# MySQL Unique Key

A unique key in MySQL is a single field or combination of fields that ensure all values going to store into the column will be unique. It means a column cannot stores **duplicate values**. For example, the email addresses and roll numbers of students in the "student\_info" table or contact number of employees in the "Employee" table should be unique.

MySQL allows us to use more than one column with UNIQUE constraint in a table. It can accept a **null** value, but MySQL allowed only one null value per column. It ensures the **integrity** of the column or group of columns to store different values into a table.

### Needs of Unique Key

* It is useful in preventing the two records from storing identical values into the column.
* It stores only distinct values that maintain the integrity and reliability of the database for accessing the information in an organized way.
* It also works with a foreign key in preserving the uniqueness of a table.
* It can contain null value into the table.

### Syntax

The following syntax is used to create a unique key in MySQL.

If we want to create only one unique key column into a table, use the syntax as below:

1. **CREATE** **TABLE** table\_name(
2. col1 datatype,
3. col2 datatype **UNIQUE**,
4. ...
5. );

If we want to create more than one unique key column into a table, use the syntax as below:

1. **CREATE** **TABLE** table\_name(
2. col1 col\_definition,
3. col2 col\_definition,
4. ...
5. [**CONSTRAINT** constraint\_name]
6. **UNIQUE**(column\_name(s))
7. );

If we have not specified the name for a unique constraint, MySQL generates a name for this column automatically. So, it is recommended to use the constraint name while creating a table.

### Parameter Explanation

The following table explains the parameters in detail.

|  |  |
| --- | --- |
| **Parameter Name** | **Descriptions** |
| table\_name | It is the name of the table that we are going to create. |
| col1, col2 | It is the column names that contain in the table. |
| constraint\_name | It is the name of the unique key. |
| column\_name(s) | It is the column name(s) that is going to be a unique key. |

### Unique Key Example

The following example explains how a unique key used in MySQL.

This statement creates a table "**Student2**" with a UNIQUE constraint:

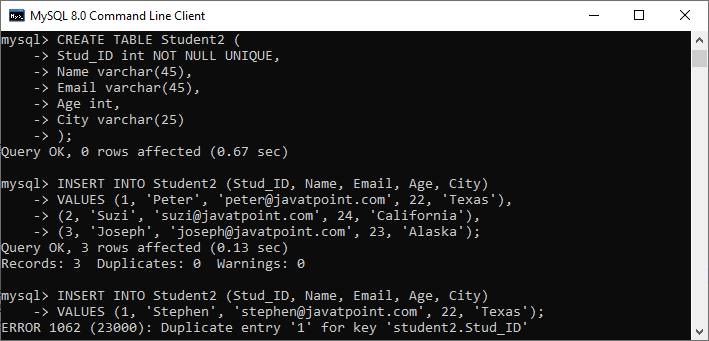
1. **CREATE** **TABLE** Student2 (
2. Stud\_ID **int** NOT NULL **UNIQUE**,
3. **Name** **varchar**(45),
4. Email **varchar**(45),
5. Age **int**,
6. City **varchar**(25)
7. );

Next, execute the insert queries listed below to understand how it works:

1. mysql> **INSERT** **INTO** Student2 (Stud\_ID, **Name**, Email, Age, City)
2. **VALUES** (1, 'Peter', 'peter@javatpoint.com', 22, 'Texas'),
3. (2, 'Suzi', 'suzi@javatpoint.com', 24, 'California'),
4. (3, 'Joseph', 'joseph@javatpoint.com', 23, 'Alaska');
6. mysql> **INSERT** **INTO** Student2 (Stud\_ID, **Name**, Email, Age, City)
7. **VALUES** (1, 'Stephen', 'stephen@javatpoint.com', 22, 'Texas');

**Output**

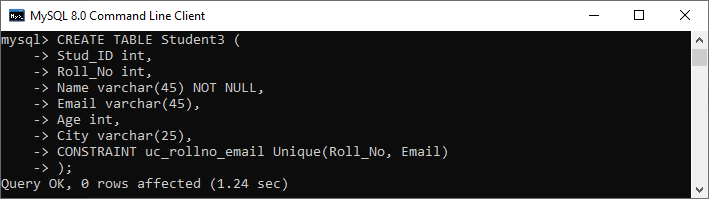
In the below output, we can see that the first [INSERT query](https://www.javatpoint.com/mysql-insert) executes correctly, but the second statement fails and gives an error that says: Duplicate entry '1' for key Stud\_ID.



If you want to define the unique key on **multiple columns**, use the query as below:

1. **CREATE** **TABLE** Student3 (
2. Stud\_ID **int**,
3. Roll\_No **int**,
4. **Name** **varchar**(45) NOT NULL,
5. Email **varchar**(45),
6. Age **int**,
7. City **varchar**(25),
8. **CONSTRAINT** uc\_rollno\_email **Unique**(Roll\_No, Email)
9. );

In the output, we can see that the unique key value contains two columns that are **Roll\_No** and **Email**.



To verify this, execute the following statement:

1. mysql> SHOW **INDEX** **FROM** Student3;

Here, we can see that the unique constraint has successfully added into the table:

IMG_258

### DROP Unique Key

The ALTER TABLE statement also allows us to drop the unique key from the table. The following syntax is used to drop the unique key:

1. **ALTER** **TABLE** table\_name  **DROP** **INDEX** constraint\_name;

In the above syntax, the **table\_name** is the name of the table that we want to modify, and **constraint\_name** is the name of the unique key we are going to remove.

**Example**

This statement will remove the **uc\_rollno\_email** constraint from the table permanently.

1. mysql> **ALTER** **TABLE** Student3 **DROP** **INDEX** uc\_rollno\_email;

We can execute the SHOW INDEX statement to very this.

### Unique Key Using ALTER TABLE Statement

This statement allows us to do the modification into the existing table. Sometimes we want to add a unique key to the column of an existing table; then, this statement is used to add the unique key for that column.

**Syntax**

Following are the syntax of the [ALTER TABLE statement](https://www.javatpoint.com/mysql-alter-table) to add a unique key:

1. **ALTER** **TABLE** table\_name **ADD** **CONSTRAINT** constraint\_name **UNIQUE**(column\_list);

**Example**

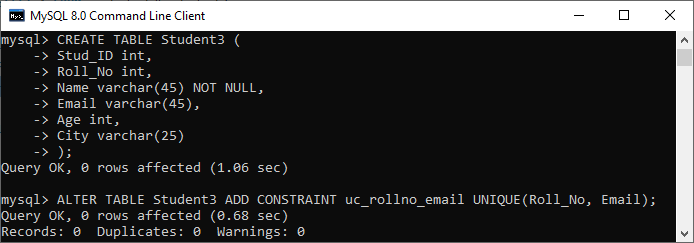
This statement creates a table "**Students3**" that have no unique key column into the table definition.

1. **CREATE** **TABLE** Student3 (
2. Stud\_ID **int**,
3. Roll\_No **int**,
4. **Name** **varchar**(45) NOT NULL,
5. Email **varchar**(45),
6. Age **int**,
7. City **varchar**(25)
8. );

After creating a table, if we want to add a unique key to this table, we need to execute the ALTER TABLE statement as below:

1. mysql> **ALTER** **TABLE** Student3 **ADD** **CONSTRAINT** uc\_rollno\_email **UNIQUE**(Roll\_No, Email);

We can see the output where both statements executed successfully.



To verify this, execute the following statement:

1. mysql> SHOW **INDEX** **FROM** Student3;

Here, we can see that the unique constraint has successfully added into the table:

IMG_260

# MySQL Primary Key

MySQL primary key is a single or combination of the field, which is used to identify each record in a table **uniquely**. If the column contains primary key constraints, then it cannot be **null or empty**. A table may have duplicate columns, but it can contain only one primary key. It always contains unique value into a column.

When you insert a new row into the table, the primary key column can also use the **AUTO\_INCREMENT** attribute to generate a sequential number for that row automatically. [MySQL](https://www.javatpoint.com/mysql-tutorial) automatically creates an index named "**Primary**" after defining a primary key into the table. Since it has an associated index, we can say that the primary key makes the query performance fast.

### Rules for Primary key

Following are the rules for the primary key:

1. The primary key column value must be unique.
2. Each table can contain only one primary key.
3. The primary key column cannot be null or empty.
4. MySQL does not allow us to insert a new row with the existing primary key.
5. It is recommended to use INT or BIGINT data type for the primary key column.

We can create a primary key in two ways:

* CREATE TABLE Statement
* ALTER TABLE Statement

Let us discuss each one in detail.

### Primary Key Using CREATE TABLE Statement

In this section, we are going to see how a primary key is created using the [CREATE TABLE](https://www.javatpoint.com/mysql-create-table) statement.

**Syntax**

The following are the syntax used to create a primary key in MySQL.

If we want to create only one primary key column into the table, use the below syntax:

1. **CREATE** **TABLE** table\_name(
2. col1 datatype **PRIMARY** **KEY**,
3. col2 datatype,
4. ...
5. );

If we want to create more than one primary key column into the table, use the below syntax:

1. **CREATE** **TABLE** table\_name
2. (
3. col1 col\_definition,
4. col2 col\_definition,
5. ...
7. **CONSTRAINT** [constraint\_name]
8. **PRIMARY** **KEY** (column\_name(s))
9. );

### Parameter Explanation

The following table explains the parameters in detail.

|  |  |
| --- | --- |
| **Parameter Name** | **Descriptions** |
| Table\_name | It is the name of the table that we are going to create. |
| Col1, col2 | It is the column names that contain in the table. |
| Constraint\_name | It is the name of the primary key. |
| Column\_name(s) | It is the column name(s) that is going to be a primary key. |

### Primary Key Example

The following example explains how a primary key used in MySQL.

This statement creates a table named "**Login**" whose "**login\_id**" column contains the primary key:

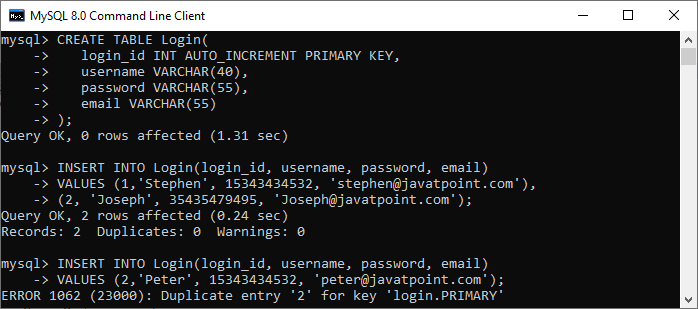
1. Mysql> **CREATE** **TABLE** Login(
2. login\_id **INT** AUTO\_INCREMENT **PRIMARY** **KEY**,
3. username **VARCHAR**(40),
4. **password** **VARCHAR**(55),
5. email **VARCHAR**(55)
6. );

Next, use the insert query to store data into a table:

1. mysql> **INSERT** **INTO** Login(login\_id, username, **password**, email)
2. **VALUES** (1,'Stephen', 15343434532, 'stephen@javatpoint.com'),
3. (2, 'Joseph', 35435479495, 'Joseph@javatpoint.com');
5. mysql> **INSERT** **INTO** Login(login\_id, username, **password**, email)
6. **VALUES** (1,'Peter', 15343434532, 'peter@javatpoint.com');

**Output**

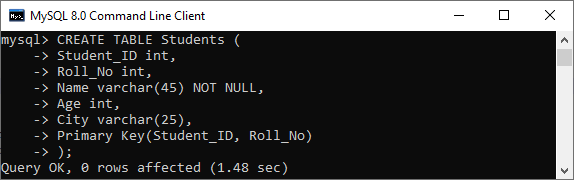
In the below output, we can see that the first insert query executes successfully. While the second insert statement fails and gives an error that says: Duplicate entry for the primary key column.



If you want to define the primary key on **multiple columns**, use the query as below:

1. mysql> **CREATE** **TABLE** Students (
2. Student\_ID **int**,
3. Roll\_No **int**,
4. **Name** **varchar**(45) NOT NULL,
5. Age **int**,
6. City **varchar**(25),
7. **Primary** **Key**(Student\_ID, Roll\_No)
8. );

In the output, we can see that the primary key value contains two columns that are **Student\_ID** and **Roll\_No**.



### Primary Key Using ALTER TABLE Statement

This statement allows us to do the modification into the existing table. When the table does not have a primary key, this statement is used to add the primary key to the column of an existing table.

**Syntax**

Following are the syntax of the ALTER TABLE statement to create a primary key in MySQL:

1. **ALTER** **TABLE** table\_name **ADD** **PRIMARY** **KEY**(column\_list);

### Example

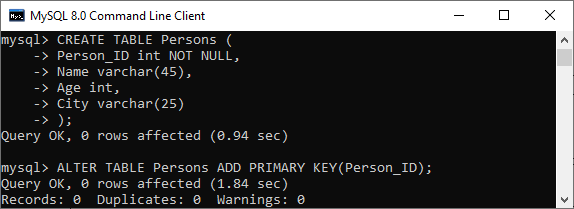
The following statement creates a table "**Persons**" that have no primary key column into the table definition.

1. mysql> **CREATE** **TABLE** Persons (
2. Person\_ID **int** NOT NULL,
3. **Name** **varchar**(45),
4. Age **int**,
5. City **varchar**(25)
6. );

After creating a table, if we want to add a primary key to this table, we need to execute the ALTER TABLE statement as below:

1. mysql> **ALTER** **TABLE** Persons **ADD** **PRIMARY** **KEY**(Person\_ID);

We can see the output where both statements executed successfully.



If the table needs to add the primary key into a table that already has data into the column, then it must be sure to the column does not contains duplicates or null values.

### DROP Primary Key

The ALTER TABLE statement also allows us to drop the primary key from the table. The following syntax is used to drop the primary key:

1. **ALTER** **TABLE** table\_name  **DROP** **PRIMARY** **KEY**;

**Example**

1. mysql> **ALTER** **TABLE** Login **DROP** **PRIMARY** **KEY**;

### Primary Key vs. Unique Key

The following comparison chart explains some of the common differences between both of them:

|  |  |  |
| --- | --- | --- |
| **SN** | **Primary Key** | **Unique Key** |
| **1.** | It is a single or combination of the field, which is used to identify each record in a table uniquely. | It also determines each row of the table uniquely in the absence of a primary key. |
| **2.** | It does not allow to store a NULL value into the primary key column. | It can accept only one NULL value into the unique key column. |
| **3.** | A table can have only one primary key. | A table can have more than one unique key. |
| **4.** | It creates a clustered index. | It creates a non-clustered index. |

# MySQL Foreign Key

The foreign key is used to link one or more than one table together. It is also known as the **referencing** key. A foreign key matches the primary key field of another table. It means a foreign key field in one table refers to the primary key field of the other table. It identifies each row of another table uniquely that maintains the **referential integrity** in MySQL.

A foreign key makes it possible to create a parent-child relationship with the tables. In this relationship, the parent table holds the initial column values, and column values of child table reference the parent column values. MySQL allows us to define a foreign key constraint on the child table.

[MySQL](https://www.javatpoint.com/mysql-tutorial) defines the foreign key in two ways:

1. Using CREATE TABLE Statement
2. Using ALTER TABLE Statement

### Syntax

Following are the basic syntax used for defining a foreign key using CREATE TABLE OR ALTER TABLE statement in the MySQL:

1. [**CONSTRAINT** constraint\_name]
2. **FOREIGN** **KEY** [foreign\_key\_name] (col\_name, ...)
3. **REFERENCES** parent\_tbl\_name (col\_name,...)
4. **ON** **DELETE** referenceOption
5. **ON** **UPDATE** referenceOption

In the above syntax, we can see the following parameters:

**constraint\_name:** It specifies the name of the foreign key constraint. If we have not provided the constraint name, MySQL generates its name automatically.

**col\_name:** It is the names of the column that we are going to make foreign key.

**parent\_tbl\_name:** It specifies the name of a parent table followed by column names that reference the foreign key columns.

**Refrence\_option:** It is used to ensure how foreign key maintains referential integrity using ON DELETE and ON UPDATE clause between parent and child table.

MySQL contains **five** different referential options, which are given below:

**CASCADE:** It is used when we delete or update any row from the parent table, the values of the matching rows in the child table will be deleted or updated automatically.

**SET NULL:** It is used when we delete or update any row from the parent table, the values of the foreign key columns in the child table are set to NULL.

**RESTRICT:** It is used when we delete or update any row from the parent table that has a matching row in the reference(child) table, MySQL does not allow to delete or update rows in the parent table.

**NO ACTION:** It is similar to RESTRICT. But it has one difference that it checks referential integrity after trying to modify the table.

**SET DEFAULT:** The MySQL parser recognizes this action. However, the InnoDB and NDB tables both rejected this action.

#### NOTE: MySQL mainly provides full support to CASCADE, RESTRICT, and SET NULL actions. If we have not specified the ON DELETE and ON UPDATE clause, MySQL takes default action RESTRICT.

### Foreign Key Example

Let us understand how foreign key works in MySQL. So first, we are going to create a database named "**mysqltestdb**" and start using it with the command below:

1. mysql> **CREATE** **DATABASE** mysqltestdb;
2. mysql> use mysqltestdb;

Next, we need to create two tables named "**customer**" and "**contact**" using the below statement:

**Table: customer**

1. **CREATE** **TABLE** customer (
2. ID **INT** NOT NULL AUTO\_INCREMENT,
3. **Name** **varchar**(50) NOT NULL,
4. City **varchar**(50) NOT NULL,
5. **PRIMARY** **KEY** (ID)
6. );

**Table: contact**

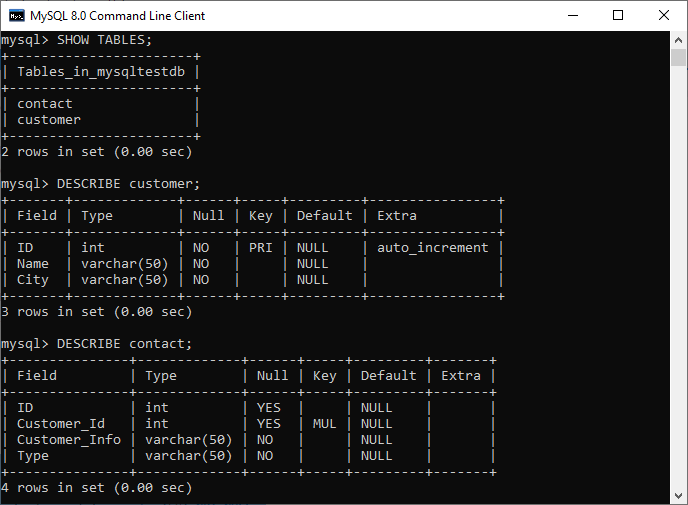
1. **CREATE** **TABLE** contact (
2. ID **INT**,
3. Customer\_Id **INT**,
4. Customer\_Info **varchar**(50) NOT NULL,
5. Type **varchar**(50) NOT NULL,
6. **INDEX** par\_ind (Customer\_Id),
7. **CONSTRAINT** fk\_customer **FOREIGN** **KEY** (Customer\_Id)
8. **REFERENCES** customer(ID)
9. **ON** **DELETE** **CASCADE**
10. **ON** **UPDATE** **CASCADE**
11. );

### Table Structure Verification

Here, we are going to see how our database structure looks like using the following queries:

1. mysql> SHOW TABLES;
2. mysql> DESCRIBE customer;
3. mysql> DESCRIBE contact;

We will get the structure as below:



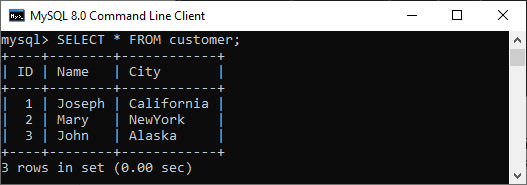
In the above output, we can see that the **PRI** in the key column of the customer table tells that this field is the primary index value. Next, the **MUL** in the key column of the contact value tells that the **Customer\_Id** field can store multiple rows with the same value.

### Insert Data to the Table

Now, we have to insert the records into both tables. Execute this statement to insert data into table customer:

1. **INSERT** **INTO** customer(**Name**, City) **VALUES**
2. ('Joseph', 'California'),
3. ('Mary', 'NewYork'),
4. ('John', 'Alaska');

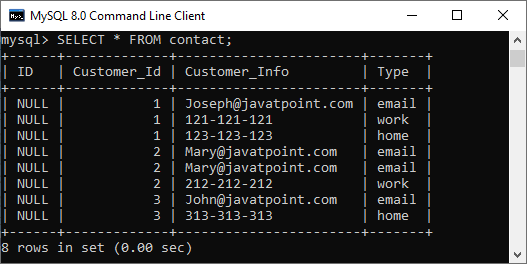
After insertion, execute the SELECT TABLE command to check the customer table data as below:



Execute the below insert statement to add data into a table contact:

1. **INSERT** **INTO** contact (Customer\_Id, Customer\_Info, Type) **VALUES**
2. (1, 'Joseph@javatpoint.com', 'email'),
3. (1, '121-121-121', 'work' ),
4. (1, '123-123-123', 'home'),
5. (2, 'Mary@javatpoint.com', 'email'),
6. (2, 'Mary@javatpoint.com', 'email'),
7. (2, '212-212-212', 'work'),
8. (3, 'John@javatpoint.com', 'email'),
9. (3, '313-313-313', 'home');

Our contact table looks like as below:



Now, let's see how foreign keys in MySQL preserve data integrity.

So here, we are going to delete the referential data that removes records from both tables. We have defined the foreign key in the contact table as:

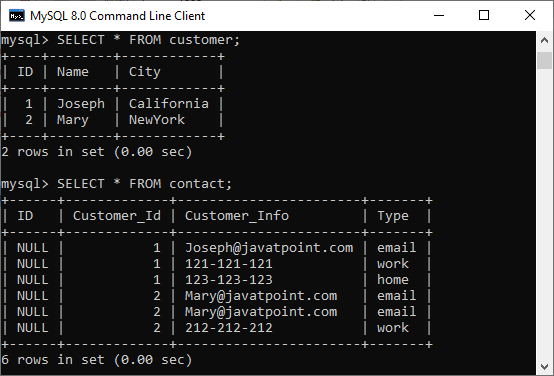
1. **FOREIGN** **KEY** (Customer\_Id) **REFERENCES** customer(ID)
2. **ON** **DELETE** **CASCADE**
3. **ON** **UPDATE** **CASCADE**.

It means if we delete any customer record from the customer table, then the related records in the contact table should also be deleted. And the ON UPDATE CASCADE will updates automatically on the parent table to referenced fields in the child table(Here, it is Customer\_Id).

Execute this statement that deletes a record from the table whose name is **JOHN**.

1. mysql> **DELETE** **FROM** customer **WHERE** **Name**='John';

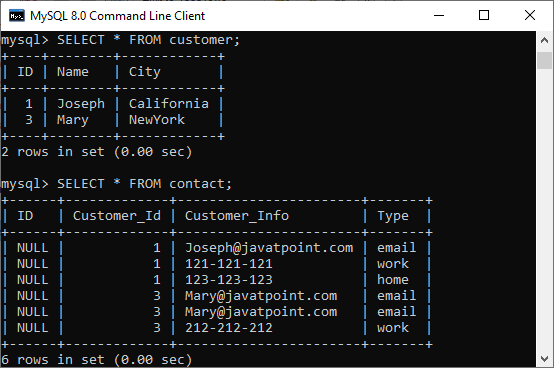
Again, if we look at our tables, we can see that both tables were changed. It means the fields with name JOHN will be removed entirely from both tables.



Now, test the **ON UPDATE CASCADE**. Here, we are going to update the Customer\_Id of **Mary** in the contact table as:

1. mysql> **UPDATE** customer **SET** id=3 **WHERE** **Name**='Mary';

Again, if we look at our tables, we can see that both tables were changed with Customer\_Id of Mary=3.



### Foreign Key example using SET NULL action

Here, we are going to understand how the SET NULL action works with a foreign key. First, we have to create two table named **Persons** and **Contacts**, as shown below:

**Table: Persons**

1. **CREATE** **TABLE** Persons (
2. ID **INT** NOT NULL AUTO\_INCREMENT,
3. **Name** **varchar**(50) NOT NULL,
4. City **varchar**(50) NOT NULL,
5. **PRIMARY** **KEY** (ID)
6. );

**Table: Customers**

1. **CREATE** **TABLE** Contacts (
2. ID **INT**,
3. Person\_Id **INT**,
4. Info **varchar**(50) NOT NULL,
5. Type **varchar**(50) NOT NULL,
6. **INDEX** par\_ind (Person\_Id),
7. **CONSTRAINT** fk\_person **FOREIGN** **KEY** (Person\_Id)
8. **REFERENCES** Persons(ID)
9. **ON** **DELETE** **SET** NULL
10. **ON** **UPDATE** **SET** NULL
11. );

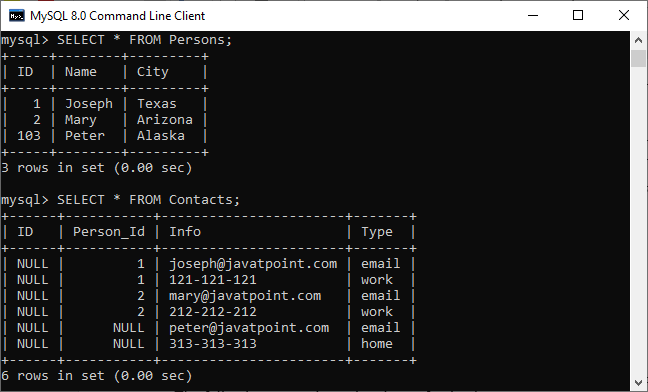
Next, we need to insert the data into both tables using the following statement:

1. **INSERT** **INTO** Persons(**Name**, City) **VALUES**
2. ('Joseph', 'Texas'),
3. ('Mary', 'Arizona'),
4. ('Peter', 'Alaska');
5. **INSERT** **INTO** Contacts (Person\_Id, Info, Type) **VALUES**
6. (1, 'joseph@javatpoint.com', 'email'),
7. (1, '121-121-121', 'work' ),
8. (2, 'mary@javatpoint.com', 'email'),
9. (2, '212-212-212', 'work'),
10. (3, 'peter@javatpoint.com', 'email'),
11. (3, '313-313-313', 'home');

Now, update the ID of the "Persons" table:

1. mysql> **UPDATE** Persons **SET** ID=103 **WHERE** ID=3;

Finally, verify the update using the SELECT statement given below:



If we look at our tables, we can see that both tables were changed. The rows with a **Person\_Id=3** in the Contacts table automatically set to **NULL** due to the ON UPDATE SET NULL action.

### How to DROP Foreign Key

MySQL allows the ALTER TABLE statement to remove an existing foreign key from the table. The following syntax is used to drop a foreign key:

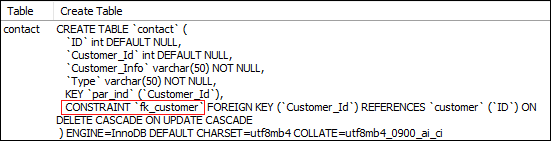
1. **ALTER** **TABLE** table\_name **DROP** **FOREIGN** **KEY** fk\_constraint\_name;

Here, the **table\_name** is the name of a table from where we are going to remove the foreign key. The **constraint\_name** is the name of the foreign key that was added during the creation of a table.

If we have not known the name of an existing foreign key into the table, execute the following command:

1. mysql> SHOW **CREATE** **TABLE** contact;

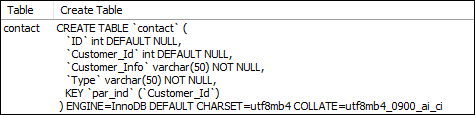
It will give the output as below where we can see that the table contact has one foreign key named fk\_customer shown in the red rectangle.



Now, to delete this foreign key constraint from the contact table, execute the statement as below:

1. mysql> **ALTER** **TABLE** contact **DROP** **FOREIGN** **KEY** fk\_customer;

We can verify whether foreign key constraint removes or not, use the SHOW CREATE TABLE statement. It will give the output as below where we can see that the foreign key is no longer available in the table contact.



### Define Foreign Key Using ALTER TABLE Statement

This statement allows us to do the modification into the existing table. Sometimes there is a need to add a foreign key to the column of an existing table; then, this statement is used to add the foreign key for that column.

**Syntax**

Following are the syntax of the ALTER TABLE statement to add a foreign key in the existing table:

1. **ALTER** **TABLE** table\_name
2. **ADD** [**CONSTRAINT** [symbol]] **FOREIGN** **KEY**
3. [index\_name] (column\_name, ...)
4. **REFERENCES** table\_name (column\_name,...)
5. **ON** **DELETE** referenceOption
6. **ON** **UPDATE** referenceOption

When we add a foreign key using the ALTER TABLE statement, it is recommended to first create an **index** on the column(s), which is referenced by the foreign key.

**Example**

The following statement creates two tables, "**Person**" and "**Contact**", without having a foreign key column into the table definition.

**Table: Person**

1. **CREATE** **TABLE** Person (
2. ID **INT** NOT NULL AUTO\_INCREMENT,
3. **Name** **varchar**(50) NOT NULL,
4. City **varchar**(50) NOT NULL,
5. **PRIMARY** **KEY** (ID)
6. );

**Table: Contact**

1. **CREATE** **TABLE** Contact (
2. ID **INT**,
3. Person\_Id **INT**,
4. Info **varchar**(50) NOT NULL,
5. Type **varchar**(50) NOT NULL
6. );

After creating a table, if we want to add a foreign key to an existing table, we need to execute the ALTER TABLE statement as below:

1. **ALTER** **TABLE** Contact **ADD** **INDEX** par\_ind ( Person\_Id );
2. **ALTER** **TABLE** Contact **ADD** **CONSTRAINT** fk\_person
3. **FOREIGN** **KEY** ( Person\_Id ) **REFERENCES** Person ( ID ) **ON** **DELETE** **CASCADE** **ON** **UPDATE** **RESTRICT**;

### Foreign Key Checks

MySQL has a special variable **foreign\_key\_cheks** to control the foreign key checking into the tables. By default, it is enabled to enforce the referential integrity during the normal operation on the tables. This variable is dynamic in nature so that it supports global and session scopes both.

Sometimes there is a need for disabling the foreign key checking, which is very useful when:

* We drop a table that is a reference by the foreign key.
* We import data from a CSV file into a table. It speeds up the import operation.
* We use ALTER TABLE statement on that table which has a foreign key.
* We can execute load data operation into a table in any order to avoid foreign key checking.

The following statement allows us to **disable** foreign key checks:

1. **SET** foreign\_key\_checks = 0;

The following statement allows us to **enable** foreign key checks:

1. **SET** foreign\_key\_checks = 1;

# MySQL Composite Key

A composite key in MySQL is a combination of two or more than two columns in a table that allows us to identify each row of the table uniquely. It is a type of **candidate key** which is formed by more than one column. MySQL guaranteed the uniqueness of the column only when they are combined. If they have taken individually, the uniqueness cannot maintain.

Any key such as primary key, super key, or candidate key can be called composite key when they have combined with more than one attribute. A composite key is useful when the table needs to identify each record with more than one attribute uniquely. A column used in the composite key can have different data types. Thus, it is not required to be the same data type for the columns to make a composite key in [MySQL](https://www.javatpoint.com/mysql-tutorial).

A composite key can be added in two ways:

1. Using CREATE Statement
2. Using ALTER Statement

Let us see both ways in detail.

### Composite Key Using CREATE Statement

Here, we are going to understand how composite key works in MySQL. Let us first create a table **"Product"**, using the following statement:

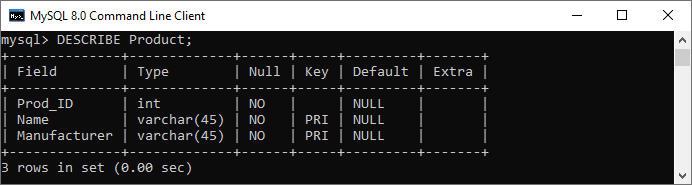
1. **CREATE** **TABLE** Product (
2. Prod\_ID **int** NOT NULL,
3. **Name** **varchar**(45),
4. Manufacturer **varchar**(45),
5. **PRIMARY** **KEY**(**Name**, Manufacturer)
6. );

In the above statement, we have created a composite primary with the column names **Name** and **Manufacturer**.

We can verify the same using the command as below:

1. DESCRIBE Product;

After the successful execution, we can see that the Key column has two **PRI**. It means we have successfully added the composite primary key on Name and Manufacturer columns.



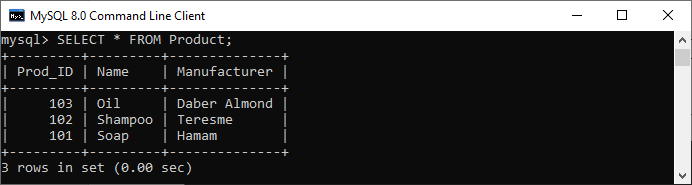
Next, we need to insert the values into this table as given below:

1. **INSERT** **INTO** Product (Prod\_ID, **Name**, Manufacturer)
2. **VALUES** (101, 'Soap', 'Hamam'),
3. (102, 'Shampoo', 'Teresme'),
4. (103, 'Oil', 'Daber Almond');

Next, execute the below command to show the table data:

1. **SELECT** \* **FROM** Product;

It will give the output below:

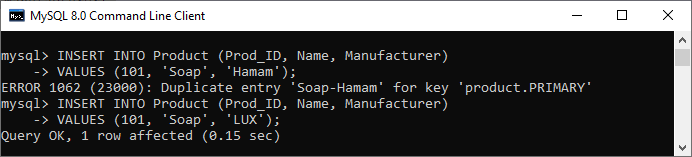


Again execute the below insert statement to understand composite key more clearly:

1. **INSERT** **INTO** Product (Prod\_ID, **Name**, Manufacturer)
2. **VALUES** (101, 'Soap', 'Hamam');
4. **INSERT** **INTO** Product (Prod\_ID, **Name**, Manufacturer)
5. **VALUES** (101, 'Soap', 'LUX');

In the below output, we can see that if we try to add the combination of the same product name and manufacturer, then it will throw an error saying that: **Duplicate entry for product.primary**.

If we execute the second insert statement, it will be added successfully into the table. It is because we can insert any number of soap in the product column, but the manufacturer column should be different.



Hence, we can say that the composite key always enforces the uniqueness of the columns of that table, which has two keys.

### Composite Key Using ALTER TABLE Statement

ALTER statement always used to do the modification into the existing table. Sometimes it is required to add the composite key to uniquely identify each record of the table with more than one attribute. In that case, we use an ALTER TABLE statement.

Let us first create a table "Student" using the below statement:

1. **CREATE** **TABLE** Student(
2. stud\_id **int** NOT NULL,
3. stud\_code **varchar**(15),
4. stud\_name **varchar**(35),
5. subject **varchar**(25),
6. marks **int**
7. );

Now, execute the ALTER TABLE statement to add a composite primary key as follows:

1. **ALTER** **TABLE** Student **add** **primary** **key**(stud\_id, subject);

We can verify the composite primary key added into a table or not using the following command:

1. DESCRIBE Student;

In the output, we can see that the key column has PRI, which means we have successfully added the composite primary key to **stud\_id** and **subject** columns.

