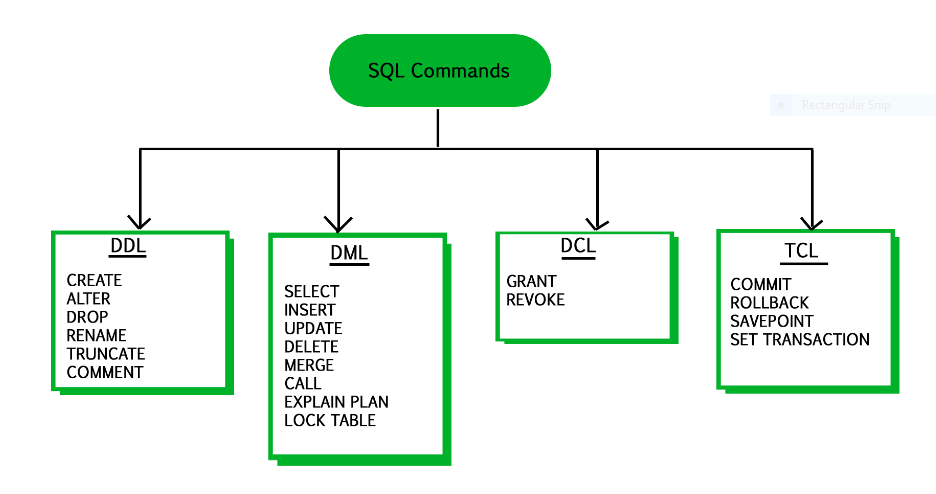
show databases;

use mysql;

select \* from user;

create user uzma@'%' identified by 'cdac';



If a table has data redundancy and is not properly normalized, then it will be difficult to handle and update the database, without facing data loss. It will also eat up extra memory space and Insertion, Update and Deletion *Anomalies are very frequent if database is not normalized*.

[Normalization](https://www.geeksforgeeks.org/database-normalization-normal-forms/) is the process of minimizing redundancy from a relation or set of relations. Redundancy in relation may cause insertion, deletion and update anomalies. So, it helps to minimize the redundancy in relations. Normal forms are used to eliminate or reduce redundancy in database tables.

There are various level of normalization. These are some of them:

**1.** First Normal Form (1NF)

**2.** Second Normal Form (2NF)

**3.** Third Normal Form (3NF)

**4.** Boyce-Codd Normal Form (BCNF)

**5.** Fourth Normal Form (4NF)

**6.** Fifth Normal Form (5NF)

In this article, we will discuss First Normal Form (1NF).

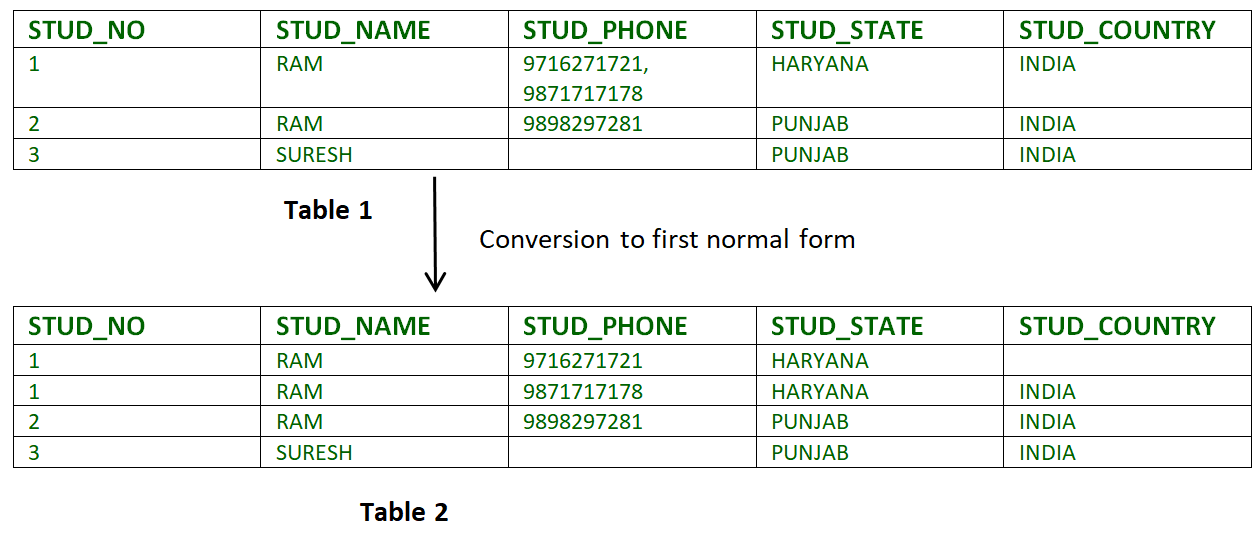
**First Normal Form (1NF):**   
If a relation contains a composite or multi-valued attribute, it violates the first normal form, or the relation is in first normal form if it does not contain any **composite** or **multi-valued attribute**. A relation is in first normal form if every attribute in that relation is singled valued attribute.

A table is in 1 NF iff:

1. There are only Single Valued Attributes.
2. Attribute Domain does not change.
3. There is a unique name for every Attribute/Column.
4. The order in which data is stored does not matter.

Consider the examples given below.

**Example-1:**   
Relation STUDENT in table 1 is not in 1NF because of multi-valued attribute STUD\_PHONE. Its decomposition into 1NF has been shown in table 2.



**Example-2:**

ID Name Courses

------------------

1 A c1, c2

2 E c3

3 M C2, c3

In the above table, Course is a m

**Difference between 1NF and 2NF :**

| **S.NO.** | **1NF** | **2NF** |
| --- | --- | --- |
| 1. | In order to be in 1NF any relation must be atomic and should not contain any composite or multi-valued attributes. | In order to be in 2NF any relation must be in 1NF and should not contain any partial dependency. |
| 2. | The identification of functional dependency is not necessary for first normal form. | The identification of functional dependency is necessary for second normal form. |
| 3. | First Normal form only deals with the schema of the table and it does not handle the update anomalies. | Second normal form handles the update anomalies. |
| 4. | A relation in 1NF may or may not be in 2NF. | A relation in 2NF is always in 1NF. |
| 5. | The primary key in case of first normal form can be a composite key. | The primary key in case of second normal form cannot be a composite key in case it arises any partial dependency. |
| 6. | The main goal of first normal form is to eliminate the redundant data within the table. | The main goal of second normal form is to actually ensure the data dependencies. |
| 7. | The first normal form is less stronger than the second normal form. | The second normal form is comparatively more strong than first normal form. |

**Second Normal Form (2NF):**  
Second Normal Form (2NF) is based on the concept of full functional dependency. Second Normal Form applies to relations with composite keys, that is, relations with a primary key composed of two or more attributes. A relation with a single-attribute primary key is automatically in at least 2NF. A relation that is not in 2NF may suffer from the update anomalies.

To be in second normal form, a relation must be in first normal form and relation must not contain any partial dependency. A relation is in 2NF if it has No Partial Dependency, i.e., no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.

In other words,

*A relation that is in First Normal Form and every non-primary-key attribute is fully functionally dependent on the primary key, then the relation is in Second Normal Form (2NF).*

**Note –** If the proper subset of candidate key determines non-prime attribute, it is called [partial dependency](https://practice.geeksforgeeks.org/problems/differentiate-between-full-functional-dependency-and-partial-dependency).

The [normalization](https://www.geeksforgeeks.org/database-normalization-normal-forms/) of 1NF relations to 2NF involves the **removal of partial dependencies**. If a partial dependency exists, we remove the partially dependent attribute(s) from the relation by placing them in a new relation along with a copy of their determinant.

Consider the examples given below.

**Example-1:**  
Consider table as following below.

STUD\_NO COURSE\_NO COURSE\_FEE

1 C1 1000

2 C2 1500

1 C4 2000

4 C3 1000

4 C1 1000

2 C5 2000

{Note that, there are many courses having the same course fee. }

Here,  
COURSE\_FEE cannot alone decide the value of COURSE\_NO or STUD\_NO;  
COURSE\_FEE together with STUD\_NO cannot decide the value of COURSE\_NO;  
COURSE\_FEE together with COURSE\_NO cannot decide the value of STUD\_NO;  
Hence,  
COURSE\_FEE would be a non-prime attribute, as it does not belong to the one only candidate key {STUD\_NO, COURSE\_NO} ;  
But, COURSE\_NO -> COURSE\_FEE, i.e., COURSE\_FEE is dependent on COURSE\_NO, which is a proper subset of the candidate key. Non-prime attribute COURSE\_FEE is dependent on a proper subset of the candidate key, which is a partial dependency and so this relation is not in 2NF.

To convert the above relation to 2NF,  
we need to split the table into two tables such as :  
Table 1: STUD\_NO, COURSE\_NO  
Table 2: COURSE\_NO, COURSE\_FEE

**Table 1** **Table 2**

STUD\_NO COURSE\_NO COURSE\_NO COURSE\_FEE

1 C1 C1 1000

2 C2 C2 1500

1 C4 C3 1000

4 C3 C4 2000

4 C1 C5 2000

2 C5

**Note –** 2NF tries to reduce the redundant data getting stored in memory. For instance, if there are 100 students taking C1 course, we dont need to store its Fee as 1000 for all the 100 records, instead once we can store it in the second table as the course fee for C1 is 1000.

**Example-2:**  
Consider following functional dependencies in relation  R (A,   B, C,   D )

AB -> C [A and B together determine C]

BC -> D [B and C together determine D]

In the above relation, AB is the only candidate key and there is no partial dependency, i.e., any proper subset of AB doesn’t determine any non-prime attribute.

|  |  |
| --- | --- |
| **DBMS** | **RDBMS** |
| 1) | DBMS applications store **data as file**. | RDBMS applications store **data in a tabular form**. |
| 2) | In DBMS, data is generally stored in either a hierarchical form or a navigational form. | In RDBMS, the tables have an identifier called primary key and the data values are stored in the form of tables. |
| 3) | **Normalization is not** present in DBMS. | **Normalization is** present in RDBMS. |
| 4) | DBMS does **not apply any security** with regards to data manipulation. | RDBMS **defines the integrity constraint** for the purpose of ACID (Atomocity, Consistency, Isolation and Durability) property. |
| 5) | DBMS uses file system to store data, so there will be **no relation between the tables**. | in RDBMS, data values are stored in the form of tables, so a **relationship** between these data values will be stored in the form of a table as well. |
| 6) | DBMS has to provide some uniform methods to access the stored information. | RDBMS system supports a tabular structure of the data and a relationship between them to access the stored information. |
| 7) | DBMS **does not support distributed database**. | RDBMS **supports distributed database**. |
| 8) | DBMS is meant to be for small organization and **deal with small data**. it supports **single user**. | RDBMS is designed to **handle large amount of data**. it supports **multiple users**. |
| 9) | Examples of DBMS are file systems, **xml** etc. | Example of RDBMS are **mysql**, **postgre**, **sql server**, **oracle** etc. |

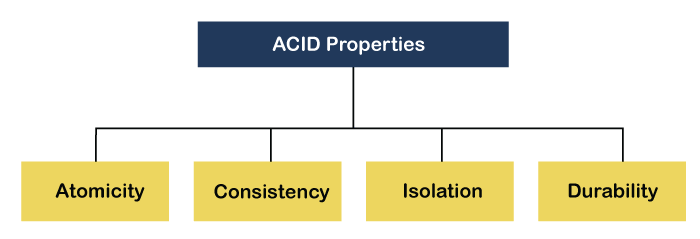
# ACID Properties in DBMS

DBMS is the management of data that should remain integrated when any changes are done in it. It is because if the integrity of the data is affected, whole data will get disturbed and corrupted. Therefore, to maintain the integrity of the data, there are four properties described in the database management system, which are known as the **ACID** properties. The ACID properties are meant for the transaction that goes through a different group of tasks, and there we come to see the role of the ACID properties.

In this section, we will learn and understand about the ACID properties. We will learn what these properties stand for and what does each property is used for. We will also understand the ACID properties with the help of some examples.

## ACID Properties

The expansion of the term ACID defines for:



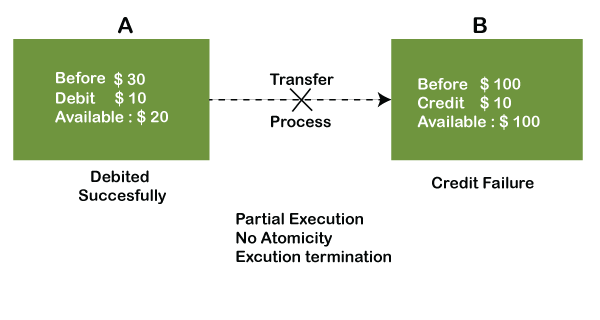
**1) Atomicity:** The term atomicity defines that the data remains atomic. It means if any operation is performed on the data, either it should be performed or executed completely or should not be executed at all. It further means that the operation should not break in between or execute partially. In the case of executing operations on the transaction, the operation should be completely executed and not partially.

62.3M

1.2K

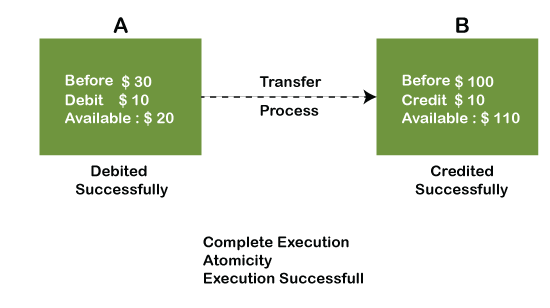
C++ vs Java

**Example:** If Remo has account A having $30 in his account from which he wishes to send $10 to Sheero's account, which is B. In account B, a sum of $ 100 is already present. When $10 will be transferred to account B, the sum will become $110. Now, there will be two operations that will take place. One is the amount of $10 that Remo wants to transfer will be debited from his account A, and the same amount will get credited to account B, i.e., into Sheero's account. Now, what happens - the first operation of debit executes successfully, but the credit operation, however, fails. Thus, in Remo's account A, the value becomes $20, and to that of Sheero's account, it remains $100 as it was previously present.



In the above diagram, it can be seen that after crediting $10, the amount is still $100 in account B. So, it is not an atomic transaction.

The below image shows that both debit and credit operations are done successfully. Thus the transaction is atomic.

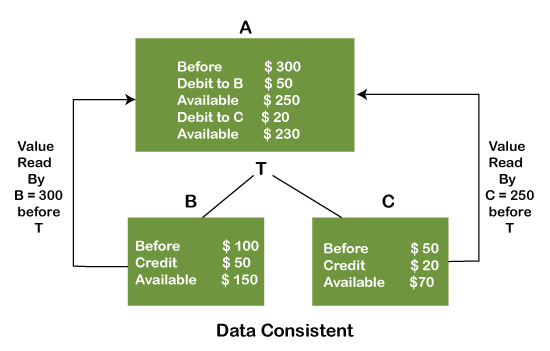


Thus, when the amount loses atomicity, then in the bank systems, this becomes a huge issue, and so the atomicity is the main focus in the bank systems.

**2) Consistency:** The word **consistency** means that the value should remain preserved always. In [DBMS](https://www.javatpoint.com/dbms-tutorial)

, the integrity of the data should be maintained, which means if a change in the database is made, it should remain preserved always. In the case of transactions, the integrity of the data is very essential so that the database remains consistent before and after the transaction. The data should always be correct.

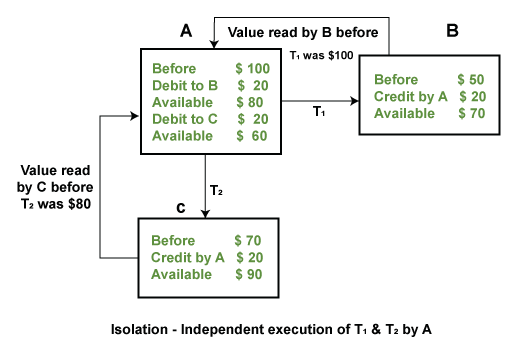
**Example:**



In the above figure, there are three accounts, A, B, and C, where A is making a transaction T one by one to both B & C. There are two operations that take place, i.e., Debit and Credit. Account A firstly debits $50 to account B, and the amount in account A is read $300 by B before the transaction. After the successful transaction T, the available amount in B becomes $150. Now, A debits $20 to account C, and that time, the value read by C is $250 (that is correct as a debit of $50 has been successfully done to B). The debit and credit operation from account A to C has been done successfully. We can see that the transaction is done successfully, and the value is also read correctly. Thus, the data is consistent. In case the value read by B and C is $300, which means that data is inconsistent because when the debit operation executes, it will not be consistent.

**4) Isolation:** The term 'isolation' means separation. In DBMS, Isolation is the property of a database where no data should affect the other one and may occur concurrently. In short, the operation on one database should begin when the operation on the first database gets complete. It means if two operations are being performed on two different databases, they may not affect the value of one another. In the case of transactions, when two or more transactions occur simultaneously, the consistency should remain maintained. Any changes that occur in any particular transaction will not be seen by other transactions until the change is not committed in the memory.

**Example:** If two operations are concurrently running on two different accounts, then the value of both accounts should not get affected. The value should remain persistent. As you can see in the below diagram, account A is making T1 and T2 transactions to account B and C, but both are executing independently without affecting each other. It is known as Isolation.



**4) Durability:** Durability ensures the permanency of something. In DBMS, the term durability ensures that the data after the successful execution of the operation becomes permanent in the database. The durability of the data should be so perfect that even if the system fails or leads to a crash, the database still survives. However, if gets lost, it becomes the responsibility of the recovery manager for ensuring the durability of the database. For committing the values, the COMMIT command must be used every time we make changes.

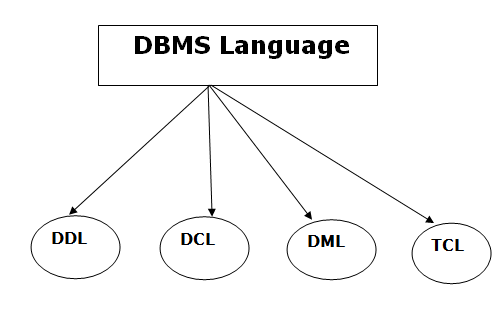
Therefore, the ACID property of DBMS plays a vital role in maintaining the consistency and availability of data in the database.

Thus, it was a precise introduction of ACID properties in DBMS. We have discussed these properties in the transaction section also.

# Database Language

* A DBMS has appropriate languages and interfaces to express database queries and updates.
* Database languages can be used to read, store and update the data in the database.

## Types of Database Language



### 1. Data Definition Language

* **DDL** stands for **D**ata **D**efinition **L**anguage. It is used to define database structure or pattern.
* It is used to create schema, tables, indexes, constraints, etc. in the database.
* Using the DDL statements, you can create the skeleton of the database.
* Data definition language is used to store the information of metadata like the number of tables and schemas, their names, indexes, columns in each table, constraints, etc.

Here are some tasks that come under DDL:

* **Create:** It is used to create objects in the database.
* **Alter:** It is used to alter the structure of the database.
* **Drop:** It is used to delete objects from the database.
* **Truncate:** It is used to remove all records from a table.
* **Rename:** It is used to rename an object.
* **Comment:** It is used to comment on the data dictionary.

These commands are used to update the database schema that's why they come under Data definition language.

### 2. Data Manipulation Language

**DML** stands for **D**ata **M**anipulation **L**anguage. It is used for accessing and manipulating data in a database. It handles user requests.

Here are some tasks that come under DML:

* **Select:** It is used to retrieve data from a database.
* **Insert:** It is used to insert data into a table.
* **Update:** It is used to update existing data within a table.
* **Delete:** It is used to delete all records from a table.
* **Merge:** It performs UPSERT operation, i.e., insert or update operations.
* **Call:** It is used to call a structured query language or a Java subprogram.
* **Explain Plan:** It has the parameter of explaining data.
* **Lock Table:** It controls concurrency.

### 3. Data Control Language

* **DCL** stands for **D**ata **C**ontrol **L**anguage. It is used to retrieve the stored or saved data.
* The DCL execution is transactional. It also has rollback parameters.

(But in Oracle database, the execution of data control language does not have the feature of rolling back.)

Here are some tasks that come under DCL:

* **Grant:** It is used to give user access privileges to a database.
* **Revoke:** It is used to take back permissions from the user.

There are the following operations which have the authorization of Revoke:

CONNECT, INSERT, USAGE, EXECUTE, DELETE, UPDATE and SELECT.

### 4. Transaction Control Language

TCL is used to run the changes made by the DML statement. TCL can be grouped into a logical transaction.

Here are some tasks that come under TCL:

* **Commit:** It is used to save the transaction on the database.
* **Rollback:** It is used to restore the database to original since the last Commit.

Data Independence

* Data independence can be explained using the three-schema architecture.
* Data independence refers characteristic of being able to modify the schema at one level of the database system without altering the schema at the next higher level.

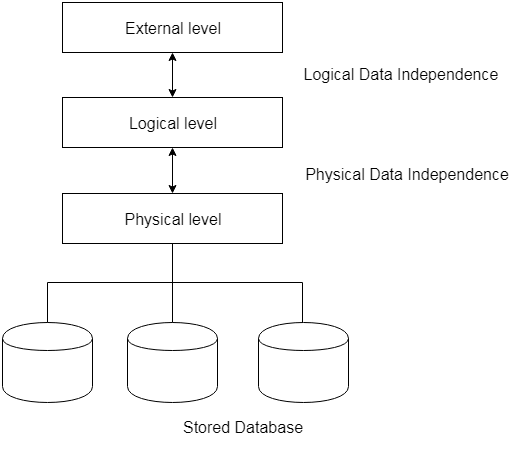
There are two types of data independence:

1. Logical Data Independence

* Logical data independence refers characteristic of being able to change the conceptual schema without having to change the external schema.
* Logical data independence is used to separate the external level from the conceptual view.
* If we do any changes in the conceptual view of the data, then the user view of the data would not be affected.
* Logical data independence occurs at the user interface level.

2. Physical Data Independence

* Physical data independence can be defined as the capacity to change the internal schema without having to change the conceptual schema.
* If we do any changes in the storage size of the database system server, then the Conceptual structure of the database will not be affected.
* Physical data independence is used to separate conceptual levels from the internal levels.
* Physical data independence occurs at the logical interface level.



**Fig: Data Independence**

|  |  |  |
| --- | --- | --- |
| **Comparison Basis** | **MySQL** | **MongoDB** |
| **Definition** | It is an open-source, cross-platform relational database management system built by Swedish Company MYSQL AB and currently supported by the Oracle. | It is a popular open-source NoSQL database management system developed and owned by MongoDB Inc. that stores data in JSON-like format. |
| **Release** | It was released on 23 May 1995. | It was released on 11 February 2009. |
| **Written in** | It is written in C and C++. | It is written in C, C++, and Java. |
| **Database Structure** | MySQL stores each individual records in tables and can access it by using the SQL queries. | MongoDB stores each individual record in JSON-like documents that may vary in structures. |
| **SQL or NoSQL** | MySQL uses Structured Query Language to process and access the database. We cannot change its schema. The inputs can only enter with a defined schema. SQL does not allow to work with unstructured and semi-structured data. | MongoDB is a NoSQL database system. It means we can define and adhere to the predefined structure of the incoming data. NoSQL allows working with unstructured and semi-structured data, which is not possible in RDBMS. Its schema can be changed. |
| **Queries Differences** | To select all records, it uses:   * Select \* from table\_name;   To insert records:   * INSERT INTO table\_name(stud\_id, branch, state) VALUES ('Joel01', 'MTech', 'Capetown') | To select all records, it uses:   * db.customer.find();   To insert records:   * db.table\_name.insert({ stud\_id: 'Joel01', branch: 'MTech', state: 'Capetown' }) |
| **Indexes Needed** | If the index is not found, the database engine searches an entire table for finding the rows. | If the index is not found, the database engine searches each document, including collection, for selecting the exact match documents. |
| **Features** | MySQL supports the following features:   * It is secure. * It is scalable. * It follows the client-server architecture. * It provides High Performance * It allows transactions to be rolled back, commit, and crash recovery. * It is flexible. * It support schema structure. * Triggers * Unicode support | MongoDB supports the following features:   * It supports ad hoc queries. * It provides duplication of data running over multiple servers. * It supports Master-Slave Replication. * It has automatic load balancing. * It does not have any schema. * It uses JavaScript instead of stored procedures. * It supports the JSON-like data model. * It supports rich query language. |
| **The Flexibility of Schema Design** | Once the schema design is defined, it cannot be changed. | Its schema design can be changed, which means it supports dynamic schema. |
| **Architecture** | MySQL does not build on Distributed System Architecture. But, MySQL Cluster has distributed database architecture. | MongoDB is Completely built on Distributed System Architecture. |
| **Terminologies Differences** | It uses:   * Table * Row * Columns * Joins | It uses:   * Collection * Document * Field * Embedded Document, linking |
| **Who uses?** | MySQL used by the following organization:   * Pinterest * Twitter * YouTube * Netflix * Spotify * US Navy * NASA * Walmart * Paypal | MongoDB used by the following organization:   * Klout * Citrix * Twitter * T-Mobile * Zendesk * Sony * Hootsuite * SurveyMonkey * MuleSoft * Foursquare * InVision |
| **Scaling** | It scales in vertically | It scales in Horizontally. |
| **Latest Release Version** | MySQL 8.0.21 (February 2020) | MongoDB 4.2 (February 2020) |

# Difference between MySQL and Oracle

MySQL and Oracle are the two famous relational databases that are used in small and big companies. Although Oracle Corporation supports both databases, they also have a lot of differences. Oracle is the more powerful software in comparison to MySQL. In this section, we are going to compare the differences between MySQL and Oracle database systems based on the various parameters.

### What is MySQL?

[MySQL](https://www.javatpoint.com/mysql-tutorial) is the popular database management system used for managing the relational database. It is open-source database software, which is supported by Oracle Company. It is fast, scalable, and easy to use database management system in comparison with Microsoft [SQL Server](https://www.javatpoint.com/sql-server-tutorial) and Oracle Database. It is commonly used with PHP scripts for creating powerful and dynamic server-side or web-based enterprise applications.

It is developed and supported by the Swedish Company, **MySQL AB**, and written in C and [C++ programming languages](https://www.javatpoint.com/cpp-tutorial). Many small and big companies use MySQL. MySQL supports many Operating Systems like Windows, [Linux](https://www.javatpoint.com/linux-tutorial), MacOS, etc. with [C](https://www.javatpoint.com/c-programming-language-tutorial), C++, and [Java languages](https://www.javatpoint.com/java-tutorial).

**Features of MySQL Database**

65.1M

1.2K

Exception Handling in Java - Javatpoint

The essential features of the MySQL database are given below:

* MySQL is a relational database management system and easy to use. We can build and interact with MySQL by using only a few simple SQL statements.
* It is secure because passwords are encrypted in MySQL.
* It follows a client /server architecture.
* It is free and open-source.
* It is scalable.
* It allows transactions to be rolled back, commit, and crash recovery.
* It provides high performance, high flexibility, and high productivity.

### What is Oracle?

[Oracle](https://www.javatpoint.com/oracle-tutorial) is a relational database system that provides self-driving, self-securing, self-repairing, and designed to eliminate error-prone manual database management. Oracle is a cross-platform database system which can run on the various operating system. It allows to store and retrieve data quickly and safely. It is available free for the student but cannot use for commercial purposes. It is the first database software developed for business purposes to manipulate data using a query language. Oracle was released in 1980 with basic SQL features. This software is scalable, portable, distributed, and programmable.

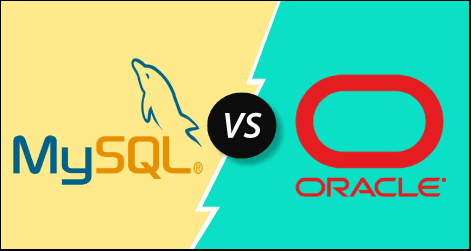
**Features of Oracle Database**

The essential features of an Oracle database are given below:

* Oracle database is a cross-platform because it can run on various operating systems such as Windows, Linux, Mac, etc.
* It supports a logical database structure that allows interacting with the database without knowing the physical storage of your data.
* It is scalable, portable, distributed, and programmable.
* It can handle a large amount of data quickly.
* It supports ACID property that allows us to maintain the integrity and reliability of your data.
* Oracle has networking stacks that enable us to communicate applications across the different platforms with oracle database smoothly.
* It has a recovery manager tool that provides cold, hot, and incremental database backups and recoveries.

### MySQL vs. Oracle

Let us summaries the popular differences between MySQL and Oracle in the tabular form given below:



|  |  |  |
| --- | --- | --- |
| **Comparison Basis** | **MySQL** | **Oracle** |
| Introduction | It is an open-source, cross-platform relational database management system built by Swedish Company MYSQL AB and currently supported by the Oracle. | Oracle is a relational database system (RDBMS) that implements object-oriented features. It allows to store and retrieve data quickly and safely. It can handle a large amount of data. |
| Release | It was released in 1995. | It was released in 1980. |
| Cost | It is free and open-source. It is licensed under the GNU. | It is licensed for commercial purposes, but it provides the express edition for free. The express edition is recommended for students only. |
| Scalability | MySQL database is used for small and big businesses. | Oracle database is used for very large scale deployments. |
| Data Partitioning | It does not support data partitioning. | It supports data partitioning. |
| Security | It requires a username, password, and host to access the database. | It requires a username, password, and profile validation to access the database. |
| System Type | It only works with the static system. | It can work with both static and dynamic systems. |
| Null Value | MySQL supports the null value. | Oracle does not support the null value. |
| Character | MySQL support only two characters that are CHAR and VARCHAR. | Oracle supports four different characters that are CHAR, VARCHAR2, NCHAR, and NVARCHAR2. |
| Backup Mechanism | It offers only two backup mechanisms that are mysqlhotcopy and mysqldump. | It offers many backup mechanisms that are backup, hot backup, import, export, etc. |
| XML Support | It does not support XML. | It supports XML. |
| Storage Features | It contains only a few storage features like tablespace, synonym, packages, and many others. | It supports many storage features that are tablespace, synonym, packages, etc. |
| Locking facility | MySQL has only a table locking facility. | Oracle has table locking as well as a row locking facility. |
| Language support | MySQL support only SQL language. | Oracle supports both SQL and PL/SQL languages. |
| Operating System Support | It supports the following Operating System:   * Windows * Mac OS X * Linux * UNIX * z/OS * BSD * Symbian * AmigaOS | It supports the following Operating System:   * Windows * Mac OS X * Linux * UNIX * z/OS |

# Difference between MySQL and SQL

SQL extends for Structured Query Language. SQL is a standard language that enables the user to design and manage databases. On the other hand, MySQL is a relational database management system that allows a user to store and retrieve data from the database. MySQL uses SQL to perform specific operations on the database. Both MySQL and SQL offer two trendy and differentiable servers: MySQL server and SQL Server for database management.

Let's understand the difference between [MySQL](https://www.javatpoint.com/mysql-tutorial) and [SQL](https://www.javatpoint.com/sql-tutorial) Server.

### 1) Developers

SQL is developed by Microsoft Corporation and named as Microsoft SQL Server (MS SQL). On the other hand, MySQL is developed by Oracle Corporation. Its name is a combo pack of "My (under co-founder daughter's name) and Structured Query Language (SQL)".

### 2) Availability

MySQL is open-source software, which is available free to all. In contrast, SQL is not an open-source software and hence not available free of cost.

52.2M

953

Java Try Catch

### 3) Platforms support

SQL was initially developed for the [Windows operating system](https://www.javatpoint.com/windows). Currently, it is supported by [Linux](https://www.javatpoint.com/linux-tutorial) and macOS (via Docker), lacking certain features that are supported in the Windows platform. While MySQL works well with Windows, macOS, Linux, Solaris platforms.

### 4) Programming Languages Support

MS SQL itself is a programming language, but the SQL Server supports basic programming languages such as [C++](https://www.javatpoint.com/cpp-tutorial), [Go](https://www.javatpoint.com/go-tutorial), [R](https://www.javatpoint.com/r-tutorial), [PHP](https://www.javatpoint.com/php-tutorial), [Python](https://www.javatpoint.com/python-tutorial), [Ruby](https://www.javatpoint.com/ruby-tutorial), Visual Basic, etc. In addition to the basic programming languages, MySQL also supports Perl, Haskel, Tcl, etc.

### 5) Storage Engine

MySQL does not require a large amount of storage space for performing different operations. It supports multiple storage engines. MySQL also supports plug-in storage engines. On the other hand, MS SQL supports only a single storage engine. Therefore, programmers need to be updated with more improved engines.

### 6) Security Offered

MySQL is a less securable server because it allows database files manipulation by other processors or its own binaries at its execution time only.

But, MS SQL provides a highly securable job. It does not allow database file access or manipulation through other processors or its own binaries at its execution time.

### 7) Backup

In MySQL, for data backup, the developer needs to extract it as SQL statements. While backing up the data, the server blocks the database, which reduces the chance of data corruption when switching from one version of MySQL to another.

In MS SQL, the server does not block the database at the time of backup. It means while data backup, the developer can perform other operations on the database.

### 8) Time Consumption in Data restoration

MySQL consumes a high amount of time for data restoration because it executes multiple SQL statements altogether, while MS SQL makes less effort and time to restore a huge amount of data.

### 9) Canceling Query Execution

MySQL does not provide the facility to stop or cancel a query at its execution time. To do so, the user needs to cancel the whole process. Unlike MySQL, the MS SQL server provides the feature to truncate a query at its execution time without disturbing or canceling the entire process.

### 10) Software Stack Component

The enterprise can select various editions of MS SQL according to the user requirements for the project. On the other hand, MySQL is used by many web application developers as a component of the LAMP stack.

### 11) Editions

There are two editions available in MySQL. The user can either use MySQL Community Server or MySQL Enterprise Server. Whereas, MS SQL is available in various specialized editions. The user can select from the web, enterprise, standard, or Express editions of SQL.

### 12) Multilingual

MySQL is available only in the English language. In contrast, SQL is available in many different languages.

### 13) Syntax

The SQL syntax is easy to use and implement. On the other hand, MySQL syntax is a little bit typical to use and implement.

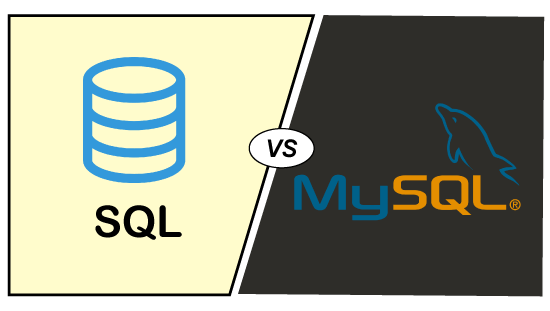
For example,

If we want to implement the length function, the following queries will be used as per:

1. MS SQL: **SELECT** LEN(req\_string) **FROM** <Table\_name>
3. MySQL: **SELECT** CHARACTER\_LENGTH(req\_string) **FROM** <Table\_name>

## MySQL vs. SQL

Let us understand some of the big differences between MySQL and SQL using the following comparison chart:



|  |  |  |  |
| --- | --- | --- | --- |
| **SN** | **Parameter** | **MySQL** | **SQL** |
| 1. | Definition | MySQL is the popular open-source database available in the market, which is developed by the Swedish company MySQL AB. | SQL (Structured Query Language) is a programming language that is useful for managing our relational databases. |
| 2. | Purpose | MySQL used for data handling, storing, deleting, and updating the data in tabular form. | It is used to query and operate the database. |
| 3. | Updates | MySQL is software, so it gets frequent updation. The current stable version is v8.0.20, which provides two times faster speed than the previous versions. | SQL is a programming language; that's why it does not get any updates. Its commands or statements always fixed and remain the same. |
| 4. | Type | It is database software that uses SQL language to conduct with the database. | It is a query language for managing databases. |
| 5. | Complexity | It is easily used through simple downloading and installation. | It requires learning the language to use it effectively. |
| 6. | Usage | MySQL is used as RDBMS for managing relational databases. | SQL commands or statements are used in various DBMS and RDBMS. MySQL itself uses SQL commands. |
| 7. | Support for Connectors | It provides the MySQL Workbench tool to design and develop databases. | No connectors are available in SQL. |
| 8. | Multilingual | It is available only in the English language. | It is available in many different languages. |
| 9. | Flexibility | It does not provide support for XMAL and user-defined functions. | It includes support for XMAL and user-defined functions. |
| 10. | Community Support | MySQL is free to use so that it has very rich community support. | It does not have excellent community support. If we find any problem, we need to go to Microsoft SQL Server support. |
| 11. | Advantage | Open-Source. Data security. High Performance. Data Security. Complete workflow controls. | No need for coding. High speed. Portability. Multiple views of data. Interactive language. |

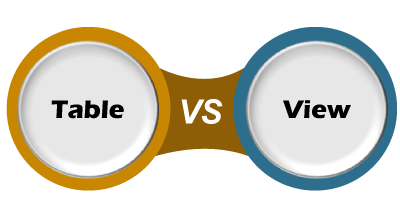
### Similarities

MySQL and SQL have some common functionalities, which are explained below:

* Both are related to managing the relational database.
* Both have data types.
* Both provides aliasing features to the database users.
* Both uses join operation (Inner, Left, Right, Self, Cross) in the table.
* Both are able to perform arithmetic (+, -, \*, /, %) and comparison (>, <, >=, <=), and logical (and, or, not) operations.
* Both can use aggregate functions such as sum, count, average, etc.
* Both have stored procedures, indexing, view, and triggers.

# Difference between Table and View

Table and view are the two basic terms used in the relational database environment. The difference between table and view is debated among beginners and database administrators (DBA) because both share some common similarities. The main difference between them is that a **table is an object that consists of rows and columns to store and retrieve data** whenever the user needs it. In contrast, the **view is a virtual table based on an SQL statement's result set** and will disappear when the current session is closed. In this article, we are going to discuss comparisons between tables and views based on various aspects.



## What is a table?

A table **consists of rows and columns used to organize data** to store and display records in a structured format. It is similar to worksheets in the spreadsheet application. It occupies space on our systems. We need three things to create a table:

* Table name
* Columns/Fields name
* Definitions for each field

We can create a table in [MySQL](https://www.javatpoint.com/mysql-tutorial)

using the below syntax:

1. **CREATE** **TABLE** [IF NOT EXISTS] table\_name (
2. column\_definition1,
3. column\_definition2,
4. ........,
5. table\_constraints
6. );

**The following are the main advantages of the table:**

1. It provides an efficient way to summarize the given information into a structured form that helps to find out the information quickly.
2. It allows us to add the data in a specific way rather than in a paragraph that makes the data more understandable.
3. It enables **quick searching** for the data we need.
4. It helps in introducing relationships between various data using **referential constraints**.
5. It can be associated with data security that allowing only authorized people for data accessing.

## What is a view?

The view is a **virtual/logical table** formed as a result of a query and used to view or manipulate parts of the table. We can create the columns of the view from one or more tables. Its content is based on **base tables**.

The view is a database object with no values and contains rows and columns the same as real tables. It **does not occupy space** on our systems.

We can create a view in MySQL using the below syntax:

1. **CREATE** **VIEW** view\_name **AS**
2. **SELECT** columns
3. **FROM** tables
4. [**WHERE** conditions];

**The following are the main advantages of the view:**

1. Views are usually virtual and do not occupy space in systems.
2. Views enable us to hide some of the columns from the table.
3. It simplifies complex queries because it can draw data from multiple tables and present it as a single table.
4. It helps in **data security** that shows only authorized information to the users.
5. It presents a consistent, unchanged image of the database structure, even if the source tables are renamed, split, or restructured.

## Key differences between Table and View

The following points explain the differences between tables and views:

* A table is a database object that holds information used in applications and reports. On the other hand, a view is also a database object utilized as a table and can also link to other tables.
* A table consists of rows and columns to store and organized data in a structured format, while the view is a result set of SQL statements.
* A table is **structured** with columns and rows, while a view is a virtual table **extracted** from a database.
* The table is an independent data object while views are usually depending on the table.
* The table is an **actual or real table** that exists in physical locations. On the other hand, views are the **virtual or logical table** that does not exist in any physical location.
* A table allows to performs add, update or delete operations on the stored data. On the other hand, we cannot perform add, update, or delete operations on any data from a view. If we want to make any changes in a view, we need to update the data in the source tables.
* We cannot **replace** the table object directly because it is stored as a physical entry. In contrast, we can easily use the replace option to recreate the view because it is a pseudo name to the SQL statement running behind on the database server.

## Table vs. View Comparison Chart

The following comparison chart explains their main differences in a quick manner:

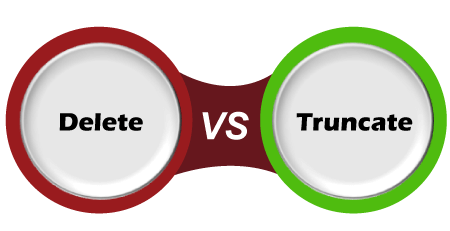
|  |  |  |
| --- | --- | --- |
| **SN** | **Table** | **View** |
| 1. | A table is used to organize data in the form of rows and columns and displayed them in a structured format. It makes the stored information more understandable to the human. | Views are treated as a virtual/logical table used to view or manipulate parts of the table. It is a database object that contains rows and columns the same as real tables. |
| 2. | Table is a physical entity that means data is actually stored in the table. | The view is a virtual entity, which means data is not actually stored in the table. |
| 3. | It is used to store the data. | It is used to extract data from the table. |
| 4. | It generates a fast result. | The view generates a slow result because it renders the information from the table every time we query it. |
| 5. | It is an independent data object. | It depends on the table. Therefore we cannot create a view without using tables. |
| 6. | Table allows us to perform DML operations. | The view will enable us to perform DML operations. |
| 7. | It is not an easy task to replace the table directly because of its physical storage. | It is an easy task to replace the view and recreate it whenever needs. |
| 8. | It occupies space on the systems. | It does not occupy space on the systems. |

### Conclusion

In this article, we have made a comparison between table and view that are two database objects. A user cannot create a view without using tables because it depends on the table.

# Command

The difference between DELETE and TRUNCATE command is the most common part of an interview question. They are mainly used to delete data from the database. **The main difference between them is that the delete statement deletes data without resetting a table's identity, whereas the truncate command resets a particular table's identity**. This article explains the complete overview of DELETE and TRUNCATE command and their differences that are primarily used interchangeably but are totally different.



## What is a DELETE command?

It is a **DML or data manipulation command** used to deletes records from a table that is not required in the database. It removes the complete row from the table and produces the count of deleted rows. We need the delete permission on the target table to execute this command. It also allows us to filter and delete any specific records using the [**WHERE** clause](https://www.javatpoint.com/mysql-where) from the table.

It clarifies that we have a backup of our database before executing this command because we cannot recover the deleted records using this query. Therefore, the **database backups** allow us to restore the data whenever we need it in the future.

The following syntax explains the DELETE command to remove data from the table:

Play Videox

1. **DELETE** **FROM** table\_name **WHERE** condition;

## What is a TRUNCATE command?

The truncate statement is a **DDL or data definition language command** used to removes complete data from the table without removing the table structure. We cannot use the **WHERE** clause with this command, so that filtering of records is not possible. After executing this command, we **cannot rollback the deleted data** because the log is not maintained while performing this operation.

The truncate command deallocates the **pages instead of rows** and makes an entry for the deallocating pages instead of rows in transaction logs. This command locks the pages instead of rows; thus, it requires fewer locks and resources. Note that we cannot use the truncate statement when a table is referenced by a foreign key or participates in an indexed view.

The following syntax explains the TRUNCATE command to remove data from the table:

1. **TRUNCATE** **TABLE** table\_name;

## Key differences between DELETE and TRUNCATE

The following points explain the differences between delete and truncate command:

1. The DELETE statement is used when we want to remove some or all of the records from the table, while the TRUNCATE statement will delete entire rows from a table.
2. DELETE is a DML command as it only modifies the table data, whereas the TRUNCATE is a DDL command.
3. DELETE command can filter the record/tuples by using the WHERE clause. However, the TRUNCATE command does not allow to use **WHERE** clause, so we cannot filter rows while truncating.
4. DELETE activates all **delete triggers** on the table to fire. However, no triggers are fired on the truncate operation because it does not operate on individual rows.
5. DELETE performs deletion row-by-row one at a time from the table, in the order, they were processed. However, TRUNCATE operates on data pages instead of rows because it deleted entire table data at a time.
6. DELETE statement only deletes records and does not reset the **table's identity**, whereas TRUNCATE resets the identity of a particular table.
7. DELETE command require more locks and database resources because it acquires the lock on every deleted row. In contrast, TRUNCATE acquires the lock on the data page before deleting the data page; thus, it requires fewer locks and few resources.
8. DELETE statement makes an entry in the **transaction log** for each deleted row whereas, TRUNCATE records the transaction log for each data page.
9. TRUNCATE command is **faster** than the DELETE command as it deallocates the data pages instead of rows and records data pages instead of rows in transaction logs.
10. Once the record deletes by using the TRUNCATE command, we cannot recover it back. In contrast, we can recover the deleted data back which we removed from the DELETE operation.

## DELETE vs. TRUNCATE Comparison Chart

The following comparison chart explains their main differences in a quick manner:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Comparison Basis** | | **DELETE** | | **TRUNCATE** | |
| **Definition** | | The delete statement is used to remove single or multiple records from an existing table depending on the specified condition. | | The truncate command removes the complete data from an existing table but not the table itself. It preserves the table structure or schema. | |
| **Language** | | It is a DML (Data Manipulation Language) command. | | It is a DDL (Data Definition Language) command. | |
| **WHERE** | | It can use the WHERE clause to filter any specific row or data from the table. | | It does not use the WHERE clause to filter records from the table. | |
| **Permission** | | We need to have DELETE permission to use this command. | | We need to have ALTER permission to use this command. | |
| **Working** | | This command eliminates records one by one. | | This command deletes the entire data page containing the records. | |
| **Lock** | | It will lock the row before deletion. | | It will lock the data page before deletion. | |
| **Table Identity** | | This command does not reset the table identity because it only deletes the data. | | It always resets the table identity. | |
| **Transaction** | | It maintains transaction logs for each deleted record. | | It does not maintain transaction logs for each deleted data page. | |
| **Speed** | | Its speed is slow because it maintained the log. | | Its execution is fast because it deleted entire data at a time without maintaining transaction logs. | |
| **Trigger** | | This command can also activate the trigger applied on the table and causes them to fire. | | This command does not activate the triggers applied on the table to fire. | |
| **Restore** | | It allows us to restore the deleted data by using the COMMIT or ROLLBACK statement. | | We cannot restore the deleted data after using executing this command. | |
| **Indexed view** | | It can be used with indexed views. | | It cannot be used with indexed views. | |
| **Space** | | The DELETE statement occupies more transaction space than truncate because it maintains a log for each deleted row. | | The TRUNCATE statement occupies less transaction space because it maintains a transaction log for the entire data page instead of each row. | |
| Delete | | Truncate | |
| The DELETE command is used to delete specified rows(one or more). | | While this command is used to delete all the rows from a table. | |
| It is a DML(Data Manipulation Language) command. | | While it is a DDL(Data Definition Language) command. | |
| There may be a WHERE clause in the DELETE command in order to filter the records. | | While there may not be WHERE clause in the TRUNCATE command. | |
| In the DELETE command, a tuple is locked before removing it. | | While in this command, the data page is locked before removing the table data. | |
| The DELETE statement removes rows one at a time and records an entry in the transaction log for each deleted row. | | TRUNCATE TABLE removes the data by deallocating the data pages used to store the table data and records only the page deallocations in the transaction log. | |
| DELETE command is slower than TRUNCATE command. | | While the TRUNCATE command is faster than the DELETE command. | |
| To use Delete you need DELETE permission on the table. | | To use Truncate on a table we need at least ALTER permission on the table. | |
| The identity of the fewer column retains the identity after using DELETE Statement on the table. | | Identity the column is reset to its seed value if the table contains an identity column. | |
| The delete can be used with indexed views. | | Truncate cannot be used with indexed views. | |
| This command can also active trigger. | | This command does not active trigger. | |
| DELETE statement occupies more transaction spaces than Truncate. | | Truncate statement occupies less transaction spaces than DELETE | |

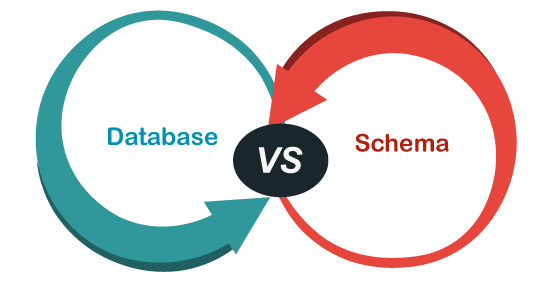
### Conclusion

In this article, we have made a comparison between delete and truncate statement. We have concluded that a DELETE command is used when we want to customize the deletion of records from the table. And a TRUNCATE command is used when we do not want to left any records or data in the table, i.e., we want to empty the table.

# Difference between Database and Schema

The database is a common term in today's life. Many organizations, firms, enterprises, and institutes require a system to store their data in a well-formatted structure so that it might be easy to get valuable information whenever needs. Each database is made of physical files that contain data and metadata. Thus we can say that a database is a memory component to store information. Each database is a collection of schema means that while designing a database, we need to specify the schema for their structural view.

The difference between DATABASE and SCHEMA terminology is the most common part of an interview question. **The main difference between them is that the database is a collection of interrelated data, whereas schema is the database's structural view**. This article explains the complete overview of DATABASE and SCHEMA and their differences that are primarily used interchangeably but are totally different.



## What is a database?

A database is an application that stores the collection of organized and interrelated data. Each database has its own structure, data types, and constraints of the data, their relation with another constraint, and the data or information about an object. **The data stored in a database would update regularly**. Hence it changes frequently. We can modify or change the data stored in the database using the **DML (data manipulation language) command**. The data in the database at a particular moment is called a database instance.

A database can be generated and operated either manually or computerized. The size of the database based on the user's needs. In today's life, the database is generally used digitally.

53.6M

982

HTML Tutorial

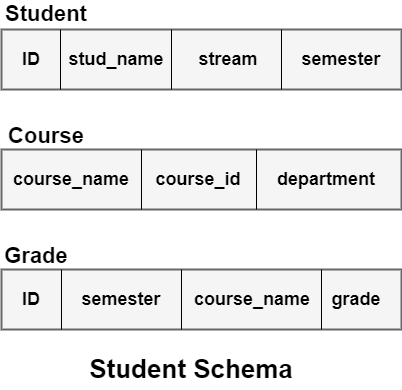
## What is a schema?

A schema is a **logical representation** of a database that describes the structural definition or description of an entire database. Generally, it is nothing more than a user who owns database objects such as tables, views, etc. We must specify schema during the design of a database. Once we define the database schema, we should not change it frequently because it would disturb the organization of data in a database.

We can display a database schema in the form of a diagram referred to as a **schema diagram**. This diagram indicates what data contains in a table, what variables are, and how they are associated with each other. Note that the schema diagram doesn't show every aspect of the database, such as database instances, type of the attributes, etc.

We can specify the schema using the **DDL (Data Definition Language) statements**. The DDL statement sets the table name, the attributes and their types, constraints, and its relation with other tables in a database. We can also use this statement when we want to modify the schema.

**For example**, the below representation shows the schema of a database with student information. Here we can see all table's name and their variables.



## Key Differences between Database and Schema

The following points explain the main differences between database and schema:

* The fundamental difference between them is that the database is an organized collection of interrelated data or information about the considered object. In contrast, the schema is a logical representation or description of an entire database.
* Once we declare a database schema, we must not change it often as it would disturb the organization of data stored in the database. In contrast, we can update the database regularly.
* The database is a collection of schema, records, and constraints for the tables. On the other hand, a schema contains the structure of tables, attributes, their types, constraints, and how they relate to other tables.
* The DDL statement is used to generate and modify the schema. On the other hand, DML statements are used to create and modify the data inside the database.
* Each database **uses the memory** to store the data, whereas the schema can **use a logical tructure** to store data.

## Database vs. Schema Comparison Chart

The following comparison chart explains their main differences in a quick manner:

|  |  |  |
| --- | --- | --- |
| **Comparison Basis** | **Database** | **Schema** |
| **Definition** | The database is an application that stores the organized collection of interrelated data. | The schema is a logical representation of a database. |
| **Statement** | A DML is used to generate and modify the records in the database. | A DDL statement defined the schema for a database. |
| **Modification** | A database is updating the data regularly so that it can change frequently. | We should not change a schema often once it is declared. |
| **Include** | A database is a collection of schema, records, and constraints for the tables. | A schema always included the name of the tables, columns name, their types, and constraints. |
| **Memory** | It uses memory to store data. | It does not use memory to store data. |

## Database vs. Schema in MySQL

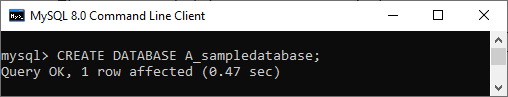
[MySQL](https://www.javatpoint.com/mysql-tutorial) does not provide any real distinction between a database and schema. They are used interchangeably, which means **schema is synonymous with the database**. As we write the query for creating the database, we can use a similar query for creating the schema.

**We can clarify this concept by creating a database and a schema using the steps are as follows**:

First, we can use the below syntax to create a database:

1. **CREATE** **DATABASE** database\_name;

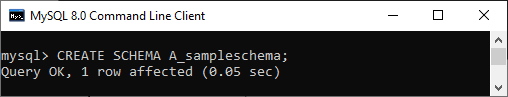
We can use this syntax in a query as below output:



Second, we can use the below syntax to create a database:

1. **CREATE** **SCHEMA** schema\_name;

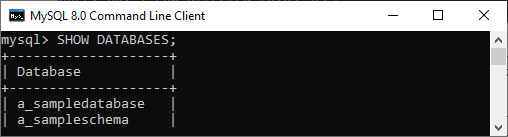
We can use this syntax in a query as below output:



Now, both the database and the schema have been created successfully. We can use the **SHOW** command to display the database and the schema. The query for the above illustration is given below:

1. mysql> SHOW DATABASES;

We will see the following output of the above query:



### Conclusion

In this article, we have made a comparison between database and schema. Here we conclude that MySQL has no actual differences between a schema and the database. They are used interchangeably means both are synonymous. On the other hand, there is a clear distinction between them in some other languages. In simple terms, databases are collections of schema, and schemas are a collection of tables.

# How to Create Index in MySQL

An index is a data structure that allows us to add indexes in the existing table. It enables you to improve the faster retrieval of records on a database table. It creates an **entry** for each value of the indexed columns. We use it to quickly find the record without searching each row in a database table whenever the table is accessed. We can create an index by using one or more **columns** of the table for efficient access to the records.

When a table is created with a primary key or unique key, it automatically creates a special index named **PRIMARY**. We called this index as a clustered index. All indexes other than PRIMARY indexes are known as a non-clustered index or secondary index.

## Need for Indexing in MySQL

Suppose we have a contact book that contains names and mobile numbers of the user. In this contact book, we want to find the mobile number of Martin Williamson. If the contact book is an unordered format means the name of the contact book is not sorted alphabetically, we need to go over all pages and read every name until we will not find the desired name that we are looking for. This type of searching name is known as sequential searching.

# SQL indexes

An index is a schema object. It is used by the server to speed up the retrieval of rows by using a pointer. It can reduce disk I/O(input/output) by using a rapid path access method to locate data quickly. An index helps to speed up select queries and where clauses, but it slows down data input, with the update and the insert statements. Indexes can be created or dropped with no effect on the data. In this article, we will see how to create, delete, and uses the INDEX in the database.

For example, if you want to reference all pages in a book that discusses a certain topic, you first refer to the index, which lists all the topics alphabetically and is then referred to one or more specific page numbers.

### ****Creating an Index:****

**Syntax:**

CREATE INDEX index

ON TABLE column;

### ****For multiple columns:****

**Syntax:**

CREATE INDEX index

ON TABLE (column1, column2,.....);

### ****Unique Indexes:****

Unique indexes are used for the maintenance of the integrity of the data present in the table as well as for the fast performance, it does not allow multiple values to enter into the table.   
 **Syntax:**

CREATE UNIQUE INDEX index

ON TABLE column;

### ****When should indexes be created:****

* A column contains a wide range of values.
* A column does not contain a large number of null values.
* One or more columns are frequently used together in a where clause or a join condition.

**When should indexes be avoided:**

* The table is small
* The columns are not often used as a condition in the query
* The column is updated frequently

**Removing an Index:**

To remove an index from the data dictionary by using the **DROP INDEX** command.

**Syntax:**

DROP INDEX index;

To drop an index, you must be the owner of the index or have the **DROP ANY INDEX** privilege. 

### ****Altering an Index:****

To modify an existing table’s index by rebuilding, or reorganizing the index.

ALTER INDEX IndexName

ON TableName REBUILD;

### ****Confirming Indexes :****

You can check the different indexes present in a particular table given by the user or the server itself and their uniqueness.

**Syntax:**

select \* from USER\_INDEXES;

It will show you all the indexes present in the server, in which you can locate your own tables too.

### ****Renaming an index :****

You can use the system stored procedure sp\_rename to rename any index in the database.

**Syntax:**

EXEC sp\_rename

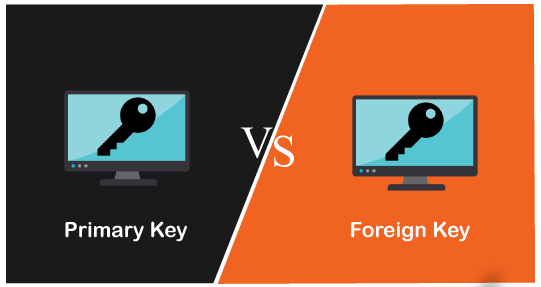
index\_name,

new\_index\_name,

N'INDEX';

# Difference between Primary Key and Foreign Key

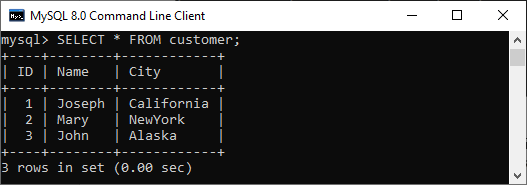
Key in MySQL are the fundamental elements for constructing a relationship between two tables. They are very useful in the maintenance of a relational database structure. **The main difference between them is that the primary key identifies each record in the table, whereas the foreign key is used to link two tables together**. In this article, we are going to cover the essential differences between Primary and Foreign Keys based on various parameters. Before making a comparison, we will discuss in brief these keys.



## What is Primary Key?

The [primary key](https://www.javatpoint.com/mysql-primary-key) is a **unique or non-null key** that uniquely identifies every record in a table or relation. Each database needs a unique identifier for every row of a table, and the primary key plays a vital role in identifying rows in the table uniquely. The primary key column can't store **duplicate values**. It is also called a **minimal super key**; therefore, we cannot specify more than one primary key in any relationship.

**For example**, we have a table named **customer** with attributes such as ID, Name, and City. Only the **ID column** can never contain duplicate and NULL values because each customer has a unique identification number. This feature helps to identify each record in the database uniquely. Hence, we can make the ID attribute a primary key.

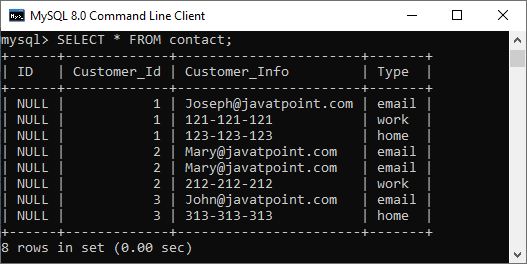


## What is Foreign Key?

The [foreign key](https://www.javatpoint.com/mysql-foreign-key) is a group of one or more columns in a database to uniquely identify another database record in some other table to maintain the referential integrity. It is also known as the **referencing key that establishes a relationship** between two different tables in a database. A foreign key always matches the primary key column in another table. It means a foreign key column in one table refers to the primary key column of another table. A foreign key is beneficial in relational database normalization, especially when we need to access records from other tables.

A foreign key creates a **parent-child relationship** with the tables where the parent table holds the initial column values, and the child table references the parent column values. We can achieve this relationship only when the foreign key constraint is found on the child table.

**For example**, we have a table named **contact** with attributes such as ID, Customer\_Id, Customer\_Info, and Type. Here we can make the **Customer\_Id column** a foreign key.



If we want to delete the referential data that removes records from both tables, we can define the foreign key in the contact table as below:

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

When we delete any record from the customer table, the related rows will also delete in the contact table, and both tables update automatically.

## Key differences between Primary Key and Foreign Key

The following points explain the differences between primary and foreign keys:

* A primary key constraint in the relational database acts as a unique identifier for every row in the table. In contrast, a foreign key constraint establishes a relationship between two different tables to uniquely identify a row of the same table or another table.
* The primary key column does not store NULL values, whereas the foreign key can accept more than one NULL value.
* Each table in a relational database can't define more than one primary key while we can specify multiple foreign keys in a table.
* We can't remove the parent table's primary key value, which is referenced with a foreign key column in the child table. In contrast, we can delete the child table's foreign key value even though they refer to the parent table's primary key.
* A primary key is a unique and non-null constraint, so no two rows can have identical values for a primary key attribute, whereas foreign key fields can store duplicate values.
* We can insert the values into the primary key column without any limitation. In contrast, we need to ensure that the value is present in a primary key column while inserting values in the foreign key table.
* We can implicitly define the primary key constraint on temporary tables, whereas we cannot enforce foreign key constraints on temporary tables.

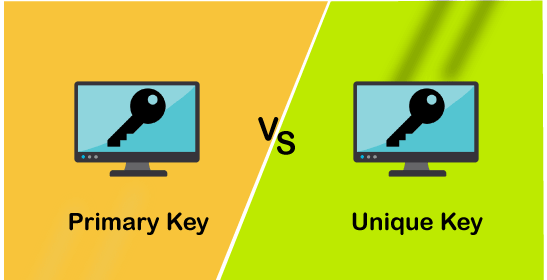
## Primary Key vs. Foreign Key Comparison Chart

The following comparison chart explains their main differences in a quick manner:

|  |  |  |
| --- | --- | --- |
| **Comparison Basis** | **Primary Key** | **Foreign Key** |
| **Basic** | It is used to identify each record into the database table uniquely. | It is used to links two tables together. It means the foreign key in one table refers to the primary key of another table. |
| **NULL** | The primary key column value can never be NULL. | The foreign key column can accept a NULL value. |
| **Count** | A table can have only one primary key. | A table can have more than one foreign key. |
| **Duplication** | The primary key is a unique attribute; therefore, it cannot stores duplicate values in relation. | We can store duplicate values in the foreign key column. |
| **Indexing** | The primary key is a clustered index by default, which means it is indexed automatically. | A foreign key is not a clustered index by default. We can make clustered indexes manually. |
| **Deletion** | The primary key value can't be removed from the table. If you want to delete it, then make sure the referencing foreign key does not contain its value. | The foreign key value can be removed from the table without bothering that it refers to the primary key of another table. |
| **Insertion** | We can insert the values into the primary key column without any limitation, either it present in a foreign key or not. | The value that is not present in the column of a primary key cannot be inserted into the referencing foreign key. |
| **Temporary table** | The primary key constraint can be defined on the temporary tables. | A foreign key constraint cannot be defined on the temporary tables. |
| **Relationship** | It cannot create a parent-child relationship in a table. | It can make a parent-child relationship in a table. |

# Difference between Primary Key and Unique key

Keys in MySQL are the column or set of columns used to construct a relationship between one or more than two tables. They are also used for accessing records from the table. Both keys provide a guaranteed uniqueness for a column or a set of columns in a table or relation. **The main difference among them is that the primary key identifies each record in the table, and the unique key prevents duplicate entries in a column except for a NULL value**. In this article, we are going to compare essential differences between Primary and Unique Keys based on various parameters. Before making a comparison, we will discuss in brief these keys.

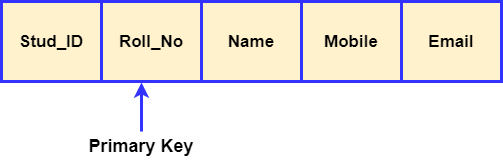


## What is Primary Key?

The [primary key](https://www.javatpoint.com/mysql-primary-key)

is a **unique or non-null** key that uniquely identifies every record in that table or relation. The primary key column cannot store duplicate values that mean primary key column values are always unique. It is also called a **minimal super key**; therefore, we cannot specify more than one primary key in any relationship. A primary key column of one table can be referenced by a foreign key column of another table.

**For example**, we have a table named **students** with attributes such as Stud\_ID, Roll\_No, Name, Mobile, and Email.



Here only the **Roll\_No** column can never contain an identical and NULL value. We know every student has a unique roll number. Therefore two students can never have the same roll number. This feature helps to identify each record in the database uniquely. Hence, we can make the Roll\_No attribute a primary key.

### Features of Primary Key

The following are the essential primary key features:

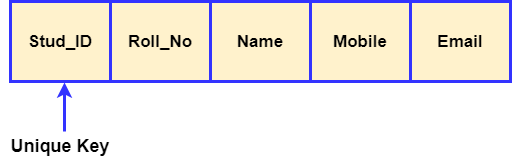
* The primary key column cannot contain duplicate values.
* The primary key implements the entity integrity of the table.
* A table cannot have more than one primary key column.
* We can make the primary key from one or more table fields.
* The primary key column should have NOT NULL constraints.

## What is a Unique Key?

The [unique key](https://www.javatpoint.com/mysql-unique-key)

is a single column or combination of columns in a table to uniquely identify database records. A unique key **prevents** from storing **duplicate values** in the column. A table can contain multiple unique key columns, unlike a primary key column. This key is similar to the primary key, except that one NULL value can be stored in the unique key column. The unique key is also called **unique constraints** and can be referenced by another table's foreign key.

**For example**, let's consider the same table named **students** with attributes such as Stud\_ID, Roll\_No, Name, Mobile, and Email.



Here **Stud\_ID** can be assigned as a unique constraint because each student must have a unique identification number. If a student changes the college, then he or she would not have any student ID. In that case, the entry may contain a **NULL** value because a unique key constraint allows storing NULL, but it should be only one.

### Features of Unique key

The following are the essential unique key features:

* We can construct the unique key from one or more table fields.
* A table can define multiple unique key columns.
* By default, a unique key is in non-clustered unique indexes.
* The unique constraint column can store NULL value, but only one NULL is allowed null per column.
* The foreign key can reference the unique constraint in preserving the uniqueness of a table.

## Key differences between Primary and Unique Key

The following points explain the key differences between primary and candidate keys:

* A primary key can constitute one or more fields of a table to identify records in a table uniquely. On the other hand, a unique key prevents two rows from having duplicate entries in a column.
* A table cannot have more than one primary key in a relational database, while there can be multiple unique keys per table.
* A primary key column cannot contain NULL values, whereas a unique key can have NULL values, but only one NULL is allowed in a table.
* A primary key should be unique, but a unique key cannot necessarily be the primary key.
* The primary key by default is a clustered index where data is physically organized in the sequential index. In contrast, the unique key is a unique non-clustered index.
* The primary key implements entity integrity, whereas the unique key enforces unique data.

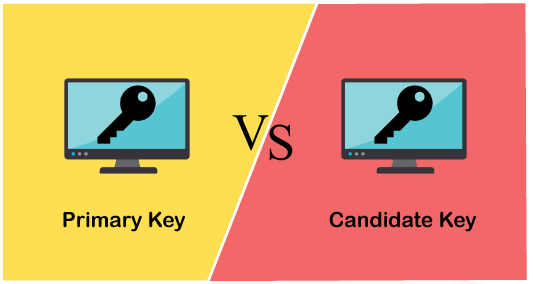
## Primary Key vs. Unique Key Comparison Chart

The following comparison chart explains their main differences in a quick manner:

|  |  |  |
| --- | --- | --- |
| **Comparison Basis** | **Primary Key** | **Unique Key** |
| **Basic** | The primary key is used as a unique identifier for each record in the table. | The unique key is also a unique identifier for records when the primary key is not present in the table. |
| **NULL** | We cannot store NULL values in the primary key column. | We can store NULL value in the unique key column, but only one NULL is allowed. |
| **Purpose** | It enforces entity integrity. | It enforces unique data. |
| **Index** | The primary key, by default, creates clustered index. | The unique key, by default, creates a non-clustered index. |
| **Number of Key** | Each table supports only one primary key. | A table can have more than one unique key. |
| **Value Modification** | We cannot change or delete the primary key values. | We can modify the unique key column values. |
| **Uses** | It is used to identify each record in the table. | It prevents storing duplicate entries in a column except for a NULL value. |
| **Syntax** | We can create a primary key column in the table using the below syntax:  CREATE TABLE Employee  (  Id INT PRIMARY KEY,  name VARCHAR(150),  address VARCHAR(250)  ) | We can create a unique key column in the table using the below syntax:  CREATE TABLE Person  (  Id INT UNIQUE,  name VARCHAR(150),  address VARCHAR(250)  ) |

# Difference between Primary Key and Candidate Key

Keys in MySQL are an attribute or a set of attributes used to access records from tables. They are also used to construct a relationship between two tables. Both Primary and Candidate Key identifies each record uniquely in a table or relation. **The most popular difference among them is that a table can have only one primary key but can have more than one candidate key**. In this article, we are going to compare essential differences between Primary and Candidate Keys based on various parameters. Before making a comparison, we will discuss in brief about these keys.



## What is Primary Key?

The [primary key](https://www.javatpoint.com/mysql-primary-key) is a **unique or not-null** key that uniquely identifies each record in a table or relation. It is a minimal super key; that's why it can contain only one primary key in any relationship. **For example**, suppose we have a table named **students** with ID, Name, Age, and Address columns. Here we can make only the ID column a primary key because all other column values can be the same, but the ID column can't be the same.

## What is Candidate Key?

A candidate key is an attribute or a set of attributes that identify each record in a table or relation uniquely but noted that a table could contain multiple candidate keys. This key can store a **NULL** value that opposes in a primary key. **For example**, suppose we have a table named **students** with ID, Name, DOB, Age, and Address columns. Here we can figure out two candidate keys that are {ID} and {Name, DOB}. Hence, it clarifies that more than one candidate key is available to identify the table or relation uniquely.

Since a table or relation can have more than one candidate key, one candidate key can qualify to become a primary key out of all candidate keys. Note that each candidate key can be a primary key, but only one should be chosen as the primary key. The rule to become a primary key among candidate keys is that the key's attribute values must be unique and can never be Null for any domain.

## Key differences between Primary and Candidate Key

The following points explain the main differences between primary and candidate keys:

* The primary key is a unique and essential attribute of a table or relation. In contrast, the candidate keys have many candidates, among which one candidate key can be selected as a primary key.
* The fundamental difference among both keys is that a table or relation in a schema can have only **one primary key**, but there can be **multiple candidate keys**.
* It is not mandatory to define a primary key, but there cannot be a relationship without the candidate key.
* The primary key attribute can **never** be a NULL because its main function is to identify a record in a table or relation uniquely. We can also use the primary key as a foreign key in other relations. Therefore, it cannot be NULL. This feature helps in finding the records in a referenced relation. The candidate key can be NULL unless the attribute constraint is specified as not-null.
* Each primary key can be a candidate key, but vice-versa is not possible.

## Primary Key vs. Candidate Key Comparison Chart

The following comparison chart explains their main differences in a quick manner:

|  |  |  |
| --- | --- | --- |
| **Comparison Basis** | **Primary Key** | **Candidate Key** |
| **Definition** | It is a unique and non-null key to identify each table's records in a schema uniquely. | It is also a unique key to identify records in relation or table uniquely. |
| **Basic** | A table or relation can contain only one primary key. | A table or relation can have more than one candidate key. |
| **NULL** | Any column of a primary key cannot be NULL. | The column of a candidate can contain a NULL value. |
| **Objective** | It is the essential part of a table or relation. | It signifies which key can be used as a primary key. |
| **Use** | It can be used as a candidate key. | It may or may not be used as a primary key. |
| **Specify** | It is not mandatory to specify a primary key for any relation. | There cannot be a relationship without specifying the candidate key. |
| **Example** | Consider a table "**student**" with columns (roll\_no., name, class, DOB, email, mobile). Here **roll\_no** column can be a primary key for the relationship because it identifies the student's records uniquely. | The **roll\_no, mobile,** and **email** columns can be candidate keys in the given table because they can uniquely identify student's records. |

# 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](https://www.javatpoint.com/mysql-tutorial).

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

63.8M

1.3K

Features of Java - Javatpoint

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:

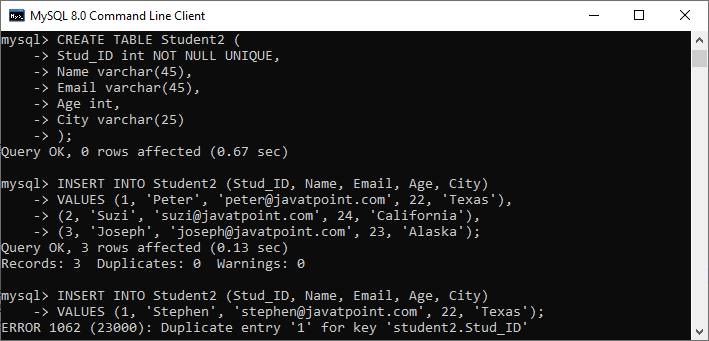
1. **CREATE** **TABLE** Student2 (
2. Stud\_ID **int** NOT NULL **UNIQUE**,
3. **Name** **varchar**(45),
4. Email **varchar**(45),
5. Age **int**,
6. City **varchar**(25)
7. );

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**

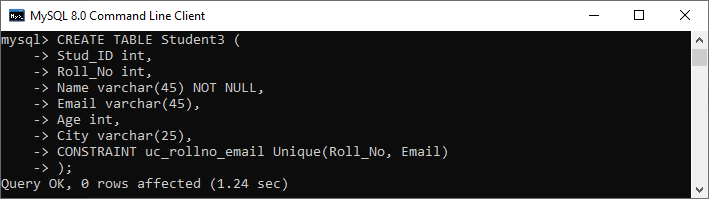
In the below output, we can see that the first [INSERT query](https://www.javatpoint.com/mysql-insert) executes correctly, but the second statement fails and gives an error that says: Duplicate entry '1' for key Stud\_ID.



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:

MySQL Unique Key

### 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**

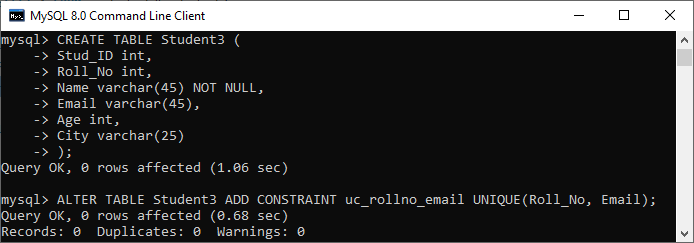
This statement creates a table "**Students3**" that have no unique key column into the table definition.

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:

MySQL Unique Key

# 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:

Play Videox

* 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:

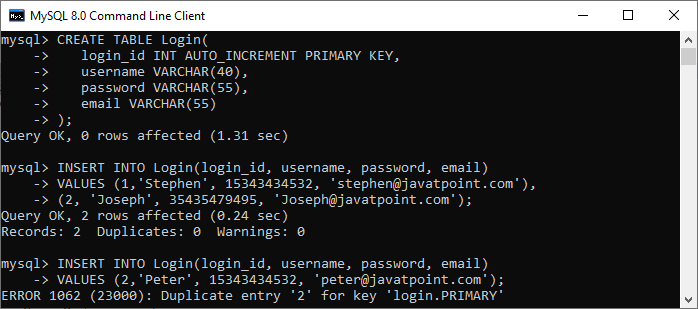
1. Mysql> **CREATE** **TABLE** Login(
2. login\_id **INT** AUTO\_INCREMENT **PRIMARY** **KEY**,
3. username **VARCHAR**(40),
4. **password** **VARCHAR**(55),
5. email **VARCHAR**(55)
6. );

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**

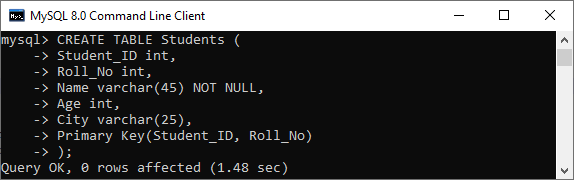
In the below output, we can see that the first insert query executes successfully. While the second insert statement fails and gives an error that says: Duplicate entry for the primary key column.



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

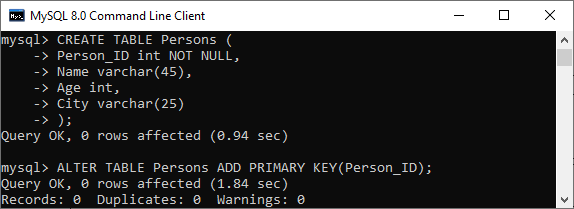
The following statement creates a table "**Persons**" that have no primary key column into the table definition.

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:

Play Videox

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**

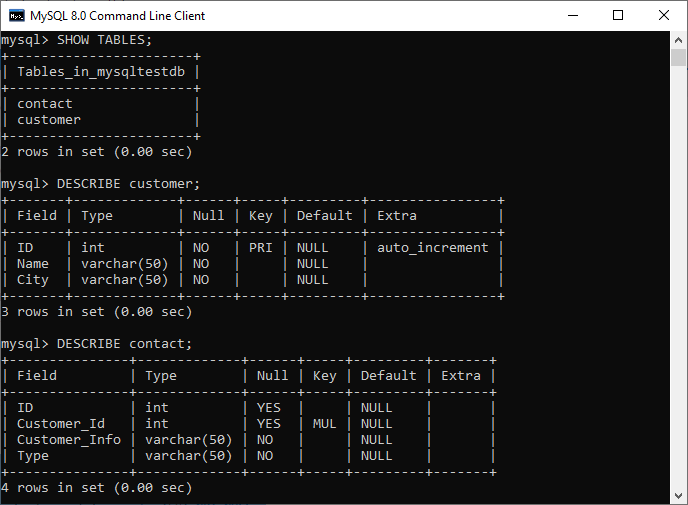
1. **CREATE** **TABLE** contact (
2. ID **INT**,
3. Customer\_Id **INT**,
4. Customer\_Info **varchar**(50) NOT NULL,
5. Type **varchar**(50) NOT NULL,
6. **INDEX** par\_ind (Customer\_Id),
7. **CONSTRAINT** fk\_customer **FOREIGN** **KEY** (Customer\_Id)
8. **REFERENCES** customer(ID)
9. **ON** **DELETE**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



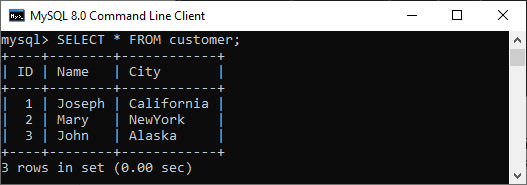
In the above output, we can see that the **PRI** in the key column of the customer table tells that this field is the primary index value. Next, the **MUL** in the key column of the contact value tells that the **Customer\_Id** field can store multiple rows with the same value.

### 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');

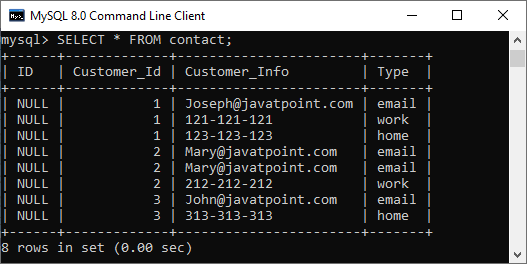
After insertion, execute the SELECT TABLE command to check the customer table data as below:



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:

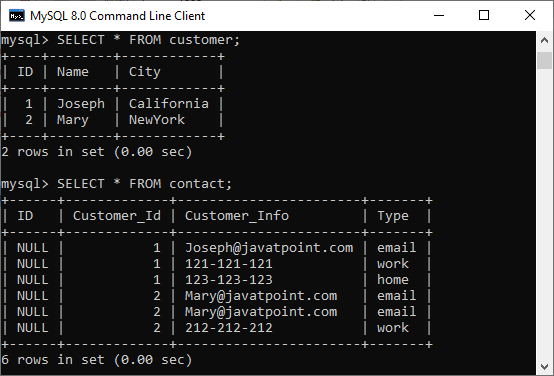
1. **FOREIGN** **KEY** (Customer\_Id) **REFERENCES** customer(ID)
2. **ON** **DELETE** **CASCADE**
3. **ON** **UPDATE** **CASCADE**.

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';

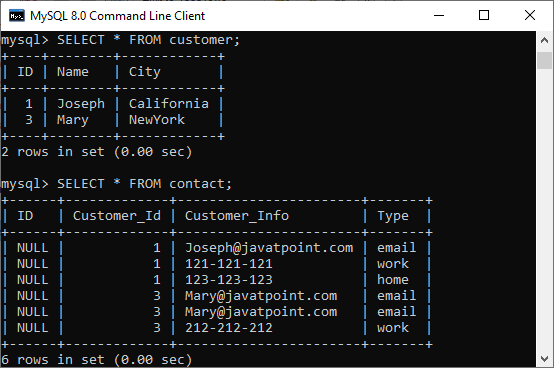
Again, if we look at our tables, we can see that both tables were changed. It means the fields with name JOHN will be removed entirely from both tables.



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. );

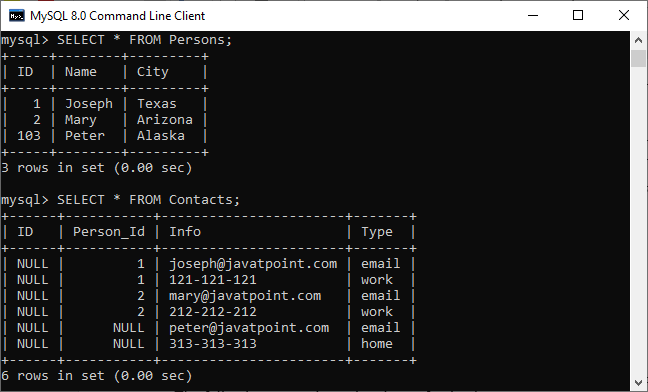
Next, we need to insert the data into both tables using the following statement:

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:

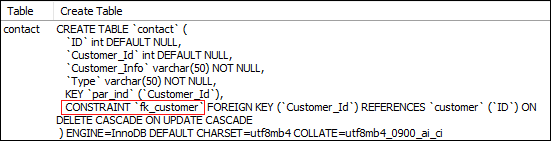
1. **ALTER** **TABLE** table\_name **DROP** **FOREIGN** **KEY** fk\_constraint\_name;

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;

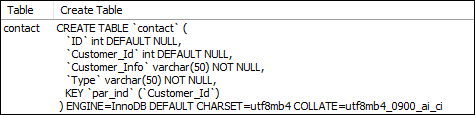
It will give the output as below where we can see that the table contact has one foreign key named fk\_customer shown in the red rectangle.



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.

44.1M

864

How to find Nth Highest Salary in SQL

### 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:

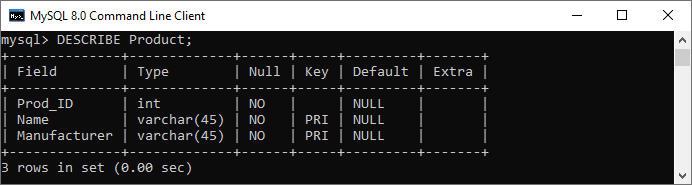
1. **CREATE** **TABLE** Product (
2. Prod\_ID **int** NOT NULL,
3. **Name** **varchar**(45),
4. Manufacturer **varchar**(45),
5. **PRIMARY** **KEY**(**Name**, Manufacturer)
6. );

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.



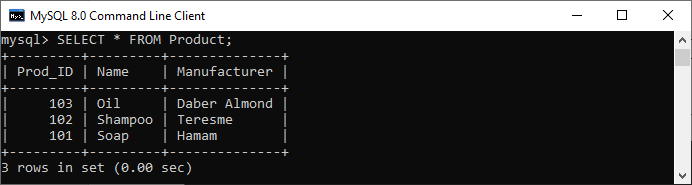
Next, we need to insert the values into this table as given below:

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:

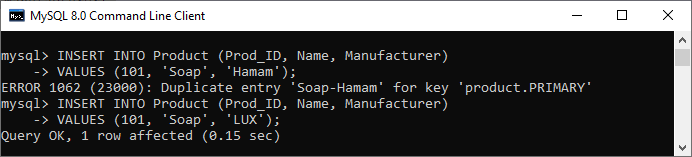


Again execute the below insert statement to understand composite key more clearly:

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](https://www.javatpoint.com/mysql-alter-table)

.

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.

# How to Create Index in MySQL

An index is a data structure that allows us to add indexes in the existing table. It enables you to improve the faster retrieval of records on a database table. It creates an **entry** for each value of the indexed columns. We use it to quickly find the record without searching each row in a database table whenever the table is accessed. We can create an index by using one or more **columns** of the table for efficient access to the records.

When a table is created with a primary key or unique key, it automatically creates a special index named **PRIMARY**. We called this index as a clustered index. All indexes other than PRIMARY indexes are known as a non-clustered index or secondary index.

## Need for Indexing in MySQL

Suppose we have a contact book that contains names and mobile numbers of the user. In this contact book, we want to find the mobile number of Martin Williamson. If the contact book is an unordered format means the name of the contact book is not sorted alphabetically, we need to go over all pages and read every name until we will not find the desired name that we are looking for. This type of searching name is known as sequential searching.

To find the name and contact of the user from table**contactbooks**, generally, we used to execute the following query:

52.2M

953

Java Try Catch

1. mysql> **SELECT** mobile\_number **FROM** contactbooks **WHERE** first\_name = 'Martin' AND last\_name = 'Taybu';

This query is very simple and easy. Although it finds the phone number and name of the user fast, the database searches entire rows of the table until it will not find the rows that you want. Assume, the contactbooks table contains **millions** of rows, then, without an index, the data retrieval takes a lot of time to find the result. In that case, the database indexing plays an important role in returning the desired result and improves the overall performance of the query.

## MySQL CREATE INDEX Statement

Generally, we create an index at the time of table creation in the database. The following statement creates a table with an index that contains two columns col2 and col3.

1. mysql> **CREATE** **TABLE** t\_index(
2. col1 **INT** **PRIMARY** **KEY**,
3. col2 **INT** NOT NULL,
4. col3 **INT** NOT NULL,
5. col4 **VARCHAR**(20),
6. **INDEX** (col2,col3)
7. );

If we want to add index in table, we will use the CREATE INDEX statement as follows:

1. mysql> **CREATE** **INDEX** [index\_name] **ON** [table\_name] (**column** names)

In this statement, **index\_name** is the name of the index, **table\_name** is the name of the table to which the index belongs, and the **column\_names** is the list of columns.

Let us add the new index for the column col4, we use the following statement:

1. mysql> **CREATE** **INDEX** ind\_1 **ON** t\_index(col4);

By default, [MySQL](https://www.javatpoint.com/mysql-tutorial)

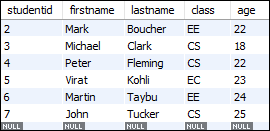
allowed index type **BTREE** if we have not specified the type of index. The following table shows the different types of an index based on the storage engine of the table.

|  |  |  |
| --- | --- | --- |
| **SN** | **Storage Engine** | **Index Type** |
| 1. | InnoDB | BTREE |
| 2. | Memory/Heap | HASH, BTREE |
| 3. | MYISAM | BTREE |

### Example

In this example, we are going to create a table **student** and perform the CREATE INDEX statement on that table.

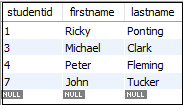
**Table Name: student**



Now, execute the following statement to return the result of the student whose **class** is **CS branch**:

1. mysql> **SELECT** studentid, firstname, lastname **FROM** student **WHERE** class = 'CS';

This statement will give the following output:



In the above table, we can see the four rows that are indicating the students whose class is the CS branch.

If you want to see how MySQL performs this query internally, execute the following statement:

1. mysql> EXPLAIN **SELECT** studentid, firstname, lastname **FROM** student **WHERE** class = 'CS';

You will get the output below. Here, MySQL scans the whole table that contains seven rows to find the student whose class is the CS branch.

How to Create Index in MySQL

Now, let us create an index for a class column using the following statement.

1. mysql> **CREATE** **INDEX** class **ON** student (class);

After executing the above statement, the index is created successfully. Now, run the below statement to see how MySQL internally performs this query.

1. mysql> EXPLAIN **SELECT** studentid, firstname, lastname **FROM** student **WHERE** class = 'CS';

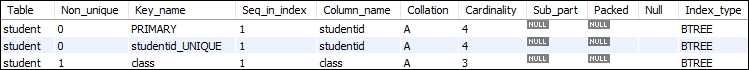
The above statement gives output, as shown below:

How to Create Index in MySQL

In this output, MySQL finds four rows from the class index without scanning the whole table. Hence, it increases the speed of retrieval of records on a database table.

If you want to **show** the indexes of a table, execute the following statement:

1. mysql> SHOW INDEXES **FROM** student;



It will give the following output.

# MySQL Drop Index

MySQL allows a DROP INDEX statement to remove the existing index from the table. To delete an index from a table, we can use the following query:

1. mysql>**DROP** **INDEX** index\_name **ON** table\_name [algorithm\_option | lock\_option];

If we want to delete an index, it requires two things:

* First, we have to specify the name of the index that we want to remove.
* Second, name of the table from which your index belongs.

The Drop Index syntax contains two optional options, which are Algorithm and Lock for reading and writing the tables during the index modifications. Let us explain both in detail:

## Algorithm Option

The algorithm\_option enables us to specify the specific algorithm for removing the index in a table. The syntax of **algorithm\_option** are as follows:

5.9K

Kindle Paperwhite (2021) VS New iPad mini 6! eReader SHOWDOWN!

1. Algorithm [=] {**DEFAULT** | INPLACE | COPY}

The Drop Index syntax supports mainly two algorithms which are INPLACE and COPY.

**COPY:** This algorithm allows us to copy one table into another new table row by row and then DROP Index statement performed on this new table. On this table, we cannot perform an INSERT and UPDATE statement for data manipulation.

**INPLACE:** This algorithm allows us to rebuild a table instead of copy the original table. We can perform all data manipulation operations on this table. On this table, [MySQL](https://www.javatpoint.com/mysql-tutorial) issues an exclusive metadata lock during the index removal.

#### Note: If you not defined the algorithm clause, MySQL uses the INPLACE algorithm. If INPLACE is not supported, it uses the COPY algorithm. The DEFAULT algorithm works the same as without using any algorithm clause with the Drop index statement.

## Lock Option

This clause enables us to control the level of concurrent reads and writes during the index removal. The syntax of **lock\_option** are as follows:

1. LOCK [=] {**DEFAULT**|NONE|SHARED|EXCLUSIVE}

In the syntax, we can see that the lock\_option contains **four modes** that are DEFAULT, NONE, SHARED, and EXCLUSIVE. Now, we are going to discuss all the modes in detail:

**SHARED:** This mode supports only concurrent reads, not concurrent writes. When the concurrent reads are not supported, it gives an error.

**DEFAULT:** This mode can have the maximum level of concurrency for a specified algorithm. It will enable concurrent reads and writes if supported otherwise enforces exclusive mode.

**NONE:** You have concurrent read and write if this mode is supported. Otherwise, it gives an error.

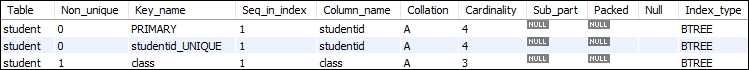
**EXCLUSIVE:** This mode enforces exclusive access.

### Example

First, execute the following command to show the indexes available in the table.

1. mysql> SHOW INDEXES **FROM** student;

It will give the following output.



In the output, we can see that there are three indexes available. Now, execute the following statement to removes the **class** index from table **student**.

1. mysql> **DROP** **INDEX** class **ON** student;

Again, execute the SHOW INDEXES statement to verify the index is removed or not. After performing this statement, we will get the following output, where only two indexes are available.

MySQL Drop Index

### Example using Algorithm and Lock

The following statement drops the **age** index form the student table using an algorithm and a lock option.

1. mysql> **DROP** **INDEX** age **ON** student ALGORITHM = INPLACE LOCK = **DEFAULT**;

## MySQL Drop PRIMARY Key Index

In some cases, the table contains a PRIMARY index that was created whenever you create a table with a primary key or unique key. In that case, we need to execute the following command because the PRIMARY is a reserved word.

1. mysql> **DROP** **INDEX** **PRIMARY** **ON** table\_name;

To remove the primary key index from the student table, execute the following statement:

1. mysql> **DROP** **INDEX** **PRIMARY** **ON** student;

# MySQL Show Indexes

We can get the index information of a table using the Show Indexes statement. This statement can be written as:

1. mysql> SHOW INDEXES **FROM** table\_name;

In the above syntax, we can see that if we want to get the index of a table, it requires to specify the **table\_name** after the FROM keyword. After the successful execution of the statement, it will return the index information of a table in the current database.

If we want to get the index information of a table in a different database or database to which you are not connected, [MySQL](https://www.javatpoint.com/mysql-tutorial) allows us to specify the database name with the Show Indexes statement. The following statement explains it more clearly:

1. mysql> SHOW INDEXES **FROM** table\_name IN database\_name;

The above statement can also be written as:

Play Videox[](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack)

1. mysql> SHOW INDEXES **FROM** database\_name.table\_name;

#### Note: It is noted that Index and Keys both are synonyms of Indexes, and IN is the synonyms of FROM keyword. Therefore, we can also write the Show Indexes statement with these synonyms as below:

1. mysql> SHOW INDEXES IN table\_name **FROM** database\_name;

OR,

1. mysql> SHOW KEYS **FROM** table\_name IN database\_name;

The SHOW INDEX query returns the following fields/information:

**Table:** It contains the name of the table.

**Non\_unique:** It returns 1 if the index contains duplicates. Otherwise, it returns 0.

**Key\_name:** It is the name of an index. If the table contains a primary key, the index name is always PRIMARY.

**Seq\_in\_index:** It is the sequence number of the column in the index that starts from 1.

**Column\_name:** It contains the name of a column.

**Collation:** It gives information about how the column is sorted in the index. It contains values where **A** represents ascending, **D** represents descending, and **Null** represents not sorted.

**Cardinality:** It gives an estimated number of unique values in the index table where the higher cardinality represents a greater chance of using indexes by MySQL.

**Sub\_part:** It is a prefix of the index. It has a NULL value if all the column of the table is indexed. When the column is partially indexed, it will return the number of indexed characters.

**Packed:** It tells how the key is packed. Otherwise, it returns NULL.

**NULL:** It contains **blank** if the column does not have NULL value; otherwise, it returns YES.

**Index\_type:** It contains the name of the index method like BTREE, HASH, RTREE, FULLTEXT, etc.

**Comment:** It contains the index information when they are not described in its column. For example, when the index is disabled, it returns disabled.

**Index\_column:** When you create an index with **comment** attributes, it contains the comment for the specified index.

**Visible:**It contains YES if the index is visible to the query optimizer, and if not, it contains NO.

**Expression:** [MySQL](https://www.javatpoint.com/mysql-tutorial) 8.0 supports **functional key parts** that affect both **expression** and **column\_name** columns. We can understand it more clearly with the below points:

* For functional parts, the expression column represents expression for the key part, and column\_name represents NULL.
* For the non-functional part, the expression represents NULL, and column\_name represents the column indexed by the key part.

### MySQL SHOW INDEX Example

Here, we are going to create a table **student\_info** that contains the student id, name, age, mobile number, and email details. Execute the following command to create a table:

1. **CREATE** **TABLE** `student\_info` (
2. `studentid` **int** NOT NULL AUTO\_INCREMENT,
3. `**name**` **varchar**(45) **DEFAULT** NULL,
4. `age` **varchar**(3) **DEFAULT** NULL,
5. `mobile` **varchar**(20) **DEFAULT** NULL,
6. `email` **varchar**(25) **DEFAULT** NULL,
7. **PRIMARY** **KEY** (`studentid`),
8. **UNIQUE** **KEY** `email\_UNIQUE` (`email`)
9. )

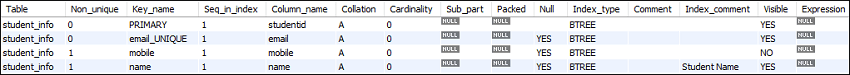
Next, we create an index on this table by the following command:

1. mysql> **CREATE** **INDEX** mobile **ON** student\_info (mobile) INVISIBLE;
3. mysql> **CREATE** **INDEX** **name** **ON** student\_info (**name**) COMMENT 'Student Name';

Now, execute the following command that returns the all index information from the student\_info table:

1. mysql> SHOW INDEXES **FROM** student\_info;

We will get the output below:



## Filter Index Information

We can filter the index information using [**where** clause](https://www.javatpoint.com/mysql-where). The following statement can be used to filter the index information:

1. Mysql> SHOW INDEXES **FROM** table\_name **where** condition;

### Example

If you want to get only **invisible** indexes of the student\_info table, execute the following command:

1. mysql> SHOW INDEXES **FROM** student\_info **WHERE** visible = 'NO';

It will give the following output:

MySQL Show Indexes

# MySQL UNIQUE INDEX

Indexing is a process to find an unordered list into an ordered list that allows us to retrieve records faster. **It creates an entry for each value that appears in the index columns**. It helps in maximizing the query's efficiency while searching on tables in MySQL. Without indexing, we need to scan the whole table to find the relevant information. The working of MySQL indexing is similar to the book index.

Generally, we use the primary key constraint to enforce the uniqueness value of one or more columns. But, we can use only one primary key for each table. So if we want to make multiple sets of columns with unique values, the primary key constraint will not be used.

[MySQL](https://www.javatpoint.com/mysql-tutorial) allows another constraint called the **UNIQUE INDEX** to enforce the uniqueness of values in one or more columns. We can create more than one UNIQUE index in a single table, which is not possible with the primary key constraint.

### Syntax

The following is a generic syntax used to create a unique index in MySQL table:

63.2M

1.3K

OOPs Concepts in Java

1. **CREATE** **UNIQUE** **INDEX** index\_name
2. **ON** table\_name (index\_column1, index\_column2,...);

MySQL allows another approach to enforcing the uniqueness value in one or more columns using the UNIQUE Key statement. We can read more information about the [UNIQUE KEY here](https://www.javatpoint.com/mysql-unique-key).

If we use a UNIQUE constraint in the table, MySQL automatically creates a UNIQUE index behind the scenes. The following statement explains how to create a unique constraint when we create a table.

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

#### NOTE: It is recommended to use the constraint name while creating a table. If we omit the constraint name, MySQL generates a name for this column automatically.

**UNIQUE Index and NULL**

NULL values in MySQL considers distinct values similar to other databases. Hence, we can store multiple NULL values in the UNIQUE index column. This feature of MySQL sometimes reported as a bug, but it is not a bug.

### MySQL UNIQUE Index Examples

Let us understand it with the help of an example. Suppose we want to manage the employee details in a database application where we need email columns unique. Execute the following statement that creates a table **"Employee\_Detail"** with a UNIQUE constraint:

1. **CREATE** **TABLE** Employee\_Detail(
2. ID **int** AUTO\_INCREMENT **PRIMARY** **KEY**,
3. **Name** **varchar**(45),
4. Email **varchar**(45),
5. Phone **varchar**(15),
6. City **varchar**(25),
7. **UNIQUE** **KEY** unique\_email (Email)
8. );

If we execute the below statement, we can see that MySQL created a UNIQUE index for **Email** column of Employee\_Detail table:

1. SHOW INDEXES **FROM** Employee\_Detail;

In the below screen, we can see that the Email column is created as a unique index.

MySQL UNIQUE INDEX

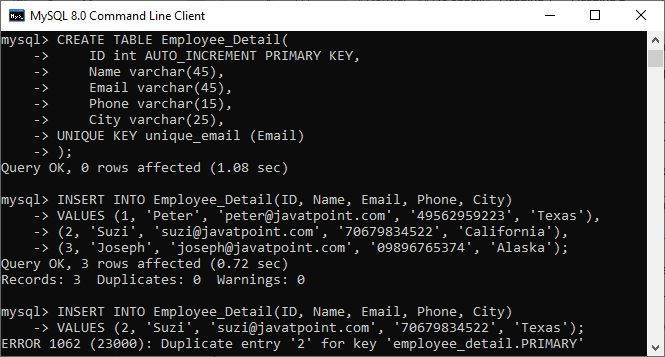
Next, we are going to insert records to the table using the following statements:

1. **INSERT** **INTO** Employee\_Detail(ID, **Name**, Email, Phone, City)
2. **VALUES** (1, 'Peter', 'peter@javatpoint.com', '49562959223', 'Texas'),
3. (2, 'Suzi', 'suzi@javatpoint.com', '70679834522', 'California'),
4. (3, 'Joseph', 'joseph@javatpoint.com', '09896765374', 'Alaska');

The above statement executed successfully because all columns are unique. If we insert a record whose email is **suzi@javatpoint.com**, we will get the duplicate error message.

1. mysql> **INSERT** **INTO** Employee\_Detail(ID, **Name**, Email, Phone, City)
2. **VALUES** (2, 'Suzi', 'suzi@javatpoint.com', '70679834522', 'Texas');

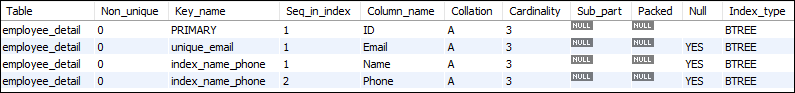
The following output explains all of the above steps more clearly:



Suppose we want the **Name** and **Phone** of the Employee\_Detail table is also unique. In this case, we will use the below statement to create a UNIQUE index for those columns:

1. **CREATE** **UNIQUE** **INDEX** index\_name\_phone
2. **ON** Employee\_Detail (**Name**, Phone);

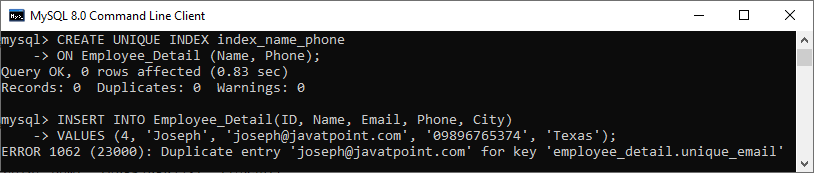
If we execute the **SHOW INDEX** statement again, we can see that MySQL created a UNIQUE index **index\_name\_phone** for name and phone columns also.



Adding this record into the table produces an error. It is because of the combination of a name and phone already exists.

1. mysql> **INSERT** **INTO** Employee\_Detail(ID, **Name**, Email, Phone, City)
2. **VALUES** (4, 'Joseph', 'joseph@javatpoint.com', '09896765374', 'Texas');

Look into this output:



# MySQL Clustered Index

An index is a separate data structure that allows us to add indexes in the existing table. It enables you to improve the faster retrieval of records on a database table. It creates an entry for each value of the indexed columns.

A clustered index is actually a table where the data for the rows are stored. It defines the order of the table data based on the key values that can be sorted in only one way. In the database, each table can have only one clustered index. In a relational database, if the table column contains a primary key or unique key, [MySQL](https://www.javatpoint.com/mysql-tutorial) allows you to create a clustered index named **PRIMARY** based on that specific column.

### Characteristics

The essential characteristics of a clustered index are as follows:

* It helps us to store data and indexes at the same time.
* It stores data in only one way based on the key values.
* Key lookup.
* They are scan and index seek.
* Clustered index always use one or more column for creating an index.

### Advantages

The main advantages of the clustered index are as follows:

* It helps us to maximize the cache hits and minimizes the page transfer.
* It is an ideal option for range or group with max, min, and count queries.
* At the start of the range, it uses a location mechanism for finding an index entry.

### Disadvantages

The main disadvantages of the clustered index are as follows:

* It contains many insert records in a non-sequential order.
* It creates many constant page splits like data pages or index pages.
* It always takes a long time to update the records.
* It needs extra work for SQL queries, such as insert, updates, and deletes.

### Clustered Index on InnoDB Tables

MySQL InnoDB table must have a clustered index. The InnoDB table uses a clustered index for optimizing the speed of most common lookups and DML (Data Manipulation Language) operations like INSERT, UPDATE, and DELETE command.

When the primary key is defined in an InnoDB table, MySQL always uses it as a clustered index named PRIMARY. If the table does not contain a primary key column, MySQL searches for the **unique key**. In the unique key, all columns are **NOT NULL** and use it as a clustered index. Sometimes, the table does not have a primary key nor unique key, then MySQL internally creates hidden clustered index **GEN\_CLUST\_INDEX** that contains the values of row id. Thus, there is only one clustered index in the InnoDB table.

The indexes other than the PRIMARY Indexes (clustered indexes) are known as a secondary index or non-clustered indexes. In the MySQL InnoDB tables, every record of the non-clustered index has primary key columns for both row and columns. MySQL uses this primary key value for searching a row in the clustered index or secondary index.

### Example

In the below statement, the PRIMARY KEY is a clustered index.

1. **CREATE** **TABLE** `student\_info` (
2. `studentid` **int** NOT NULL AUTO\_INCREMENT,
3. `**name**` **varchar**(45) **DEFAULT** NULL,
4. `age` **varchar**(3) **DEFAULT** NULL,
5. `mobile` **varchar**(20) **DEFAULT** NULL,
6. `email` **varchar**(25) **DEFAULT** NULL,
7. **PRIMARY** **KEY** (`studentid`), //clustered **index**
8. **UNIQUE** **KEY** `email\_UNIQUE` (`email`)
9. )

# Difference between MySQL Clustered and Non-Clustered Index

The difference between clustered and non-clustered index is the most famous question in the database related interviews. Both indexes have the same physical structure and are stored as a BTREE structure in the MySQL server database. In this section, we are going to explain the most popular differences between them.

Indexing in MySQL is a process that helps us to return the requested data from the table very fast. If the table does not have an index, it scans the whole table for the requested data. [MySQL](https://www.javatpoint.com/mysql-tutorial) allows two different types of Indexing:

1. Clustered Index
2. Non-Clustered Index

Let us first discuss clustered and non-clustered indexing in brief.

### What is a Clustered Index?

A clustered index is a table where the data for the rows are stored. It defines the order of the table data based on the key values that can be sorted in only one direction. In the database, each table can contains only one clustered index. In a relational database, if the table column contains a primary key or unique key, MySQL allows you to create a clustered index named **PRIMARY** based on that specific column.

63.2M

1.3K

OOPs Concepts in Java

### Example

The following example explains how the clustered index created in MySQL:

1. **CREATE** **TABLE** Student
2. ( post\_id **INT** NOT NULL AUTO\_INCREMENT, user\_id **INT** NOT NULL,
3. **CONSTRAINT** Post\_PK
4. **PRIMARY** **KEY** (user\_id, post\_id),    //clustered **index**
5. **CONSTRAINT** post\_id\_UQ
6. **UNIQUE** (post\_id)
7. ) ENGINE = InnoDB ;

### Characteristics

Following are the essential characteristics of a clustered index:

* It enables us to store data and indexes together.
* It stores data in only one way based on the key values.
* Key lookup.
* It support index scan and index seek data operations.
* Clustered index always use one or more column for creating an index.

### What is a Non-Clustered Index?

The indexes other than PRIMARY indexes (clustered indexes) called a non-clustered index. The non-clustered indexes are also known as secondary indexes. The non-clustered index and table data are both stored in different places. It is not able to sort (ordering) the table data. The non-clustered indexing is the same as a book where the content is written in one place, and the index is at a different place. MySQL allows a table to store one or more than one non-clustered index. The non-clustered indexing improves the performance of the queries which uses keys without assigning primary key.

### Example

1. //It will **create** non-clustered **index**
2. **CREATE** NonClustered **INDEX** index\_name **ON** table\_name (column\_name **ASC**);

### Characteristics

Following are the essential characteristics of a non-clustered index:

* It stores only key values.
* It allows accessing secondary data that has pointers to the physical rows.
* It helps in the operation of an index scan and seeks.
* A table can contain one or more than one non-clustered index.
* The non-clustered index row stores the value of a non-clustered key and row locator.

### Clustered VS Non-Clustered Index

Let us see some of the popular differences between clustered and non-clustered indexes through the tabular form:

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Clustered Index** | **Non-Clustered Index** |
| Definition | A clustered index is a table where the data for the rows are stored. In a relational database, if the table column contains a primary key, MySQL automatically creates a clustered index named **PRIMARY**. | The indexes other than PRIMARY indexes (clustered indexes) called a non-clustered index. The non-clustered indexes are also known as secondary indexes. |
| Use for | It can be used to sort the record and store the index in physical memory. | It creates a logical ordering of data rows and uses pointers for accessing the physical data files. |
| Size | Its size is large. | Its size is small in comparison to a clustered index. |
| Data Accessing | It accesses the data very fast. | It has slower accessing power in comparison to the clustered index. |
| Storing Method | It stores records in the leaf node of an index. | It does not store records in the leaf node of an index that means it takes extra space for data. |
| Additional Disk Space | It does not require additional reports. | It requires an additional space to store the index separately. |
| Type of Key | It uses the primary key as a clustered index. | It can work with unique constraints that act as a composite key. |
| Contains in Table | A table can only one clustered index. | A table can contain one or more than a non-clustered index. |
| Index Id | A clustered index always contains an index id of 0. | A non-clustered index always contains an index id>0. |

# MySQL Trigger

A trigger in MySQL is a set of SQL statements that reside in a system catalog. **It is a special type of stored procedure that is invoked automatically in response to an event**. Each trigger is associated with a table, which is activated on any DML statement such as **INSERT, UPDATE**, or **DELETE**.

A trigger is called a special procedure because it cannot be called directly like a stored procedure. The main difference between the trigger and procedure is that a trigger is called automatically when a data modification event is made against a table. In contrast, a stored procedure must be called explicitly.

Generally, **triggers are of two types** according to the [SQL](https://www.javatpoint.com/sql-tutorial) standard: row-level triggers and statement-level triggers.

**Row-Level Trigger:** It is a trigger, which is activated for each row by a triggering statement such as insert, update, or delete. For example, if a table has inserted, updated, or deleted multiple rows, the row trigger is fired automatically for each row affected by the [insert](https://www.javatpoint.com/mysql-insert), [update](https://www.javatpoint.com/mysql-update), or [delete statement](https://www.javatpoint.com/mysql-delete).

**Statement-Level Trigger:** It is a trigger, which is fired once for each event that occurs on a table regardless of how many rows are inserted, updated, or deleted.

#### NOTE: We should know that MySQL doesn't support statement-level triggers. It provides supports for row-level triggers only.

### Why we need/use triggers in MySQL?

We need/use triggers in MySQL due to the following features:

* Triggers help us to enforce business rules.
* Triggers help us to validate data even before they are inserted or updated.
* Triggers help us to keep a log of records like maintaining audit trails in tables.
* SQL triggers provide an alternative way to check the integrity of data.
* Triggers provide an alternative way to run the scheduled task.
* Triggers increases the performance of SQL queries because it does not need to compile each time the query is executed.
* Triggers reduce the client-side code that saves time and effort.
* Triggers help us to scale our application across different platforms.
* Triggers are easy to maintain.

### Limitations of Using Triggers in MySQL

* MySQL triggers do not allow to use of all validations; they only provide extended validations. **For example**, we can use the NOT NULL, UNIQUE, CHECK and FOREIGN KEY constraints for simple validations.
* Triggers are invoked and executed invisibly from the client application. Therefore, it isn't easy to troubleshoot what happens in the database layer.
* Triggers may increase the overhead of the database server.

### Types of Triggers in MySQL?

We can define the maximum six types of actions or events in the form of triggers:

1. [**Before Insert**](https://www.javatpoint.com/mysql-before-insert-trigger)**:** It is activated before the insertion of data into the table.
2. [**After Insert**](https://www.javatpoint.com/mysql-after-insert-trigger)**:** It is activated after the insertion of data into the table.
3. [**Before Update**](https://www.javatpoint.com/mysql-before-update-trigger)**:** It is activated before the update of data in the table.
4. [**After Update**](https://www.javatpoint.com/mysql-after-update-trigger)**:** It is activated after the update of the data in the table.
5. [**Before Delete**](https://www.javatpoint.com/mysql-before-delete-trigger)**:** It is activated before the data is removed from the table.
6. [**After Delete**](https://www.javatpoint.com/mysql-after-delete-trigger)**:** It is activated after the deletion of data from the table.

When we use a statement that does not use INSERT, UPDATE or DELETE query to change the data in a table, the triggers associated with the trigger will not be invoked.

### Naming Conventions

Naming conventions are the set of rules that we follow to give appropriate unique names. It saves our time to keep the work organize and understandable. Therefore, **we must use a unique name for each trigger associated with a table**. However, it is a good practice to have the same trigger name defined for different tables.

The following naming convention should be used to name the trigger in [MySQL](https://www.javatpoint.com/mysql-tutorial):

1. (BEFOR | **AFTER**) table\_name (**INSERT** | **UPDATE** | **DELETE**)

Thus,

**Trigger Activation Time:** BEFORE | AFTER

**Trigger Event:** INSERT | UPDATE | DELETE

### How to create triggers in MySQL?

We can use the **CREATE TRIGGER** statement for creating a new trigger in MySQL. Below is the syntax of creating a trigger in MySQL:

1. **CREATE** **TRIGGER** trigger\_name
2. (**AFTER** | BEFORE) (**INSERT** | **UPDATE** | **DELETE**)
3. **ON** table\_name **FOR** EACH ROW
4. **BEGIN**
5. --variable declarations
6. --trigger code
7. **END**;

# MySQL Create Trigger

In this article, we are going to learn how to create the first trigger in MySQL. We can create a new trigger in MySQL by using the CREATE TRIGGER statement. It is to ensure that we have trigger privileges while using the CREATE TRIGGER command. The following is the basic syntax to create a trigger:

1. **CREATE** **TRIGGER** trigger\_name  trigger\_time trigger\_event
2. **ON** table\_name **FOR** EACH ROW
3. **BEGIN**
4. --variable declarations
5. --trigger code
6. **END**;

### Parameter Explanation

The create trigger syntax contains the following parameters:

**trigger\_name:** It is the name of the trigger that we want to create. It must be written after the CREATE [TRIGGER statement](https://www.javatpoint.com/mysql-trigger). It is to make sure that the trigger name should be unique within the schema.

**trigger\_time:** It is the trigger action time, which should be either BEFORE or AFTER. It is the required parameter while defining a trigger. It indicates that the trigger will be invoked before or after each row modification occurs on the table.

**trigger\_event:** It is the type of operation name that activates the trigger. It can be either [INSERT](https://www.javatpoint.com/mysql-insert), [UPDATE](https://www.javatpoint.com/mysql-update), or [DELETE](https://www.javatpoint.com/mysql-delete) operation. The trigger can invoke only one event at one time. If we want to define a trigger which is invoked by multiple events, it is required to define multiple triggers, and one for each event.

**table\_name:** It is the name of the table to which the trigger is associated. It must be written after the ON keyword. If we did not specify the table name, a trigger would not exist.

**BEGIN END Block:** Finally, we will specify the statement for execution when the trigger is activated. If we want to execute multiple statements, we will use the BEGIN END block that contains a set of queries to define the logic for the trigger.

The trigger body can access the column's values, which are affected by the DML statement. The **NEW** and **OLD** modifiers are used to distinguish the column values **BEFORE** and **AFTER** the execution of the DML statement. We can use the column name with NEW and OLD modifiers as **OLD.col\_name** and **NEW.col\_name**. The OLD.column\_name indicates the column of an existing row before the updation or deletion occurs. NEW.col\_name indicates the column of a new row that will be inserted or an existing row after it is updated.

**For example**, suppose we want to update the column name **message\_info** using the trigger. In the trigger body, we can access the column value before the update as **OLD.message\_info** and the new value **NEW.message\_info**.

We can understand the availability of OLD and NEW modifiers with the below table:

|  |  |  |
| --- | --- | --- |
| **Trigger Event** | **OLD** | **NEW** |
| INSERT | No | Yes |
| UPDATE | Yes | Yes |
| ELETE | Yes | No |

### MySQL Trigger Example

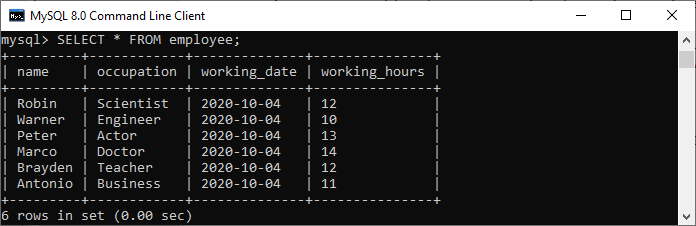
Let us start creating a trigger in [MySQL](https://www.javatpoint.com/mysql-tutorial) that makes modifications in the employee table. First, we will create a new table named **employee** by executing the below statement:

1. **CREATE** **TABLE** employee(
2. **name** **varchar**(45) NOT NULL,
3. occupation **varchar**(35) NOT NULL,
4. working\_date **date**,
5. working\_hours **varchar**(10)
6. );

Next, execute the below statement to **fill the records** into the employee table:

1. **INSERT** **INTO** employee **VALUES**
2. ('Robin', 'Scientist', '2020-10-04', 12),
3. ('Warner', 'Engineer', '2020-10-04', 10),
4. ('Peter', 'Actor', '2020-10-04', 13),
5. ('Marco', 'Doctor', '2020-10-04', 14),
6. ('Brayden', 'Teacher', '2020-10-04', 12),
7. ('Antonio', 'Business', '2020-10-04', 11);

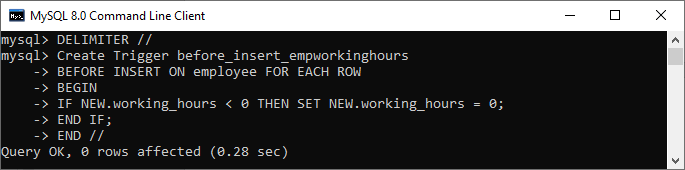
Next, execute the [**SELECT statement**](https://www.javatpoint.com/mysql-select) to verify the inserted record:



Next, we will create a [**BEFORE INSERT trigger**](https://www.javatpoint.com/mysql-before-insert-trigger). This trigger is invoked automatically insert the **working\_hours = 0** if someone tries to insert **working\_hours < 0**.

1. mysql> DELIMITER //
2. mysql> **Create** **Trigger** before\_insert\_empworkinghours
3. BEFORE **INSERT** **ON** employee **FOR** EACH ROW
4. **BEGIN**
5. IF NEW.working\_hours < 0 **THEN** **SET** NEW.working\_hours = 0;
6. **END** IF;
7. **END** //

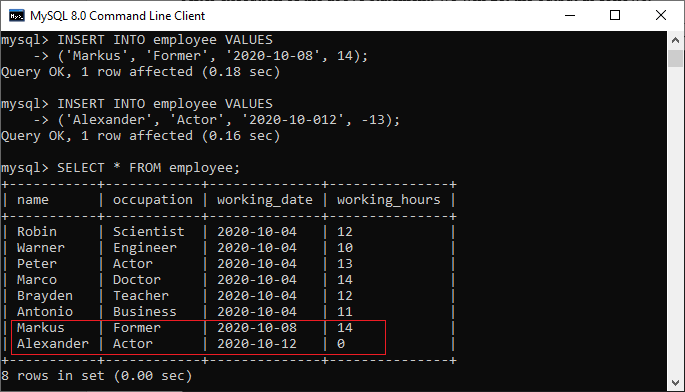
If the trigger is created successfully, we will get the output as follows:



Now, we can use the following statements to invoke this trigger:

1. mysql> **INSERT** **INTO** employee **VALUES**
2. ('Markus', 'Former', '2020-10-08', 14);
4. mysql> **INSERT** **INTO** employee **VALUES**
5. ('Alexander', 'Actor', '2020-10-012', -13);

After execution of the above statement, we will get the output as follows:



In this output, we can see that on inserting the negative values into the working\_hours column of the table will automatically fill the zero value by a trigger.

# MySQL BEFORE INSERT TRIGGER

Before Insert Trigger in MySQL is invoked automatically whenever an insert operation is executed. In this article, we are going to learn how to create a before insert trigger with its syntax and example.

### Syntax

The following is the syntax to create a BEFORE INSERT [trigger in MySQL](https://www.javatpoint.com/mysql-trigger):

1. **CREATE** **TRIGGER** trigger\_name
2. BEFORE **INSERT**
3. **ON** table\_name **FOR** EACH ROW
4. Trigger\_body ;

The BEFORE INSERT trigger syntax parameter can be explained as below:

* First, we will specify the **name of the trigger** that we want to create. It should be unique within the schema.
* Second, we will specify the **trigger action time**, which should be BEFORE INSERT. This trigger will be invoked before each row modifications occur on the table.
* Third, we will specify the **name of a table** to which the trigger is associated. It must be written after the ON keyword. If we did not specify the table name, a trigger would not exist.
* Finally, we will specify the statement for execution when the trigger is activated.

If we want to execute multiple statements, we will use the **BEGIN END** block that contains a set of queries to define the logic for the trigger. See the below syntax:

Play Videox[](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack)

1. DELIMITER $$
2. **CREATE** **TRIGGER** trigger\_name BEFORE **INSERT**
3. **ON** table\_name **FOR** EACH ROW
4. **BEGIN**
5. variable declarations
6. **trigger** code
7. **END**$$
8. DELIMITER ;

### Restrictions

* We can access and change the **NEW** values only in a BEFORE INSERT trigger.
* We cannot access the **OLD** If we try to access the OLD values, we will get an error because OLD values do not exist.
* We cannot create a BEFORE INSERT trigger on a **VIEW**.

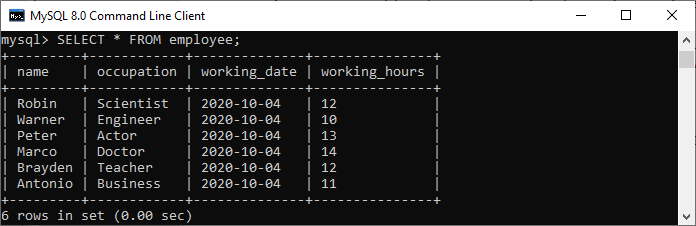
### BEFORE INSERT Trigger Example

Let us understand how to create a BEFORE INSERT trigger using the [CREATE TRIGGER statement](https://www.javatpoint.com/mysql-create-trigger) in [MySQL](https://www.javatpoint.com/mysql-tutorial) with an example.

Suppose we have created a table named **employee** as follows:

1. **CREATE** **TABLE** employee(
2. **name** **varchar**(45) NOT NULL,
3. occupation **varchar**(35) NOT NULL,
4. working\_date **date**,
5. working\_hours **varchar**(10)
6. );

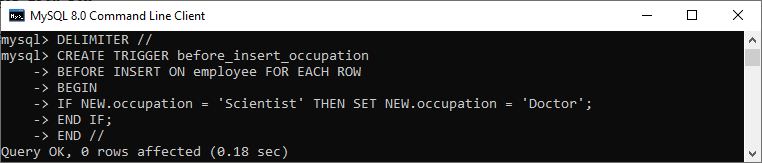
Next, we will insert some records into the employee table and then execute the [SELECT statement](https://www.javatpoint.com/mysql-select) to see the table data as follows:



Next, we will use a **CREATE TRIGGER** statement to create a BEFORE INSERT trigger. This trigger is invoked automatically that inserts the **occupation = 'Leader'** if someone tries to insert the **occupation = 'Scientist'**.

1. mysql> DELIMITER //
2. mysql> **Create** **Trigger** before\_insert\_occupation
3. BEFORE **INSERT** **ON** employee **FOR** EACH ROW
4. **BEGIN**
5. IF NEW.occupation = 'Scientist' **THEN** **SET** NEW.occupation = 'Doctor';
6. **END** IF;
7. **END** //

If the trigger is created successfully, we will get the output as follows:

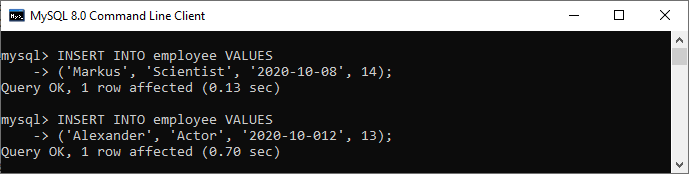


### How to call the BEFORE INSERT trigger?

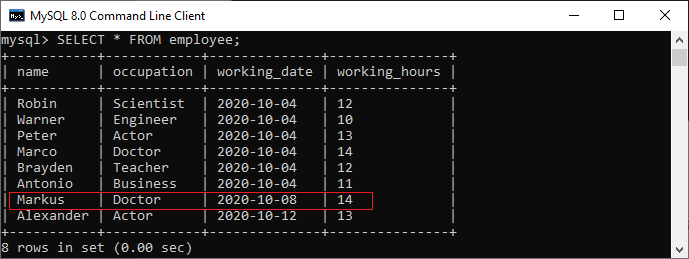
We can use the following statements to invoke the above-created trigger:

1. mysql> **INSERT** **INTO** employee **VALUES**
2. ('Markus', 'Scientist', '2020-10-08', 14);
4. mysql> **INSERT** **INTO** employee **VALUES**
5. ('Alexander', 'Actor', '2020-10-012', 13);

After execution of the above statement, we will get the output as follows:



Execute the SELECT statement to verify the output:



In this output, we can see that on inserting the occupation column values as 'Scientist', the table will automatically fill the 'Doctor' value by invoking a trigger.

# MySQL AFTER INSERT Trigger

After Insert Trigger in MySQL is invoked automatically whenever an insert event occurs on the table. In this article, we are going to learn how to create an after insert trigger with its syntax and example.

### Syntax

The following is the syntax to create an **AFTER INSERT** [trigger in MySQL](https://www.javatpoint.com/mysql-trigger):

1. **CREATE** **TRIGGER** trigger\_name
2. **AFTER** **INSERT**
3. **ON** table\_name **FOR** EACH ROW
4. trigger\_body ;

The AFTER INSERT trigger syntax parameter can be explained as below:

* First, we will specify the **name of the trigger** that we want to create. It should be unique within the schema.
* Second, we will specify the **trigger action time**, which should be AFTER INSERT clause to invoke the trigger.
* Third, we will specify the **name of a table** to which the trigger is associated. It must be written after the ON keyword. If we did not specify the table name, a trigger would not exist.
* Finally, we will specify the **trigger body** that contains one or more statements for execution when the trigger is activated.

If we want to execute multiple statements, we will use the **BEGIN END** block that contains a set of SQL queries to define the logic for the trigger. See the below syntax:

49.4M

884

Prime Ministers of India | List of Prime Minister of India (1947-2020)

1. DELIMITER $$
2. **CREATE** **TRIGGER** trigger\_name **AFTER** **INSERT**
3. **ON** table\_name **FOR** EACH ROW
4. **BEGIN**
5. variable declarations
6. **trigger** code
7. **END**$$
8. DELIMITER ;

### Restrictions

* We can access the **NEW** values but **cannot change them** in an AFTER INSERT trigger.
* We cannot access the **OLD** If we try to access the OLD values, we will get an error because there is no OLD on the INSERT trigger.
* We cannot create the AFTER INSERT trigger on a **VIEW**.

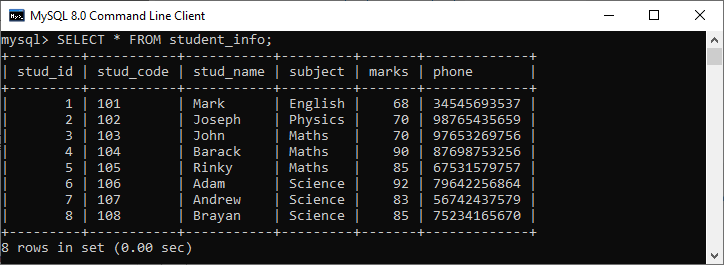
### AFTER INSERT Trigger Example

Let us understand how to create an AFTER INSERT trigger using the [**CREATE TRIGGER**](https://www.javatpoint.com/mysql-create-trigger) statement in [MySQL](https://www.javatpoint.com/mysql-tutorial) with an example.

Suppose we have created a table named "**student\_info**" as follows:

1. **CREATE** **TABLE** student\_info (
2. stud\_id **int** NOT NULL,
3. stud\_code **varchar**(15) **DEFAULT** NULL,
4. stud\_name **varchar**(35) **DEFAULT** NULL,
5. subject **varchar**(25) **DEFAULT** NULL,
6. marks **int** **DEFAULT** NULL,
7. phone **varchar**(15) **DEFAULT** NULL,
8. **PRIMARY** **KEY** (stud\_id)
9. )

Next, we will insert some records into this table and then execute the [SELECT statement](https://www.javatpoint.com/mysql-select) to see the table data as follows:



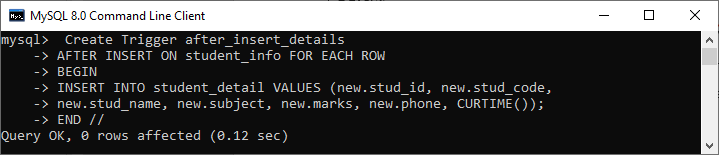
Again, we will create a new table named **"student\_detail"** as follows:

1. **CREATE** **TABLE** student\_detail (
2. stud\_id **int** NOT NULL,
3. stud\_code **varchar**(15) **DEFAULT** NULL,
4. stud\_name **varchar**(35) **DEFAULT** NULL,
5. subject **varchar**(25) **DEFAULT** NULL,
6. marks **int** **DEFAULT** NULL,
7. phone **varchar**(15) **DEFAULT** NULL,
8. Lasinserted **Time**,
9. **PRIMARY** **KEY** (stud\_id)
10. );

Next, we will use a CREATE TRIGGER statement to create an **after\_insert\_details** trigger on the **student\_info** table. This trigger will be fired after an insert operation is performed on the table.

1. mysql> DELIMITER //
2. mysql> **Create** **Trigger** after\_insert\_details
3. **AFTER** **INSERT** **ON** student\_info **FOR** EACH ROW
4. **BEGIN**
5. **INSERT** **INTO** student\_detail **VALUES** (new.stud\_id, new.stud\_code,
6. new.stud\_name, new.subject, new.marks, new.phone, CURTIME());
7. **END** //

If the trigger is created successfully, we will get the output as follows:



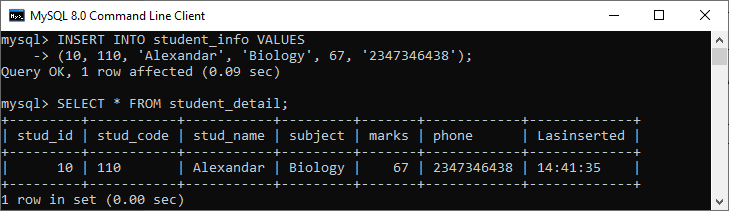
### How to call the AFTER INSERT trigger?

We can use the following statements to invoke the above-created trigger:

1. mysql> **INSERT** **INTO** student\_info **VALUES**
2. (10, 110, 'Alexandar', 'Biology', 67, '2347346438');

The table that has been modified after the update query executes is student\_detail. We can verify it by using the SELECT statement as follows:

1. mysql> **SELECT** \* **FROM** student\_detail;



In this output, we can see that on inserting values into the student\_info table, the student\_detail table will automatically fill the records by invoking a trigger.

# MySQL BEFORE UPDATE Trigger

BEFORE UPDATE Trigger in MySQL is invoked automatically whenever an update operation is fired on the table associated with the trigger. In this article, we are going to learn how to create a before update trigger with its syntax and example.

### Syntax

The following is the syntax to create a BEFORE UPDATE trigger in MySQL:

1. **CREATE** **TRIGGER** trigger\_name
2. BEFORE **UPDATE**
3. **ON** table\_name **FOR** EACH ROW
4. trigger\_body ;

The BEFORE UPDATE trigger syntax parameter are explained as below:

* First, we will specify the **trigger name** that we want to create. It should be unique within the schema.
* Second, we will specify the **trigger action time**, which should be BEFORE UPDATE. This trigger will be invoked before each row of alterations occurs on the table.
* Third, we will specify the name of a table to which the trigger is associated. It must be written after the **ON keyword**. If we did not specify the table name, a trigger would not exist.
* Finally, we will specify the **trigger body** that contains a statement for execution when the trigger is activated.

If we want to execute multiple statements, we will use the **BEGIN END** block that contains a set of queries to define the logic for the trigger. See the below syntax:

Play Videox[](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack)

1. DELIMITER $$
2. **CREATE** **TRIGGER** trigger\_name BEFORE **UPDATE**
3. **ON** table\_name **FOR** EACH ROW
4. **BEGIN**
5. variable declarations
6. **trigger** code
7. **END**$$
8. DELIMITER ;

### Restrictions

* We cannot update the OLD values in a BEFORE UPDATE trigger.
* We can change the NEW values.
* We cannot create a BEFORE UPDATE trigger on a VIEW.

### BEFORE UPDATE Trigger Example

Let us understand how to create a BEFORE UPDATE trigger using the [CREATE TRIGGER statement in MySQL](https://www.javatpoint.com/mysql-create-trigger) with an example.

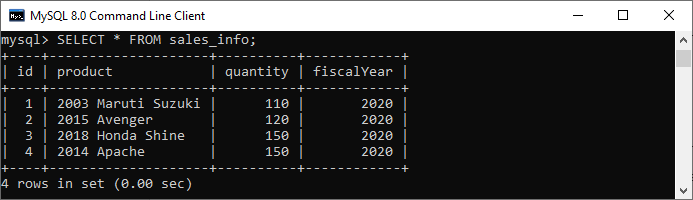
Suppose we have created a table named **sales\_info** as follows:

1. **CREATE** **TABLE** sales\_info (
2. id **INT** AUTO\_INCREMENT,
3. product **VARCHAR**(100) NOT NULL,
4. quantity **INT** NOT NULL **DEFAULT** 0,
5. fiscalYear **SMALLINT** NOT NULL,
6. **CHECK**(fiscalYear BETWEEN 2000 and 2050),
7. **CHECK** (quantity >=0),
8. **UNIQUE**(product, fiscalYear),
9. **PRIMARY** **KEY**(id)
10. );

Next, we will insert some records into the sales\_info table as follows:

1. **INSERT** **INTO** sales\_info(product, quantity, fiscalYear)
2. **VALUES**
3. ('2003 Maruti Suzuki',110, 2020),
4. ('2015 Avenger', 120,2020),
5. ('2018 Honda Shine', 150,2020),
6. ('2014 Apache', 150,2020);

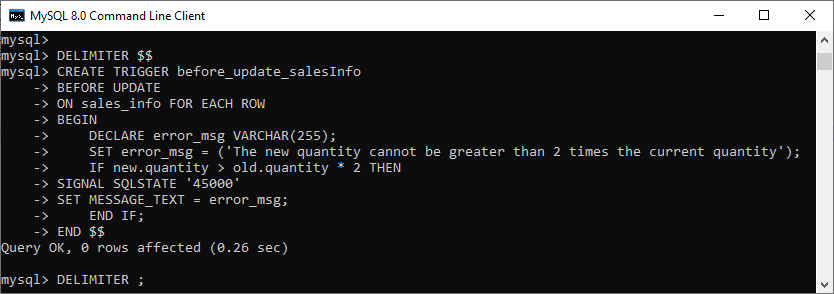
Then, execute the **SELECT statement** to see the table data as follows:



Next, we will use a **CREATE TRIGGER** statement to create a BEFORE UPDATE trigger. This trigger is invoked automatically before an update event occurs in the table.

1. DELIMITER $$
3. **CREATE** **TRIGGER** before\_update\_salesInfo
4. BEFORE **UPDATE**
5. **ON** sales\_info **FOR** EACH ROW
6. **BEGIN**
7. **DECLARE** error\_msg **VARCHAR**(255);
8. **SET** error\_msg = ('The new quantity cannot be greater than 2 times the current quantity');
9. IF new.quantity > old.quantity \* 2 **THEN**
10. SIGNAL SQLSTATE '45000'
11. **SET** MESSAGE\_TEXT = error\_msg;
12. **END** IF;
13. **END** $$
15. DELIMITER ;

If the trigger is created successfully, we will get the output as follows:



The trigger produces an error message and stops the updation if we update the value in the quantity column to a new value two times greater than the current value.

Let us understand the created trigger in details:

First, we have specified the trigger name as befor\_update\_salesInfo in the CREATE TRIGGER clause. Second, specify the triggering event and then the table name on which the trigger is associated. Third, we have declared a variable and set its value. Finally, we have specified the trigger body that checks if the new value is two times greater than the old value and then raises an error.

### How to call the BEFORE UPDATE trigger?

First, we can use the following statements that update the quantity of the row whose id = 2:

1. mysql> **UPDATE** sales\_info **SET** quantity = 125 **WHERE** id = 2;

This statement works well because it does not violate the rule. Next, we will execute the below statements that update the quantity of the row as 600 whose id = 2

1. mysql> **UPDATE** sales\_info **SET** quantity = 600 **WHERE** id = 2;

It will give the error as follows because it violates the rule. See the below output.

# MySQL AFTER UPDATE TRIGGER

The AFTER UPDATE trigger in MySQL is invoked automatically whenever an UPDATE event is fired on the table associated with the triggers. In this article, we are going to learn how to create an AFTER UPDATE trigger with its syntax and example.

### Syntax

The following is the syntax to create an **AFTER UPDATE** trigger in MySQL:

1. **CREATE** **TRIGGER** trigger\_name
2. **AFTER** **UPDATE**
3. **ON** table\_name **FOR** EACH ROW
4. trigger\_body ;

We can explain the parameters of AFTER UPDATE trigger syntax as below:

* First, we will specify the **trigger name** that we want to create. It should be unique within the schema.
* Second, we will specify the **trigger action time**, which should be AFTER UPDATE. This trigger will be invoked after each row of alterations occurs on the table.
* Third, we will specify the **table name** to which the trigger is associated. It must be written after the **ON** If we did not specify the table name, a trigger would not exist.
* Finally, we will specify the **trigger body** that contains a statement for execution when the trigger is activated.

If we want to execute more than one statement, we will use the **BEGIN END** block that contains a set of SQL queries to define the logic for the [trigger](https://www.javatpoint.com/mysql-trigger). See the below syntax:

Play Videox[](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack)

1. DELIMITER $$
2. **CREATE** **TRIGGER** trigger\_name **AFTER** **UPDATE**
3. **ON** table\_name **FOR** EACH ROW
4. **BEGIN**
5. variable declarations
6. **trigger** code
7. **END**$$
8. DELIMITER ;

### Restrictions

* We can access the OLD rows but cannot update them.
* We can access the NEW rows but cannot update them.
* We cannot create an AFTER UPDATE trigger on a **VIEW**.

### AFTER UPDATE Trigger Example

Let us understand how to create an AFTER UPDATE trigger using the [CREATE TRIGGER statement in MySQL](https://www.javatpoint.com/mysql-create-trigger) with an example.

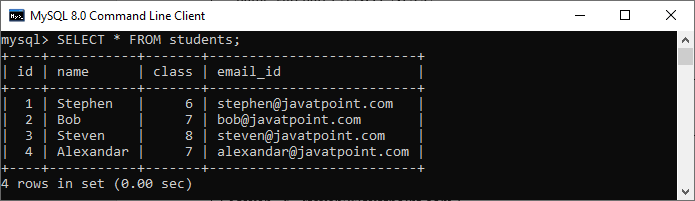
Suppose we have created a table named **students** to store the student's information as follows:

1. mysql> **CREATE** **TABLE** students(
2. id **int** NOT NULL AUTO\_INCREMENT,
3. **name** **varchar**(45) NOT NULL,
4. class **int** NOT NULL,
5. email\_id **varchar**(65) NOT NULL,
6. **PRIMARY** **KEY** (id)
7. );

Next, we will insert some records into this table using the below statement:

1. **INSERT** **INTO** students (**name**, class, email\_id)
2. **VALUES** ('Stephen', 6, 'stephen@javatpoint.com'),
3. ('Bob', 7, 'bob@javatpoint.com'),
4. ('Steven', 8, 'steven@javatpoint.com'),
5. ('Alexandar', 7, 'alexandar@javatpoint.com');

Execute the [**SELECT**](https://www.javatpoint.com/mysql-select) query to see the table data.

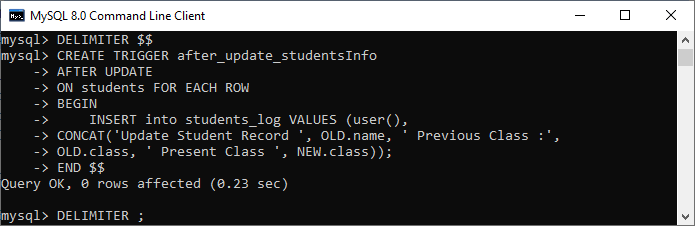


Third, we will create another table named **students\_log** that keeps the updated information in the selected user.

1. mysql> **CREATE** **TABLE** students\_log(
2. user **varchar**(45) NOT NULL,
3. descreptions **varchar**(65) NOT NULL
4. );

We will then create an AFTER UPDATE **trigger that promotes all students in the next class**, i.e., 6 will be 7, 7 will be 8, and so on. Whenever an updation is performed on a single row in the "**students**" table, a new row will be inserted in the **"students\_log**" table. This table keeps the **current user id** and a **description** regarding the current update. See the below trigger code.

1. DELIMITER $$
3. **CREATE** **TRIGGER** after\_update\_studentsInfo
4. **AFTER** **UPDATE**
5. **ON** students **FOR** EACH ROW
6. **BEGIN**
7. **INSERT** **into** students\_log **VALUES** (user(),
8. CONCAT('Update Student Record ', OLD.**name**, ' Previous Class :',
9. OLD.class, ' Present Class ', NEW.class));
10. **END** $$
12. DELIMITER ;



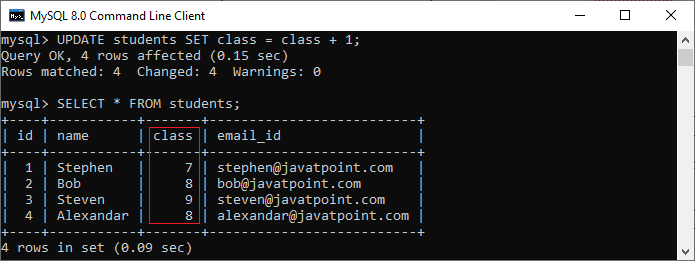
In this trigger, we have first specified the trigger name **after\_update\_studentsInfo**. Then, specify the triggering event. Third, we have specified the table name on which the trigger is associated. Finally, we have written the trigger logic inside the trigger body that performs updation in the "students" table and keeps the log information in the "students\_log" table.

### How to call the AFTER UPDATE trigger?

First, we will update the "students" table using the following statements that invoke the above-created trigger:

1. mysql> **UPDATE** students **SET** class = class + 1;

Next, we will query data from the **students** and **students\_log table**. We can see that table has been updated after the execution of the query. See the below output:



Again, we will query data from the students\_log table that keeps the current user id and a description regarding the current update. See the below output:

# MySQL BEFORE DELETE Trigger

BEFORE DELETE Trigger in MySQL is invoked automatically whenever a delete operation is fired on the table. In this article, we are going to learn how to create a before delete trigger with its syntax and example.

### Syntax

The following is the syntax to create a BEFORE DELETE trigger in MySQL:

1. **CREATE** **TRIGGER** trigger\_name
2. BEFORE **DELETE**
3. **ON** table\_name **FOR** EACH ROW
4. Trigger\_body ;

The BEFORE DELETE trigger syntax parameter can be explained as below:

* First, we will specify the name of the trigger that we want to create. It should be unique within the schema.
* Second, we will specify the trigger action time, which should be BEFORE DELETE. This trigger will be invoked before each row of alterations occurs on the table.
* Third, we will specify the name of a table to which the trigger is associated. It must be written after the ON keyword. If we did not specify the table name, a trigger would not exist.
* Finally, we will specify the statement for execution when the trigger is activated.

If we want to execute multiple statements, we will use the BEGIN END block that contains a set of queries to define the logic for the trigger. See the below syntax:

1. DELIMITER $$
2. **CREATE** **TRIGGER** trigger\_name BEFORE **DELETE**
3. **ON** table\_name **FOR** EACH ROW
4. **BEGIN**
5. variable declarations
6. **trigger** code
7. **END**$$
8. DELIMITER ;

### Restrictions

* We can access the OLD rows but cannot update them in a BEFORE DELETE trigger.
* We cannot access the NEW rows. It is because there are no new row exists.
* We cannot create a BEFORE DELETE trigger on a VIEW.

### BEFORE DELETE Trigger Example

Let us understand how to create a BEFORE DELETE trigger using the [CREATE TRIGGER statement in MySQL](https://www.javatpoint.com/mysql-create-trigger) with an example.

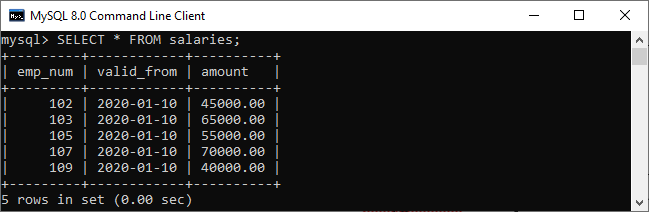
Suppose we have created a table named salaries to store the salary information of an employee as follows:

1. **CREATE** **TABLE** salaries (
2. emp\_num **INT** **PRIMARY** **KEY**,
3. valid\_from **DATE** NOT NULL,
4. amount **DEC**(8 , 2 ) NOT NULL **DEFAULT** 0
5. );

Next, we will insert some records into this table using the below statement:

1. **INSERT** **INTO** salaries (emp\_num, valid\_from, amount)
2. **VALUES**
3. (102, '2020-01-10', 45000),
4. (103, '2020-01-10', 65000),
5. (105, '2020-01-10', 55000),
6. (107, '2020-01-10', 70000),
7. (109, '2020-01-10', 40000);

Execute the SELECT query to see the table data.

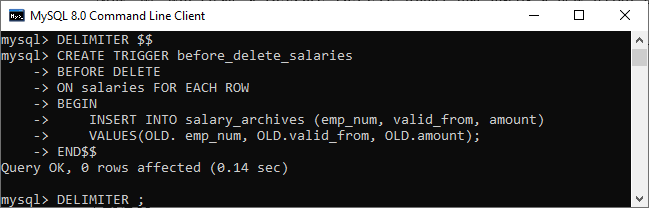


Third, we will create another table named salary\_archives that keeps the information of deleted salary.

1. **CREATE** **TABLE** salary\_archives (
2. id **INT** **PRIMARY** **KEY** AUTO\_INCREMENT,
3. emp\_num **INT**,
4. valid\_from **DATE** NOT NULL,
5. amount **DEC**(18 , 2 ) NOT NULL **DEFAULT** 0,
6. deleted\_time **TIMESTAMP** **DEFAULT** NOW()
7. );

We will then create a BEFORE DELETE trigger that inserts a new record into the salary\_archives table before a row is deleted from the salaries table.

1. DELIMITER $$
3. **CREATE** **TRIGGER** before\_delete\_salaries
4. BEFORE **DELETE**
5. **ON** salaries **FOR** EACH ROW
6. **BEGIN**
7. **INSERT** **INTO** salary\_archives (emp\_num, valid\_from, amount)
8. **VALUES**(OLD. emp\_num, OLD.valid\_from, OLD.amount);
9. **END**$$
11. DELIMITER ;



In this trigger, we have first specified the trigger name before\_delete\_salaries. Then, specify the triggering event. Third, we have specified the table name on which the trigger is associated. Finally, we have written the trigger logic inside the trigger body that insert the deleted row into the salary\_archives table.

### How to call the BEFORE DELETE trigger?

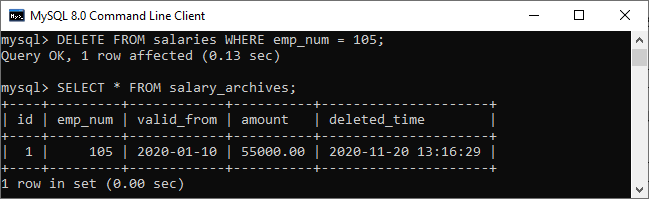
Let us test the above created BEFORE DELETE trigger and how we can call them. So first, we will remove a row from the salaries table:

1. mysql> **DELETE** **FROM** salaries **WHERE** emp\_num = 105;

Second, we will query data from the salary\_archives table to verify the above-created trigger is invoked or not by using the select statement:

1. mysql> **SELECT** \* **FROM** salary\_archives;

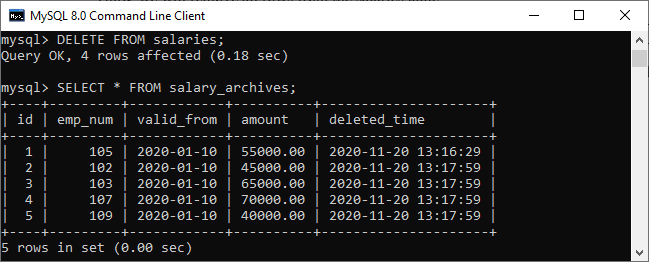
After executing a statement, we can see that the trigger was invoked successfully and inserted a new record into the salary\_archives table.



Third, we will remove all rows from the salaries table:

1. mysql> **DELETE** **FROM** salaries;

Finally, we will query data from the salary\_archives table again. The trigger was called four times because the DELETE statement removed four records from the salaries table. See the below output:



# MySQL AFTER DELETE Trigger

The AFTER DELETE Trigger in MySQL is invoked automatically whenever a delete event is fired on the table. In this article, we are going to learn how to create an AFTER DELETE trigger with its syntax and example.

### Syntax

The following is the syntax to create an **AFTER DELETE** trigger in MySQL:

1. **CREATE** **TRIGGER** trigger\_name
2. **AFTER** **DELETE**
3. **ON** table\_name **FOR** EACH ROW
4. Trigger\_body ;

The AFTER DELETE trigger syntax parameter can be explained as below:

* First, we will specify the **name of the trigger** that we want to create. It should be unique within the schema.
* Second, we will specify the **trigger action time**, which should be AFTER DELETE. This trigger will be invoked after each row of alterations occurs on the table.
* Third, we will specify the **name of a table** to which the trigger is associated. It must be written after the ON keyword. If we did not specify the table name, a trigger would not exist.
* Finally, we will specify the **trigger body** that contains a statement for execution when the trigger is activated.

If we want to execute multiple statements, we will use the **BEGIN END** block that contains a set of SQL queries to define the logic for the trigger. See the below syntax:

1. DELIMITER $$
2. **CREATE** **TRIGGER** trigger\_name **AFTER** **DELETE**
3. **ON** table\_name **FOR** EACH ROW
4. **BEGIN**
5. variable declarations
6. **trigger** code
7. **END**$$
8. DELIMITER ;

### Restrictions

* We can access the OLD rows but cannot update them in the AFTER DELETE trigger.
* We cannot access the NEW rows. It is because there are no NEW row exists.
* We cannot create an AFTER DELETE trigger on a VIEW.

### AFTER DELETE Trigger Example

Let us understand how to create an AFTER DELETE trigger using the [CREATE TRIGGER statement in MySQL](https://www.javatpoint.com/mysql-create-trigger) with an example.

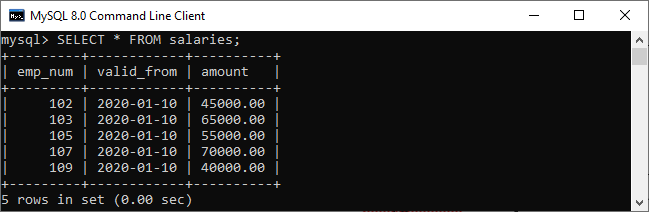
Suppose we have created a table named **salaries** to store the salary information of an employee as follows:

1. **CREATE** **TABLE** salaries (
2. emp\_num **INT** **PRIMARY** **KEY**,
3. valid\_from **DATE** NOT NULL,
4. amount **DEC**(8 , 2 ) NOT NULL **DEFAULT** 0
5. );

Next, we will insert some records into this table using the below statement:

1. **INSERT** **INTO** salaries (emp\_num, valid\_from, amount)
2. **VALUES**
3. (102, '2020-01-10', 45000),
4. (103, '2020-01-10', 65000),
5. (105, '2020-01-10', 55000),
6. (107, '2020-01-10', 70000),
7. (109, '2020-01-10', 40000);

Execute the SELECT query to see the table data.



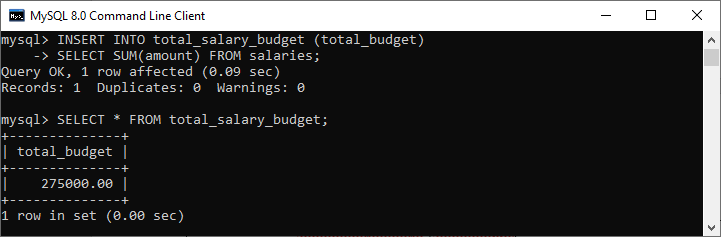
Third, we will create another table named **total\_salary\_budget** that keeps the salary information from the salaries table.

1. **CREATE** **TABLE** total\_salary\_budget(
2. total\_budget **DECIMAL**(10,2) NOT NULL
3. );

Fourth, we will use the **SUM()** function that returns the total salary from the salaries table and keep this information in the total\_salary\_budget table:

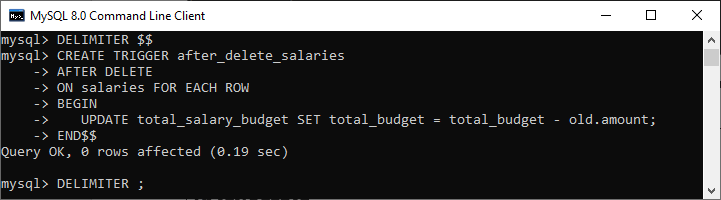
1. mysql> **INSERT** **INTO** total\_salary\_budget (total\_budget)
2. **SELECT** SUM(amount) **FROM** salaries;

Execute the SELECT statement to verify the table:



We will then create an AFTER DELETE trigger that updates the total salary into the total\_salary\_budget table after a row is deleted from the salaries table.

1. DELIMITER $$
3. **CREATE** **TRIGGER** after\_delete\_salaries
4. **AFTER** **DELETE**
5. **ON** salaries **FOR** EACH ROW
6. **BEGIN**
7. **UPDATE** total\_salary\_budget **SET** total\_budget = total\_budget - old.amount;
8. **END**$$
10. DELIMITER ;



In this trigger, we have first specified the trigger name after\_delete\_salaries. Then, specify the triggering event. Third, we have specified the table name on which the trigger is associated. Finally, we have written the trigger logic inside the trigger body that updates the total salary into the total\_salary\_budget table after a row is deleted from the salaries table.

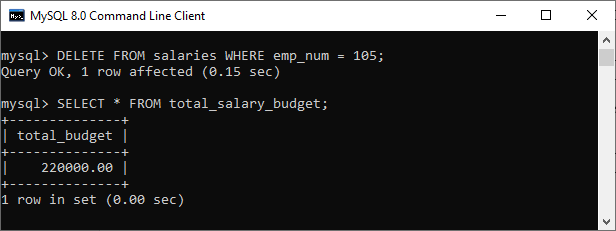
### How to call the AFTER DELETE trigger?

First, we will delete a salary from the salaries table using the following statements to invoke the above-created trigger:

1. mysql> **DELETE** **FROM** salaries **WHERE** emp\_num = 105;

Next, we will query data from the total\_salary\_budget table. We can see that table has been modified after the execution of the query. See the below output:

1. mysql> **SELECT** \* **FROM** total\_salary\_budget;

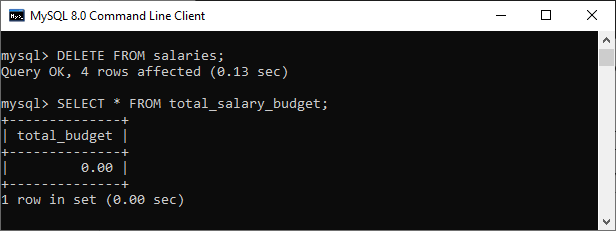


In the output, we can see that the deleted salary reduces the total\_budget.

Third, we will remove all data from the salaries table:

1. mysql> **DELETE** **FROM** salaries;

Again, we will query data from the total\_salary\_budget table. We can see that the trigger updated the table to zero after the execution of the query. See the below output:



# ACID Properties in DBMS

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)

A [**transaction**](https://www.geeksforgeeks.org/sql-transactions/) is a single logical unit of work that accesses and possibly modifies the contents of a database. Transactions access data using read and write operations.   
In order to maintain consistency in a database, before and after the transaction, certain properties are followed. These are called **ACID** properties.

### 

### ****Atomicity:****

By this, we mean that either the entire transaction takes place at once or doesn’t happen at all. There is no midway i.e. transactions do not occur partially. Each transaction is considered as one unit and either runs to completion or is not executed at all. It involves the following two operations.   
—**Abort**: If a transaction aborts, changes made to the database are not visible.   
—**Commit**: If a transaction commits, changes made are visible.   
Atomicity is also known as the ‘All or nothing rule’.

Consider the following transaction **T** consisting of **T1** and **T2**: Transfer of 100 from account **X** to account **Y**.



If the transaction fails after completion of **T1** but before completion of **T2**.( say, after **write(X)** but before **write(Y)**), then the amount has been deducted from **X** but not added to **Y**. This results in an inconsistent database state. Therefore, the transaction must be executed in its entirety in order to ensure the correctness of the database state.

### Consistency:

This means that integrity constraints must be maintained so that the database is consistent before and after the transaction. It refers to the correctness of a database. Referring to the example above,   
The total amount before and after the transaction must be maintained.   
Total **before T** occurs = **500 + 200 = 700**.   
Total **after T occurs** = **400 + 300 = 700**.   
Therefore, the database is **consistent**. Inconsistency occurs in case **T1** completes but **T2** fails. As a result, T is incomplete.

### Isolation:

This property ensures that multiple transactions can occur concurrently without leading to the inconsistency of the database state. Transactions occur independently without interference. Changes occurring in a particular transaction will not be visible to any other transaction until that particular change in that transaction is written to memory or has been committed. This property ensures that the execution of transactions concurrently will result in a state that is equivalent to a state achieved these were executed serially in some order.   
Let **X**= 500, **Y** = 500.   
Consider two transactions **T** and **T”.**



Suppose **T** has been executed till **Read (Y)** and then **T’’** starts. As a result, interleaving of operations takes place due to which **T’’** reads the correct value of **X** but the incorrect value of **Y** and sum computed by   
**T’’: (X+Y = 50, 000+500=50, 500)**   
is thus not consistent with the sum at end of the transaction:   
**T: (X+Y = 50, 000 + 450 = 50, 450)**.   
This results in database inconsistency, due to a loss of 50 units. Hence, transactions must take place in isolation and changes should be visible only after they have been made to the main memory.

### Durability:

This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if a system failure occurs. These updates now become permanent and are stored in non-volatile memory. The effects of the transaction, thus, are never lost.

|  |  |
| --- | --- |
| **Triggers** are defined as stored programs which are automatically executed whenever some events such as CREATE, ALTER, UPDATE, INSERT, DELETE takes place.They can be defined on a database, table, view with which event is associated.  Triggers can be broadly classified into **Row Level** and **Statement Level** triggers. Broadly, these can be differentiated as: |  |
|  |  |

**Cursor** is a Temporary Memory or Temporary Work Station. It is Allocated by Database Server at the Time of Performing DML(Data Manipulation Language) operations on Table by User. Cursors are used to store Database Tables. There are 2 types of Cursors: Implicit Cursors, and Explicit Cursors. These are explained as following below.

1. **Implicit Cursors:**  
   Implicit Cursors are also known as Default Cursors of SQL SERVER. These Cursors are allocated by SQL SERVER when the user performs DML operations.
2. **Explicit Cursors :**  
   Explicit Cursors are Created by Users whenever the user requires them. Explicit Cursors are used for Fetching data from Table in Row-By-Row Manner.

**How to create Explicit Cursor:**

1. **Declare Cursor Object.**  
   **Syntax :** DECLARE cursor\_name CURSOR FOR SELECT \* FROM table\_name

DECLARE s1 CURSOR FOR SELECT \* FROM studDetails

1. **Open Cursor Connection.**  
   **Syntax :** OPEN cursor\_connection