE statement to delete all rows from a table efficiently.
* The TRUNCATE TABLE statement resets the AUTO\_INCREMENT counter.

# **MySQL Table Locking**

**Summary**: in this tutorial, you will learn how to use MySQL locking for cooperating table accesses between sessions.

A lock is a flag associated with a table. MySQL allows a client session to explicitly acquire a table lock to prevent other sessions from accessing the same table during a specific period.

A client session can acquire or release table locks only for itself. A client session cannot acquire or release table locks for other client sessions.

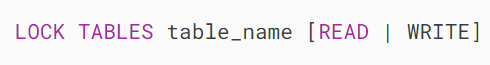


Before we move on, let’s [create a table](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) named messages for practicing with the table locking statements.

## 

## **MySQL LOCK TABLES statement**

The following LOCK TABLES statement explicitly acquires a table lock:



In this syntax, you specify the name of the table that you want to lock after the LOCK TABLES keywords. In addition, you specify the type of lock, either  READ or WRITE.

MySQL allows you to lock multiple tables by specifying a list of comma-separated table names with lock types that you want to lock after the LOCK TABLES keywords:

## 

## **MySQL UNLOCK TABLES statement**

The UNLOCK TABLES statement releases any table locks held by the current session:

## 

## **READ Locks**

A READ lock has the following features:

* A READ lock for a table can be acquired by multiple sessions at the same time. In addition, other sessions can read data from the table without acquiring the lock.
* The session that holds the READ lock can only read data from the table, but cannot write. And other sessions cannot write data to the table until the READ lock is released. The write operations from another session will be put into the waiting states until the READ lock is released.
* If the session is terminated, either normally or abnormally, MySQL will release all the locks implicitly. This feature is also relevant to the WRITE lock.

Let’s take a look at how the READ lock works in the following scenario.

First, connect to the database in the first session and use the CONNECTION\_ID() function to get the current connection id as follows:

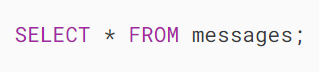




Then, [insert a new row](https://www.mysqltutorial.org/mysql-basics/mysql-insert/) into the messages table.



Next, [query the data](https://www.mysqltutorial.org/mysql-basics/mysql-select-from/) from the messages table.





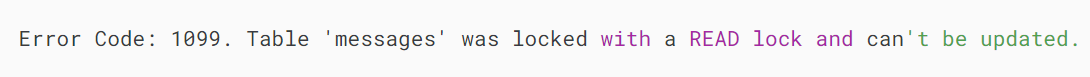
After that, acquire a lock using the LOCK TABLE statement.



Finally, try to insert a new row into the messages table:



MySQL issued the following error:



So once the READ lock is acquired, you cannot write data to the table within the same session.

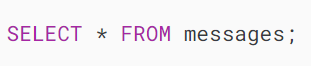
Let’s check the READ lock from a different session.

First, connect to the database and check the connection id:





Next, query data from the messages  table:





Then, [insert a new row](https://www.mysqltutorial.org/mysql-basics/mysql-insert/) into the messages table:

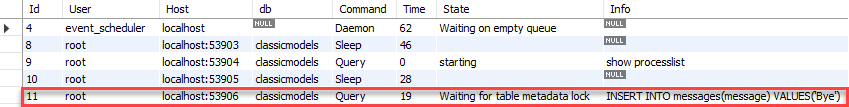
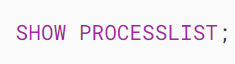


Here is the output:



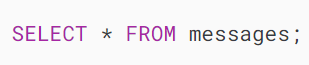
The insert operation from the second session is in the waiting state because a READ lock is already acquired on the messages table by the first session and it has not been released yet.

From the first session, use the [SHOW PROCESSLIST](https://www.mysqltutorial.org/mysql-administration/mysql-show-processlist/) statement to show detailed information:



After that, go back to the first session and release the lock by using the UNLOCK TABLES statement. After you release the READ lock from the first session, the INSERT operation in the second session is executed.

Finally, check the data of the messages table to see if the INSERT operation from the second session was executed.





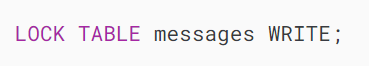
## **Write Locks**

A WRITE lock has the following features:

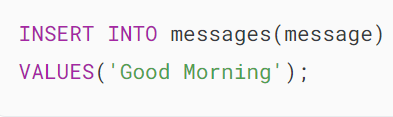
* The only session that holds the lock of a table can read and write data from the table.
* Other sessions cannot read data from and write data to the table until the WRITE lock is released.

Let’s go into detail to see how the WRITE lock works.

First, acquire a WRITE lock from the first session.



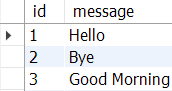
Then, insert a new row into the messages table.



It worked.

Next, query data from the messages table.



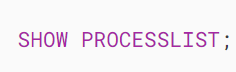


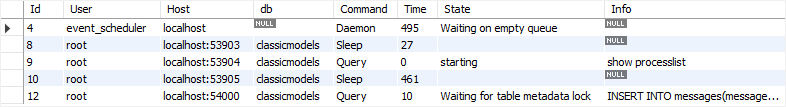
It also works.

After that, from the second session, attempt to write and read data:



MySQL puts these operations into a waiting state. You can check it using the SHOW PROCESSLIST statement:

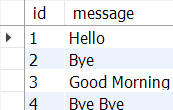




Finally, release the lock from the first session.



You will see all pending operations from the second session executed and the following picture illustrates the result:



## **Read vs. Write locks**

* Read locks are “shared” locks that prevent a write lock is being acquired but not other read locks.
* Write locks are “exclusive ” locks that prevent any other lock of any kind.

In this tutorial, you have learned how to lock and unlock tables to cooperate with the table accesses between sessions.