Learning Outcome

After completing this module, a student will be able to:

**Basic Concepts of MySQL**

* Understanding MySQL database
* MySQL syntax and semantics
* MySQL DDL
* MySQL DML
* MySQL Transaction Control
* MySQL User Management
* MySQL Security aspects
* MySQL Query execution
* MySQL query optimization
* MySQL Complex queries

**Understanding MySQL database**

**What is a Database?**

A database is a separate application that stores a collection of data. Each database has one or more distinct APIs for creating, accessing, managing, searching and replicating the data it holds.

Other kinds of data stores can also be used, such as files on the file system or large hash tables in memory but data fetching and writing would not be so fast and easy with those type of systems.

Nowadays, we use relational database management systems (RDBMS) to store and manage huge volume of data. This is called relational database because all the data is stored into different tables and relations are established using primary keys or other keys known as Foreign Keys.

**A Relational DataBase Management System (RDBMS) is a software that –**

* Enables you to implement a database with tables, columns and indexes.
* Guarantees the Referential Integrity between rows of various tables.
* Updates the indexes automatically.
* Interprets an SQL query and combines information from various tables.

**RDBMS Terminology**

Before we proceed to explain the MySQL database system, let us revise a few definitions related to the database.

* **Database −** A database is a collection of tables, with related data.
* **Table −** A table is a matrix with data. A table in a database looks like a simple spreadsheet.
* **Column −** One column (data element) contains data of one and the same kind, for example the column postcode.
* **Row −** A row (= tuple, entry or record) is a group of related data, for example the data of one subscription.
* **Redundancy −** Storing data twice, redundantly to make the system faster.
* **Primary Key −** A primary key is unique. A key value can not occur twice in one table. With a key, you can only find one row.
* **Foreign Key −** A foreign key is the linking pin between two tables.
* **Compound Key −** A compound key (composite key) is a key that consists of multiple columns, because one column is not sufficiently unique.
* **Index −** An index in a database resembles an index at the back of a book.
* **Referential Integrity −** Referential Integrity makes sure that a foreign key value always points to an existing row.

**MySQL Database**

MySQL is a fast, easy-to-use RDBMS being used for many small and big businesses. MySQL is developed, marketed and supported by MySQL AB, which is a Swedish company. MySQL is becoming so popular because of many good reasons –

* MySQL is released under an open-source license. So you have nothing to pay to use it.
* MySQL is a very powerful program in its own right. It handles a large subset of the functionality of the most expensive and powerful database packages.
* MySQL uses a standard form of the well-known SQL data language.
* MySQL works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA, etc.
* MySQL works very quickly and works well even with large data sets.
* MySQL is very friendly to PHP, the most appreciated language for web development.
* MySQL supports large databases, up to 50 million rows or more in a table. The default file size limit for a table is 4GB, but you can increase this (if your operating system can handle it) to a theoretical limit of 8 million terabytes (TB).
* MySQL is customizable. The open-source GPL license allows programmers to modify the MySQL software to fit their own specific environments.

**MySQL syntax and Semitics**

**Key and Relational Data Manipulation   
What are Keys?**

A DBMS key is an attribute or set of an attribute which helps you to identify a row(tuple) in a relation(table). They allow you to find the relation between two tables. Keys help you uniquely identify a row in a table by a combination of one or more columns in that table.

**Example:**

Student ID FirstName LastName

1 Qanith Khan

2 john Hale

3 Rajesh Singh

In the above-given example, Student ID is a primary key because it uniquely identifies an Student record. In this table, no other Student can have the same Student ID.

**Why do we need a Key?**

Here are reasons for using Keys in the DBMS system.

* Keys help you to identify any row of data in a table. In a real-world application, a table could contain thousands of records. Moreover, the records could be duplicated. Keys ensure that you can uniquely identify a table record despite these challenges.
* Allows you to establish a relationship between and identify the relation between tables
* Help you to enforce identity and integrity in the relationship.

**Various Keys**

Each keys have their different functionality:

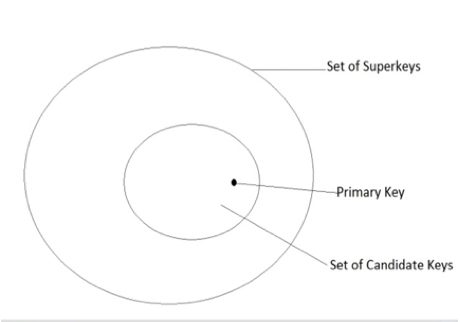
* Super Key
* Primary Key
* Candidate Key
* Alternate Key
* Foreign Key
* Compound Key
* Composite Key
* Surrogate Key

**Super key:**

A superkey is a group of single or multiple keys which identifies rows in a table. A Super key may have additional attributes that are not needed for unique identification.

**Example:**

|  |  |  |
| --- | --- | --- |
| **EmpSSN** | **EmpNum** | **Empname** |
| 9812345098 | AB05 | Shown |
| 9876512345 | AB06 | Roslyn |
| 199937890 | AB07 | James |



https://www.slideshare.net/TechtudNetwork/relation-between-super-key-candidate-key-and-primary-key?qid=c7bb9eac-8135-480c-92ee-890bf4c 744aa&v=&b=&from\_search=1

**Primary Key**

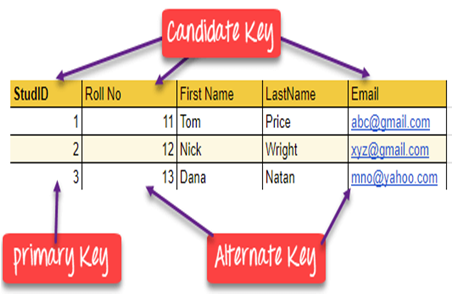
**PRIMARY KEY** is a column or group of columns in a table that uniquely identify every row in that table. The Primary Key can't be a duplicate meaning the same value can't appear more than once in the table. A table cannot have more than one primary key.

**Rules for defining Primary key:**

* Two rows can't have the same primary key value
* It must for every row to have a primary key value.
* The primary key field cannot be null.
* The value in a primary key column can never be modified or updated if any foreign key refers to that primary key.

**Example:**

In the following example, <code>StudID</code> is a Primary Key.



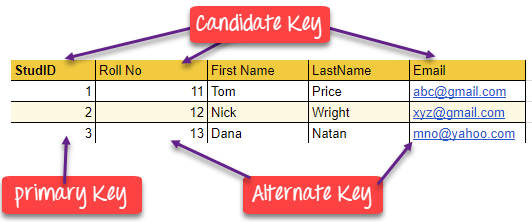
Reference- https://www.guru99.com/dbms-keys.html

**Alternate key**

**ALTERNATE KEYS** is a column or group of columns in a table that uniquely identify every row in that table. A table can have multiple choices for a primary key but only one can be set as the primary key. All the keys which are not primary key are called an Alternate Key.

**Example:**

In this table, StudID, Roll No, Email are qualified to become a primary key. But since StudID is the primary key, Roll No, Email becomes the alternative key.



**Alternate Key** Reference-https://www.guru99.com/dbms-keys.html

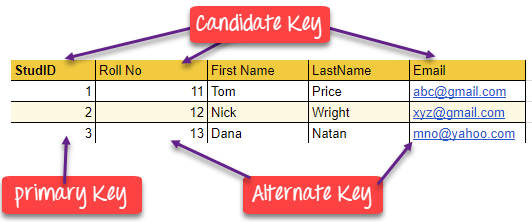
**Candidate Key**

**CANDIDATE KEY** is a set of attributes that uniquely identify tuples in a table. Candidate Key is a super key with no repeated attributes. The Primary key should be selected from the candidate keys. Every table must have at least a single candidate key. A table can have multiple candidate keys but only a single primary key.

**Properties of Candidate key:**

* It must contain unique values
* Candidate key may have multiple attributes
* Must not contain null values
* It should contain minimum fields to ensure uniqueness
* Uniquely identify each record in a table

**Example:** In the given table Stud ID, Roll No, and email are candidate keys which help us to uniquely identify the student record in the table.



Foreign key

Image 14- Candidate Key Reference-https://www.guru99.com/dbms-keys.html

**FOREIGN KEY** is a column that creates a relationship between two tables. The purpose of Foreign keys is to maintain data integrity and allow navigation between two different instances of an entity. It acts as a cross-reference between two tables as it references the primary key of another table.

**Example:**

|  |  |
| --- | --- |
| **DeptCode** | **DeptName** |
| 001 | Science |
| 002 | English |
| 005 | Computer |

|  |  |  |
| --- | --- | --- |
| **Teacher ID** | **Fname** | **Lname** |
| B002 | David | Warner |
| B017 | Sara | Joseph |
| B009 | Mike | Brunton |

In this example, we have two table, teach and department in a school. However, there is no way to see which search work in which department.

In this table, adding the foreign key in Deptcode to the Teacher name, we can create a relationship between the two tables.

|  |  |  |  |
| --- | --- | --- | --- |
| **Teacher ID** | **DeptCode** | **Fname** | **Lname** |
| B002 | 002 | David | Warner |
| B017 | 002 | Sara | Joseph |
| B009 | 001 | Mike | Brunton |

This concept is also known as Referential Integrity.

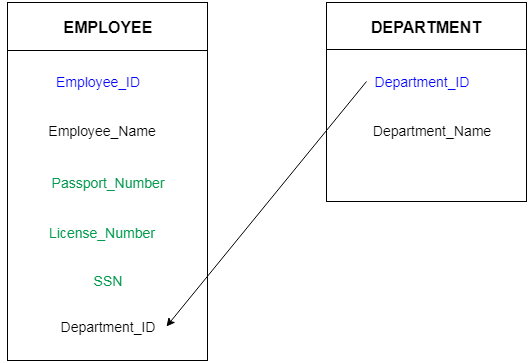


Image 15- Foreign key  
Reference- https://www.javatpoint.com/dbms-keys

**Compound key**

**COMPOUND KEY** has two or more attributes that allow you to uniquely recognize a specific record. It is possible that each column may not be unique by itself within the database. However, when combined with the other column or columns the combination of composite keys become unique. The purpose of compound key is to uniquely identify each record in the table.

**Example:**

|  |  |  |  |
| --- | --- | --- | --- |
| **OrderNo** | **ProductID** | **ProductName** | **Quantity** |
| B005 | JAP102459 | Mouse | 5 |
| B005 | DKT321573 | USB | 10 |
| B005 | OMG446789 | LCD Monitor | 20 |
| B004 | DKT321573 | USB | 15 |
| B002 | OMG446789 | Laser Printer | 3 |

In this example, OrderNo and ProductID can't be a primary key as it does not uniquely identify a record. However, a compound key of Order ID and Product ID could be used as it uniquely identified each record.

**Composite key**

**COMPOSITE KEY** is a combination of two or more columns that uniquely identify rows in a table. The combination of columns guarantees uniqueness, though individually uniqueness is not guaranteed. Hence, they are combined to uniquely identify records in a table.

The difference between compound and the composite key is that any part of the compound key can be a foreign key, but the composite key may or maybe not a part of the foreign key.

**Surrogate Key**

An artificial key which aims to uniquely identify each record is called a surrogate key. These kind of key are unique because they are created when you don't have any natural primary key. They do not lend any meaning to the data in the table. Surrogate key is usually an integer.

|  |  |  |  |
| --- | --- | --- | --- |
| **Fname** | **Lastname** | **Start Time** | **End Time** |
| Anne | Smith | 09:00 | 18:00 |
| Jack | Francis | 08:00 | 17:00 |
| Anna | McLean | 11:00 | 20:00 |
| Shown | Willam | 14:00 | 23:00 |

Above, given example, shown shift timings of the different employee. In this example, a surrogate key is needed to uniquely identify each employee.

Surrogate keys are allowed when

* No property has the parameter of the primary key.
* In the table when the primary key is too big or complicated.

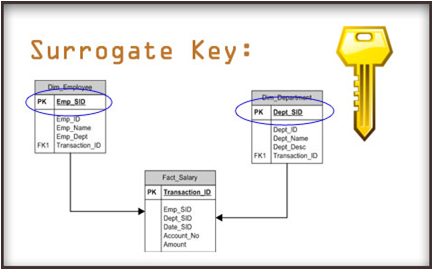


Image 16- Surrogate Key  
Reference- https://www.google.com/search?q=Surrogate+Key+&tbm=isch&ved=2ahUKEwiX99ycj-roAhUI5DgGHffQBnQQ2-cCegQIABAA&oq=Surroga te+Key+&gs\_lcp=CgNpbWcQAzIECAAQQzICCAAyAggAMgIIADICCAAyAggAMgIIADICCAAyAggAMgIIAFD-nBpY\_pwaYJuhGmgAcA B4AIABswGIAbMBkgEDMC4xmAEAoAEBqgELZ3dzLXdpei1pbWc&sclient=img&ei=89OWXtexAojI4-EP96GboAc&bih=657&biw=1366# imgrc=JViJUIE4gOGs0M

**Data Definition language (DDL)**

**Introduction to DDL**

* DDL stands for **Data Definition Language.**
* It is a language used for defining and modifying the data and its structure.
* It is used to build and modify the structure of your tables and other objects in the database.

**DDL commands are as follows:**

1. CREATE
2. DROP
3. ALTER
4. RENAME
5. TRUNCATE

* These commands can be used to add, remove or modify tables within a database.
* DDL has pre-defined syntax for describing the data.

**1. CREATE COMMAND**

* **CREATE command** is used for creating objects in the database.
* It creates a new table.

**Syntax:**

CREATE TABLE <table\_name>

( column\_name1 datatype, column\_name2 datatype,

.  
.  
.  
column\_name\_n datatype

);

**Example:** Create command

CREATE TABLE employee

(  
empid INT,

ename CHAR,  
age INT,  
city CHAR(25), phone\_no VARCHAR(20)

);

**2. DROP COMMAND**

* **DROP command** allows to remove entire database objects from the database.
* It removes entire data structure from the database.
* It deletes a table, index or view.

**Syntax:**

DROP TABLE <table\_name>;  
OR  
DROP DATABASE <database\_name>;

**Example:** DROP Command  
DROP TABLE employee;  
OR  
DROP DATABASE employees;

If you want to remove individual records, then use DELETE command of the DML statement.

**3. ALTER COMMAND**

* An **ALTER command** allows to alter or modify the structure of the database.
* It modifies an existing database object.
* Using this command, you can add additional column, drop existing column and even change the data type of columns.

**Syntax:**

ALTER TABLE <table\_name>

ADD <column\_name datatype>;

OR

ALTER TABLE <table\_name>  
CHANGE <old\_column\_name> <new\_column\_name>;

OR

ALTER TABLE <table\_name>

DROP COLUMN <column\_name>;

**Example:** Alter Command

ALTER TABLE employee

ADD (address varchar2(50));

OR

ALTER TABLE employee

CHANGE (phone\_no) (contact\_no);

OR

ALTER TABLE employee

DROP COLUMN age;

To view the changed structure of table, use 'DESCRIBE' command.

**For example:**

DESCRIBE TABLE employee;

**4. RENAME COMMAND**

* **RENAME command** is used to rename an object.
* It renames a database table.

**Syntax:**

RENAME TABLE <old\_name> TO <new\_name>;

**Example:**

RENAME TABLE emp TO employee;

**5. TRUNCATE COMMAND**

* **TRUNCATE command** is used to delete all the rows from the table permanently.
* It removes all the records from a table, including all spaces allocated for the records.
* This command is same as DELETE command, but TRUNCATE command does not generate any rollback data.

**Syntax:**

TRUNCATE TABLE <table\_name>;

**Example:**

TRUNCATE TABLE employee;

**Data Manipulation Language (DML)**

**Introduction to DML**

* DML stands for Data Manipulation Language.
* It is a language used for selecting, inserting, deleting and updating data in a database.
* It is used to retrieve and manipulate data in a relational database.

**DDL commands are as follows:**

1. SELECT
2. INSERT
3. UPDATE
4. DELETE

DML performs read-only queries of data.

**1. SELECT COMMAND**

* SELECT command is used to retrieve data from the database.
* This command allows database users to retrieve the specific information they desire from an operational database.
* It returns a result set of records from one or more tables.

**SELECT Command has many optional clauses are as stated below:**

|  |  |
| --- | --- |
| **Clause** | **Description** |
| WHERE | It specifies which rows to retrieve. |
| GROUP BY | It is used to arrange the data into groups. |
| HAVING | It selects among the groups defined by the GROUP BY clause. |
| ORDER BY | It specifies an order in which to return the rows. |
| AS | It provides an alias which can be used to temporarily rename tables or columns. |

**Syntax:**

SELECT \* FROM <table\_name>;

**Example:** Select Command SELECT \* FROM employee;

OR  
SELECT \* FROM employee where salary >=10,000;

**2. INSERT COMMAND**

* INSERT command is used for inserting data into a table.
* Using this command, you can add one or more records to any single table in a database.
* It is also used to add records to an existing code.

**Syntax:**

INSERT INTO <table\_name> (`column\_name1` <datatype>, `column\_name2` <datatype>, . . . , `column\_name\_n` <database>) VALUES (`value1`, `value2`, . . . , `value n`);

**Example:**

INSERT INTO employee (`eid` int, `ename` varchar(20), `city` varchar(20)) VALUES (`1`, `ABC`, `PUNE`);

**3. UPDATE COMMAND**

* UPDATE command is used to modify the records present in existing table.
* This command updates existing data within a table.
* It changes the data of one or more records in a table.

**Syntax:**

UPDATE <table\_name>  
SET <column\_name = value> WHERE condition;

**Example:** Update Command UPDATE employee

SET salary=20000 WHERE ename='ABC';

**4. DELETE COMMAND**

* DELETE command is used to delete some or all records from the existing table.
* It deletes all the records from a table.

**Syntax:**

DELETE FROM <table\_name> WHERE <condition>;

**Example:** Delete Command DELETE FROM employee WHERE emp\_id = '001';

If we does not write the WHERE condition, then all rows will get deleted.

# MySQL Transaction Control (TCL)

A transaction in MySQL is a **sequential group of statements**, queries, or operations such as select, insert, update or delete to perform as a one single work unit that can be committed or rolled back. If the transaction makes multiple modifications into the database, two things happen:

* Either all modification is successful when the transaction is committed.
* Or, all modifications are undone when the transaction is rollback.

In other words, a transaction cannot be successful without completing each operation available in the set. It means if any statement fails, the transaction operation cannot produce results.

A transaction in MySQL starts with the first executable SQL statement and ends when it finds a commit or rolled back either explicitly or implicitly. It explicitly uses COMMIT or ROLLBACK statement and implicitly when a DDL statement is used.

**Let us understand the concept of a transaction through the following explanation.**

We can understand the concept of a transaction in [MySQL](https://www.javatpoint.com/mysql-tutorial) by considering a **banking database**. Suppose a bank customer wants to transfer money from one account to another account. We can achieve this by using the SQL statements that will be divided into the following steps:

* First, it is required to check the availability of the requested amount in the first account.
* Next, if the amount is available, deduct it from the first account. Then, update the first account.
* Finally, deposit the amount in the second account. Then update the second account to complete the transaction.
* If any of the above processes fails, the transaction will be rolled back into its previous state.

### **Properties of Transaction**

The transaction contains mainly four properties, which referred to as **ACID** property. Now, we are going to discuss the ACID property in detail. The ACID property stands for:

1. Atomicity
2. Consistency
3. Isolation
4. Durability

**Atomicity:** This property ensures that all statements or operations within the transaction unit must be executed successfully. Otherwise, if any operation is failed, the whole transaction will be aborted, and it goes rolled back into their previous state. It includes features:

* COMMIT statement.
* ROLLBACK statement.
* Auto-commit setting.
* Operational data from the INFORMATION\_SCHEMA tables.

**Consistency:** This property ensures that the database changes state only when a transaction will be committed successfully. It is also responsible for protecting data from crashes. It includes features:

* InnoDB doublewrite buffer.
* InnoDB crash recovery.

**Isolation:** This property guarantees that each operation in the transaction unit operated independently. It also ensures that statements are transparent to each other. It includes features:

* SET ISOLATION LEVEL statement.
* Auto-commit setting.
* The low-level details of InnoDB locking.

**Durability:** This property guarantees that the result of committed transactions persists permanently even if the system crashes or failed. It includes features:

* Write buffer in a storage device.
* Battery-backed cache in a storage device.
* Configuration option innodb\_file\_per\_table.
* Configuration option innodb\_flush\_log\_at\_trx\_commit.
* Configuration option sync\_binlog.

### **MySQL Transaction Statement**

MySQL control transactions with the help of the following statement:

* MySQL provides a START TRANSACTION statement to begin the transaction. It also offers a "BEGIN" and "BEGIN WORK" as an alias of the START TRANSACTION.
* We will use a COMMIT statement to commit the current transaction. It allows the database to make changes permanently.
* We will use a ROLLBACK statement to roll back the current transaction. It allows the database to cancel all changes and goes into their previous state.
* We will use a SET auto-commit statement to disable/enable the auto-commit mode for the current transaction. By default, the COMMIT statement executed automatically. So if we do not want to commit changes automatically, use the below statement:

**SET** autocommit = 0;

OR,

**SET** autocommit = **OFF**:

Again, use the below statement to enable auto-commit mode:

**SET** autocommit = 1;

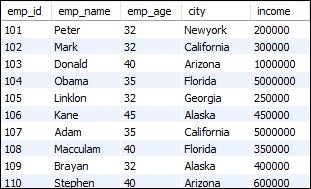
OR,

**SET** autocommit = **ON**:

### **MySQL Transaction Example**

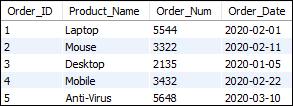
Suppose we have two tables named **"employees"** and **"Orders"** that contains the following data:

**Table: employees**



https://static.javatpoint.com/mysql/images/mysql-transaction.png

**Table: orders**



https://static.javatpoint.com/mysql/images/mysql-transaction2.png

### **COMMIT Example**

If we want to use a transaction, it is required to break the [SQL](https://www.javatpoint.com/sql-tutorial) statements into logical portions. After that, we can define whether the data should be committed or rollback.

The following steps illustrate to create a transaction:

1. Begin the transaction using the START TRANSACTION statement.
2. Then, select maximum income among the employee.
3. Add a new record to the employee table.
4. Add a new record into the order table.
5. Use the COMMIT statement to complete the transaction.

Below are the commands that perform the above operations:

-- 1. Start a new transaction

START **TRANSACTION**;

-- 2. Get the highest income

**SELECT** @income:= **MAX**(income) **FROM** employees;

-- 3. Insert a new record into the employee table

**INSERT** **INTO** employees(emp\_id, emp\_name, emp\_age, city, income)

**VALUES** (111, 'Alexander', 45, 'California', 70000);

-- 4. Insert a new record into the order table

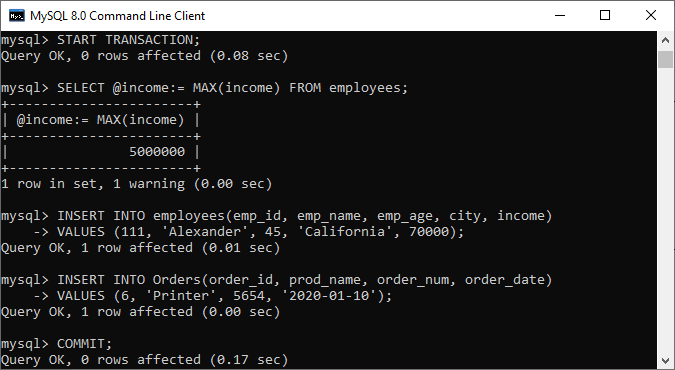
**INSERT** **INTO** Orders(order\_id, prod\_name, order\_num, order\_date)

**VALUES** (6, 'Printer', 5654, '2020-01-10');

-- 5. Commit changes

**COMMIT**;

The below image explains it more clearly:



### https://static.javatpoint.com/mysql/images/mysql-transaction3.png

### **ROLLBACK Example**

We can understand the rollback transaction with the help of the following illustration. First, open the MySQL command prompt and log into the database server using the password. Next, we have to select a database.

Suppose our database contains the "**Orders**" table. Now, the following are the scripts that perform the rollback operations:

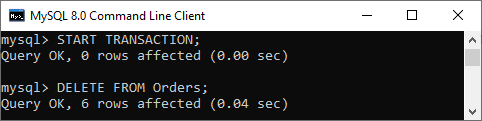
-- 1. Start a new transaction

START **TRANSACTION**;

-- 2. Delete data from the order table

**DELETE** **FROM** Orders;

After the execution of the above statement, we will get the output as below that shows all the records from the table Orders were successfully deleted.

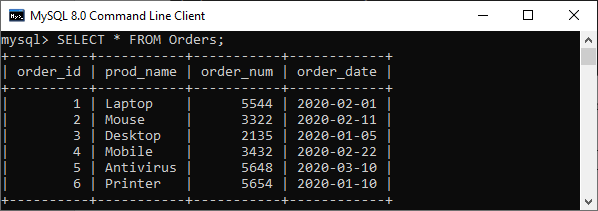


https://static.javatpoint.com/mysql/images/mysql-transaction4.png

Now, we need to open a separate session of MySQL database server and execute the below statement to verify the data in Orders table:

**SELECT** \* **FROM** Orders;

It will give the output as below.



https://static.javatpoint.com/mysql/images/mysql-transaction5.png

Although we have made changes in the first session, we still can see the records are available in the table. It is because the changes are not permanent until we have not executed the **COMMIT or ROLLBACK** statement in the first session.

Therefore if we want to make changes permanent, use the COMMIT statement. Otherwise, execute the ROLLBACK statement to roll back the changes in the first session.

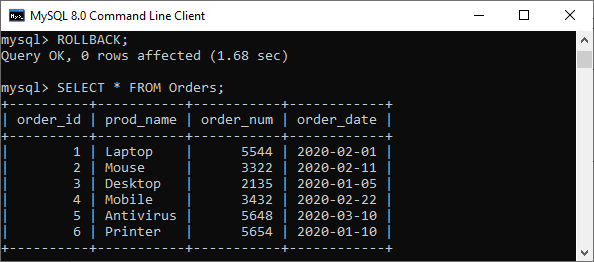
-- 3. Rollback changes

**ROLLBACK**;

-- 4. Verify the records in the first session

**SELECT** \* **FROM** Orders;

After the successful execution, it will produce the following result where we can see that the change has been rolled back.



**https://static.javatpoint.com/mysql/images/mysql-transaction6.png**

**Statements that cannot be a rollback in using MySQL Transaction.**

MySQL Transaction cannot be able to roll back all statements. For example, these statements include DDL (Data Definition Language) commands such as CREATE, ALTER, or DROP database as well as CREATE, UPDATE, or DROP tables or stored routines. We have to make sure that when we design our transaction, these statements do not include.

### **SAVEPOINT, ROLLBACK TO SAVEPOINT, RELEASE SAVEPOINT**

The **SAVEPOINT** statement creates a special mark with the name of the **identifier** inside a transaction. It allows all statements that are executed after savepoint would be rolled back. So that the transaction restores to the previous state it was in at the point of the savepoint. If we have set multiple savepoints in the current transaction with the same name, the newly savepoint is responsible for rollback.

The **ROLLBACK TO SAVEPOINT** statement allows us to rolls back all transactions to the given savepoint was established without aborting the transaction.

The **RELEASE SAVEPOINT** statement destroys the named savepoint from the current transaction without undoing the effects of queries executed after the savepoint was established. After these statements, no rollback command occurs. If the savepoint does not exist in the transaction, it gives an error.

The following are the **syntax** of the above statements in MySQL Transaction:

SAVEPOINT savepoint\_name

**ROLLBACK** **TO** [SAVEPOINT] savepoint\_name

RELEASE SAVEPOINT savepoint\_name

### **Example**

Let us understand how to use these statements through the example. In the below example, we are going to use SAVEPOINT and ROLLBACK TO SAVEPOINT statements that explain how a savepoint determines which records of the current transaction can be rolled back.

START **TRANSACTION**;

**SELECT** \* **FROM** Orders;

**INSERT** **INTO** Orders(order\_id, prod\_name, order\_num, order\_date)

**VALUES** (6, 'Printer', 5654, '2020-01-10');

SAVEPOINT my\_savepoint;

**INSERT** **INTO** Orders(order\_id, prod\_name, order\_num, order\_date)

**VALUES** (7, 'Ink', 5894, '2020-03-10');

**ROLLBACK** **TO** SAVEPOINT my\_savepoint;

**INSERT** **INTO** Orders(order\_id, prod\_name, order\_num, order\_date)

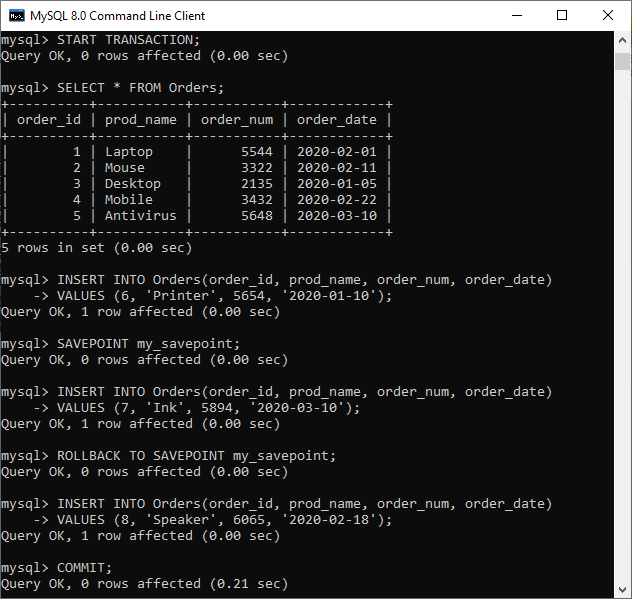
**VALUES** (8, 'Speaker', 6065, '2020-02-18');

**COMMIT**;

In the above,

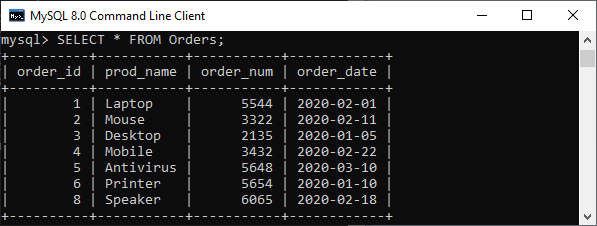
* We have to first begin the transaction and then show the records available in the Orders table.
* Next, we have inserted one record into the table and then creates a savepoint mark.
* Again, we have inserted one record into the table and then use a ROLLBACK TO SAVEPOINT statement to remove changes where the savepoint established.
* Again, we have inserted one record into the table.
* Finally, execute the COMMIT statement to make changes permanently.

The output below explains the above steps in a sequential order that helps to understand it very easily.



https://static.javatpoint.com/mysql/images/mysql-transaction7.png

Now, we will use a SELECT statement to verify the above operation. In the output, we can see that the **order\_id=6** and **order\_id=8** is added successfully, but **order\_id=7** is not inserted into the table. It rolls back the values entered after the savepoint was established:



https://static.javatpoint.com/mysql/images/mysql-transaction8.png

Now we are going to take another example RELEASE SAVEPOINT that establishes the my\_savepoint and then removes a savepoint.

START **TRANSACTION**;

**INSERT** **INTO** Orders(order\_id, prod\_name, order\_num, order\_date)

**VALUES** (7, 'Ink', 5894, '2020-03-10');

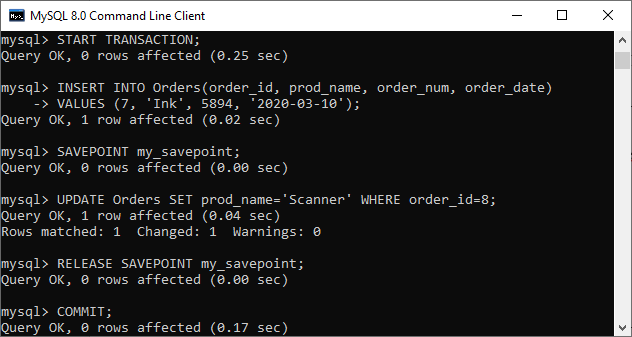
SAVEPOINT my\_savepoint;

**UPDATE** Orders **SET** prod\_name='Scanner' **WHERE** order\_id=8;

RELEASE SAVEPOINT my\_savepoint;

**COMMIT**;

In the output, we can see that all statements in the transaction executed successfully. Here, both [INSERT](https://www.javatpoint.com/mysql-insert) and [UPDATE](https://www.javatpoint.com/mysql-update) statements modify the table at COMMIT.



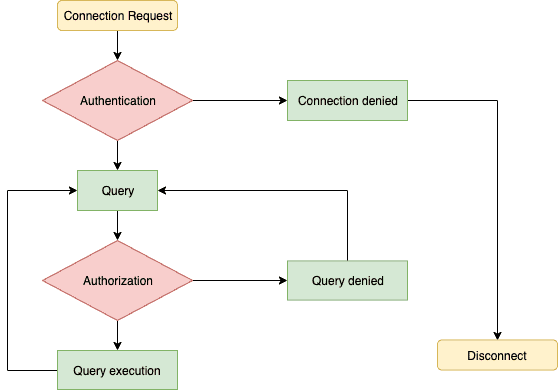
https://static.javatpoint.com/mysql/images/mysql-transaction9.png

**MySQL User Management**

Managing users in MySQL gives you the ability to control what users can and cannot do.

* Create user accounts with different privileges that are appropriate to their function.
* Avoid using the root account – Constrain compromised applications and protect against mistakes during routine maintenance.
* Ensure data integrity by proper assignment of individual user privileges. Permit authorized users to do their work. Prevent unauthorized users from accessing data beyond their privileges.

**User Account Verification**



https://cdn.thegeekdiary.com/wp-content/uploads/2019/10/Untitled-Diagram.png

When you connect to a MySQL server and execute a query, it authenticates you and authorizes your activity.

* **Authentication**: Verifies the user’s identity. This is the first stage of access control. You must successfully authenticate each time you connect. If you fail to authenticate, your connection fails and your client disconnects.
* **Authorization**: Verifies the user’s privileges. This second stage of access control takes place for each request on an active connection on which authentication has succeeded. For every request, MySQL determines what operation you want to perform, and then checks whether you have sufficient privileges to do so.

## View User Account Setting

## – Query the mysql database to view user identification info:

mysql> SELECT user, host, password

-> FROM mysql.user WHERE user='root';

+------+-----------+-------------------------------------------+

| user | host | password |

+------+-----------+-------------------------------------------+

| root | localhost | \*2447D497B9A6A15F2776055CB2D1E9F86758182F |

| root | 127.0.0.1 | \*2447D497B9A6A15F2776055CB2D1E9F86758182F |

| root | ::1 | \*2447D497B9A6A15F2776055CB2D1E9F86758182F |

+------+-----------+-------------------------------------------+

3 rows in set (0.00 sec)

View all user info, including privileges:

mysql> SELECT \* FROM mysql.user\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

...

Select\_priv: Y

Insert\_priv: Y

Update\_priv: Y

Delete\_priv: Y

...

The mysql database contains the information for all user accounts on the server. To view this information, run the SELECT statements shown above. The value **Y** in a **\*\_priv** field indicates that the privilege is enabled. The root account has full access. All of its privilege columns have the value Y.

As well as privileges, the user table contains other information that is useful in the authentication process. For example, you can see in the following output that the tester user:

* Has a password (visible in encrypted form in the Password column), and this password is not expired (indicated by the N in the password\_expired column)
* Has no defined resource limits (indicated by the 0s in the max\_\* columns)
* Does not have any SSL or x509 settings (indicated by the blank values in the ssl\_\* and x509\_\* columns)
* Uses the mysql\_native\_password plugin to authenticate (The plugin name is listed in the plugin column.)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Host: localhost

User: testuser

Password: \*14E65567ABDB5135D0CFD9A70B3032C179A49EE7

Select\_priv: Y

Insert\_priv: N

...

Trigger\_priv: N

Create\_tablespace\_priv: N

ssl\_type:

ssl\_cipher:

x509\_issuer:

x509\_subject:

max\_questions: 0

max\_updates: 0

max\_connections: 0

max\_user\_connections: 0

plugin: mysql\_native\_password

authentication\_string:

password\_expired: N

## Native Authentication

When you connect to a MySQL server using the native password authentication plugin (the default authentication mechanism), it matches the username that you specified, the host from which you are connecting, and your password against rows in the mysql.user table to determine whether you can connect and perform actions.

To connect to the local server using the mysql client, specify the username and password for the account that you want to use:

shell> mysql -u[username] -p[password]

To connect to a server that is not installed on your client’s local host, provide the host name of the server to which you are connecting:

shell> mysql -u[username] -p[password] -h[server\_host]

## Creating a User Account

An account name consists of a username and the name of the client host from which the user must connect to the server. Account names have the format ‘**user\_name’@’host\_name**‘. Usernames can be up to 16 characters long. You must use single quotation marks around usernames and host names if these contain special characters, such as dashes. If a value is valid as an unquoted identifier, the quotes are optional. However, you can always use quotes.

For example, use the CREATE USER…IDENTIFIED BY statement to build an account:  
– For a user named jim  
– To connect from localhost  
– Using the password Abc123

CREATE USER 'jim'@'localhost' IDENTIFIED BY 'Abc123';

Avoid possible security risks when creating accounts:

* Do not create accounts without a password.
* Do not create anonymous accounts.
* Where possible, avoid wildcards when you specify account host names.

**Host Name Patterns**

Use a host pattern containing the **%** or **\_** wildcard characters to set up an account that enables the user to connect from any host in an entire domain or subnet. If you omit the host part of an account name when writing an account management statement, MySQL assumes a host name of %.

A host value of %.example.com matches any host in the example.com domain. A host value of 192.168.% matches any host in the 192.168 subnet. A host value of % matches any host, permitting the user to connect from any host. Use an IP address with a subnet mask to enable the user to connect from any host with an address within that subnet. For example, a value of 10.0.0.0/255.255.255.0 matches any host with 10.0.0 in the first 24 bits of its IP address.

Avoid using wildcards in host names except where it is strictly necessary and properly audited to avoid abuse or accidental exposure. Run periodic checks as follows:

mysql> SELECT User, Host FROM mysql.user WHERE Host LIKE '%\%%';

Username and host name examples:

* john@10.20.30.40
* john@’10.20.30.%’
* john@’%.ourdomain.com’
* john@’10.20.30.0/255.255.255.0′

To specify an anonymous-user account (that is, an account that matches any username), specify an empty string for the username part of the account name:

mysql> CREATE USER ''@'localhost';

Avoid creating anonymous accounts, especially ones that have no password (as in the above example). This helps avoid security risks that would come from opening up access to the MySQL installation. If a host matches two or more patterns, MySQL chooses the most specific pattern.

## Setting the Account Password

The most common way to change an existing account’s password without changing any of its privileges is to use the **SET PASSWORD** statement. For example, to set the password for jim on the local host to NewPass, use the following statement:

mysql> SET PASSWORD FOR jim@localhost = PASSWORD('NewPass');

If you are logged in as a non-root user and your user does not have the UPDATE privilege for the mysql database, you can change only your own password. Do this by using the SET PASSWORD statement without the FOR clause:

mysql> SET PASSWORD = PASSWORD('NewPass');

When using SET PASSWORD, use the **PASSWORD()** function to encrypt the password. Note that the CREATE USER statement automatically encrypts the password that you provide, so you do not need to use the PASSWORD() function when creating a user with CREATE USER.

Use the following mysqladmin commands to set passwords from the shell:

shell> mysqladmin -u root password 'rootpass'

shell> mysqladmin -u root -h host\_name password 'rootpass'

In the preceding examples, ‘rootpass’ represents the root password and ‘host\_name’ is the name of the host from which the root account accesses the MySQL server.

## Confirming Passwords

Assign strong, unique passwords to all user accounts. Avoid passwords that can be easily guessed. Use the following SELECT statement to list any accounts  
without passwords:

SELECT Host, User FROM mysql.user WHERE Password = '';

To identify duplicate passwords:

SELECT User FROM mysql.user GROUP BY password HAVING count(user)>1;

To expire any users password, use below query:

ALTER USER jim@localhost PASSWORD EXPIRE;

To issue the SELECT statements in the examples above, you must connect with a user account with SELECT privileges on the mysql schema or mysql.user table. You can have multiple accounts that apply to a specific username. For example, if the user jim logs in from two locations and you set up accounts for each location, such as jim@localhost and jim@’192.168.14.38′, both accounts identified as jim might have the same password.

You can expire a user’s password with the ALTER USER…PASSWORD EXPIRE statement. If your password expires, you must change your password using a SET PASSWORD statement the next time you log in. All statements you execute that do not start with SET return an error until you change your password, as in this example:

mysql> SELECT \* FROM City WHERE 1=2;

ERROR 1820 (HY000): You must SET PASSWORD before executing this statement

mysql> SET PASSWORD = PASSWORD('new\_pwd');

Query OK, 0 rows affected (0.01 sec)

mysql> SELECT \* FROM City WHERE 1=2;

Empty set (0.00 sec)

## Manipulating User Accounts

Use the **RENAME USER** statement to rename a user account:

RENAME USER 'jim'@'localhost' TO 'james'@'localhost';

The above query changes the account name of an existing account or Changes either the username or hostname parts of the account name, or both.

Use the DROP USER statement to remove a user account:

DROP USER 'jim'@'localhost';

The above query revokes all privileges for an existing account and then removes the account. It also deletes all records for the account from any grant table in which they exist.

**MySQL Security aspects**

Database security entails allowing or disallowing user actions on the database and the objects within it. When you will create an database application, the security policy is the first step. An application security policy is a list of application security requirements and rules that regulate user access to database objects. This chapter discusses aspects of application security and MySQL Database features which contains the following topics :

**MySQL general security issues**

**Security Guidelines :**

* Except MySQL root account does not permit anyone to access the user table in the MySQL database.
* Use the GRANT and REVOKE statements to control access to MySQL. Do not grant unnecessary privileges and never grant privileges to all hosts.
* Never store simple text passwords in your database. Store the hash value using  SHA2(), [SHA1()](https://www.w3resource.com/mysql/encryption-and-compression-functions/sha1().php), [MD5()](https://www.w3resource.com/mysql/encryption-and-compression-functions/md5().php) functions or other hashing function in a different way. Try to use a complex password.
* Try to use a firewall and put MySQL behind the firewall.
* 3306 is the default user port of MySQL and this port should not be accessible from untrusted hosts. You can scan the ports from Internet using a tool such as nmap. From a remote machine you can check whether the port is open or not with this command: shell> telnet server\_host 3306. If telnet hangs or the connection is refused, the port is blocked. If you get a connection and some garbage characters, the port is open and should be closed on your firewall or router, unless you really have a good reason to keep it open.
* Some applications access MySQL database for different a purpose. Never trust these input data entered by the user and must validate properly before access database.
* Do not transmit unencrypted data over the Internet. Use an encrypted protocol such as SSL (MySQL supports internal SSL connections) or SSH.
* Use tcpdump and strings utilities. By issuing this command **shell> tcpdump -l -i eth0 -w - src or dst port 3306 | strings** you can check whether MySQL data streams are unencrypted or not.

**Keeping Passwords Secure:**

* End-User Guidelines for Password Security
  + Use the -p or --password option on the command line with no password value specified. Here is the command

shell> mysql -u user\_id -p database\_name  
Enter password : \*\*\*\*\*\*\*\*\*\*\*  
When you input the password it will not visible.

* + Store your password in an option file. For example Unix you can list your password in [client] section of the .my.cnf file in your home directory and to keep password safe, set the file access mode to 400 or 600.
* Administrator Guidelines for Password Security : MySQL stores passwords for user accounts in the mysql.user table. Therefore this table should not be accessed by any nonadministrative accounts. User account password must reset time to time.
* Passwords and Logging : Passwords can be written as plain text in SQL statements such as CREATE USER, GRANT, and SET PASSWORD, or statements that invoke the PASSWORD() function. If these statements are logged by the MySQL server as written, such passwords become available to anyone with access to the logs. Beginning with MySQL 5.6.3, statement logging is modified so that passwords do not appear in plain text for the following statements:
* CREATE USER ... IDENTIFIED BY ...GRANT ... IDENTIFIED BY ...SET PASSWORD ...SLAVE START ... PASSWORD = ... (as of 5.6.4)CREATE SERVER ... OPTIONS(... PASSWORD ...) (as of 5.6.9)ALTER SERVER ... OPTIONS(... PASSWORD ...) (as of 5.6.9)

Passwords in those statements are rewritten not to appear literally in statement text, for the general query log, slow query log, and binary log. Rewriting does not apply to other statements.

* Password Hashing in MySQL : MySQL lists user accounts in the user table of the MySQL database. Each MySQL account can be assigned a password, although the user table does not store the cleartext version of the password, but a hash value computed from it.
* Implications of Password Hashing Changes in MySQL 4.1 for Application Programs : An upgrade to MySQL version 4.1 or later can cause compatibility issues for applications that use PASSWORD() to generate passwords for their own purposes. Applications really should not do this, because [PASSWORD()](https://www.w3resource.com/mysql/encryption-and-compression-functions/password().php) should be used only to manage passwords for MySQL accounts.
* The validate\_password plugin (available as of MySQL 5.6.6) can be used to test passwords and improve security.

**Making MySQL Secure Against Attackers :**

To make a MySQL system secure, you should maintain the following suggestions :

* Require all MySQL accounts to have a password.
* Make sure that the only Unix user account with read or write privileges in the database directories is the account that is used for running mysqld.
* Never run the MySQL server as the Unix root user
* Do not grant the FILE privilege to nonadministrative users
* Do not permit the use of symlinks to tables.
* [Stored programs](https://www.w3resource.com/mysql/mysql-procedure.php) and [views](https://www.w3resource.com/mysql/mysql-views.php) should be written using the security guidelines
* If you do not trust your DNS, you should use IP addresses rather than hostnames in the grant tables.
* If you want to restrict the number of connections permitted to a single account, you can do so by setting the max\_user\_connections variable in mysqld.

**Security-Related mysqld Options and Variables :**

The following table shows mysqld options and system variables that affect security.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | **Cmd-Line** | **Option file** | **System Var** | **Var Scope** | **Dynamic** |
| allow-suspicious-udfs | This option controls whether user-defined functions that have only an xxx symbol for the main function can be loaded. By default, | Yes | Yes |  |  |  |
| automatic\_sp\_privileges | When this variable has a value of 1 (the default), the server automatically grants the EXECUTE and ALTER ROUTINE privileges to the creator of a stored routine, if the user cannot already execute and alter or drop the routine. (The ALTER ROUTINE privilege is required to drop the routine.) The server also automatically drops those privileges from the creator when the routine is dropped. If automatic\_sp\_privileges is 0, the server does not automatically add or drop these privileges. |  |  | Yes | Global | Yes |
| chroot | Put the mysqld server in a closed environment during startup by using the chroot() system call. | Yes | Yes |  |  |  |
| des-key-file | Read the default DES keys from this file. These keys are used by the DES\_ENCRYPT() and DES\_DECRYPT() functions. | Yes | Yes |  |  |  |
| local\_infile | Whether LOCAL is supported for LOAD DATA INFILE statements. If this variable is disabled, clients cannot use LOCAL in LOAD DATA statements. |  |  | Yes | Global | Yes |
| old\_passwords | This variable determines the type of password hashing performed by the PASSWORD() function and statements such as CREATE USER and GRANT. |  |  | Yes | Both | Yes |
| safe-user-create | If this option is enabled, a user cannot create new MySQL users by using the GRANT statement unless the user has the INSERT privilege for the mysql.user table or any column in the table. I | Yes | Yes |  |  |  |
| secure-auth | This option causes the server to block connections by clients that attempt to use accounts that have passwords stored in the old (pre-4.1) format. Use it to prevent all use of passwords employing the old format (and hence insecure communication over the network). | Yes | Yes |  | Global | Yes |
| - Variable: secure\_auth | If this variable is enabled, the server blocks connections by clients that attempt to use accounts that have passwords stored in the old (pre-4.1) format. |  |  | Yes | Global | Yes |
| secure-file-priv | By default, this variable is empty. If set to the name of a directory, it limits the effect of the LOAD\_FILE() function and the LOAD DATA and SELECT ... INTO OUTFILE statements to work only with files in that directory. | Yes | Yes |  | Global | No |
| - Variable: secure\_file\_priv |  |  |  | Yes | Global | No |
| skip-grant-tables | This option causes the server to start without using the privilege system at all, which gives anyone with access to the server unrestricted access to all databases. | Yes | Yes |  |  |  |
| skip-name-resolve | All interaction with mysqld must be made using named pipes or shared memory (on Windows) or Unix socket files (on Unix). This option is highly recommended for systems where only local clients are permitted. | Yes | Yes |  | Global | No |
| - Variable: skip\_name\_resolve |  |  |  | Yes | Global | No |
| skip-networking | All interaction with mysqld must be made using named pipes or shared memory (on Windows) or Unix socket files (on Unix). This option is highly recommended for systems where only local clients are permitted. | Yes | Yes |  | Global | No |
| - Variable: skip\_networking |  |  |  | Yes | Global | No |
| skip-show-database | This option sets the skip\_show\_database system variable that controls who is permitted to use the SHOW DATABASES statement. | Yes | Yes |  | Global | No |
| - Variable: skip\_show\_database |  |  |  | Yes | Global | No |

**How to Run MySQL as a Normal User:**

* On Windows, you can run the server as a Windows service using a normal user account.
* On Unix, the MySQL server mysqld can be started and run by any user. However, you should avoid running the server as the Unix root user for security reasons.

**Security Issues with LOAD DATA LOCAL:**

There are two potential security issues with supporting the LOCAL version of LOAD DATA statements :

* The transfer of the file from the client host to the server host is initiated by the MySQL server. In theory, a patched server could be built that would tell the client program to transfer a file of the server's choosing rather than the file named by the client in the LOAD DATA statement. Such a server could access any file on the client host to which the client user has read access.
* In a Web environment where the clients are connecting from a Web server, a user could use LOAD DATA LOCAL to read any files that the Web server process has read access to (assuming that a user could run any command against the SQL server). In this environment, the client with respect to the MySQL server actually is the Web server, not the remote program being run by the user who connects to the Web server.

**Client Programming Security Guidelines:**

Applications that access MySQL should not trust any data entered by users, who can try to trick your code by entering special or escaped character sequences in Web forms, URLs, or whatever application you have built. Be sure that your application remains secure if a user enters something like "; DROP DATABASE mysql;". This is an extreme example, but large security leaks and data loss might occur as a result of hackers using similar techniques if you do not prepare for them. See the following guidelines :

* Enable strict SQL mode to tell the server to be more restrictive of what data values it accepts.
* Try to enter single and double quotation marks (“'” and “"”) in all of your Web forms. If you get any kind of MySQL error, investigate the problem right away.
* Try to modify dynamic URLs by adding %22 (“"”), %23 (“#”), and %27 (“'”) to them.
* Try to modify data types in dynamic URLs from numeric to character types using the characters shown in the previous examples. Your application should be safe against these and similar attacks.
* Try to enter characters, spaces, and special symbols rather than numbers in numeric fields. Your application should remove them before passing them to MySQL or else generate an error. Passing unchecked values to MySQL is very dangerous.
* Check the size of data before passing it to MySQL.
* Do not give your applications any access privileges they do not need.

**MySQL Query Execution**

Here’s a brief overview of the order in which a query is executed inside a MySQL server.

When you execute a SQL query, the order in which the SQL directives get executed is:

* FROM clause
* WHERE clause
* GROUP BY clause
* HAVING clause
* SELECT clause
* ORDER BY clause

However, HAVING and GROUP BY clauses can come after SELECT depending on the order it is specified in the query.

How are queries executed at the back-end by the database engine? Let’s take an example for each of the clauses and understand the sequence.

|  |
| --- |
| **SELECT** \* **FROM** order\_details **WHERE** category = 'produce'; |

The first clause which gets executed is the FROM clause, which is used to list the tables and any joins required for the query. It is through this clause we can narrow down possible record set sizes. The above query is straight forward without any joins.

Let’s take another example with JOIN in the query

|  |
| --- |
| **SELECT** order\_details.order\_id, customers.customer\_name  **FROM** customers  **INNER** **JOIN** order\_details  **ON** customers.customer\_id = order\_details.customer\_id; |

In the above query, JOIN condition is evaluated in the first step. The order of JOIN operation is determined dynamically by the query optimizer when it constructs its query plan. The ON condition is the criteria for deciding which rows to join from each table. The result of the FROM clause is a temporary result (like a temporary table), consisting of combined rows which satisfy all the join conditions. In the above example, MySQL would return all rows from the customers and order\_details tables where there is a matching customer\_id value in both the customers and order\_details tables.

Next, comes the WHERE clause. If you don’t specify the WHERE clause in the statement the optimizer retrieves all the rows from the temporary result. In a query with a WHERE clause, each row in the temporary result is evaluated according to the WHERE conditions, and either discarded or retained.

Next, comes the GROUP BY clause, which is an optional part of the SELECT statement. If there’s a GROUP BY clause, the temporary result is now split into groups, one group for every combination of values in the columns in the GROUP BY clause. When you perform GROUP BY on table it will retrieve the first row in that group. The below GROUP BY example uses the COUNT function to return the product and the number of orders (for that product) that are in the produce category.

|  |
| --- |
| **SELECT** product, **COUNT**(\*) **AS** "Number of orders"  **FROM** order\_details  **WHERE** category = 'produce'  **GROUP** **BY** product; |

Now comes the HAVING clause. The HAVING clause enables you to specify conditions that filter which group results appear in the final results. It operates once on each group, and all rows from groups which do not satisfy the HAVING clause are eliminated. In the below query, after having assembled an entire temporary result table, the optimizer will filter the results so that only products with more than 20 orders will be returned. After the HAVING clause has filtered the groups, a new temporary result set is produced, and in this new temporary result, there is only one row per group.

|  |
| --- |
| **SELECT** product, **COUNT**(\*) **AS** "Number of orders"  **FROM** order\_details  **WHERE** category = 'produce'  **GROUP** **BY** product  **HAVING** **COUNT**(\*) > 20; |

The MySQL HAVING clause is used in combination with the GROUP BY clause to restrict the groups of returned rows to only those whose the condition is TRUE.

Next, comes the SELECT clause. From the rows of the new temporary result produced by the GROUP BY and HAVING clauses, the SELECT now assembles the columns it needs.

Finally, the last step is the ORDER BY clause. The ORDER BY clause is used to sort the records in the result set. In queries with both a GROUP BY and ORDER BY clause, you can reference columns in the ORDER BY only if they are in the new temporary result produced by the grouping process, i.e. columns in the GROUP BY or aggregate functions.

|  |
| --- |
| **SELECT** \* **FROM** (**SELECT** product, **COUNT**(\*) **AS** "Number of orders"  **FROM** order\_details  **WHERE** supplier\_name = 'Microsoft'  **GROUP** **BY** product)  **AS** temp\_table **ORDER** **BY** supplier\_city **DESC**; |

In the above example, GROUP BY will be executed first and then ORDER BY Clause. Using non-aggregate columns in a SELECT with a GROUP BY clause is non-standard. MySQL will generally return the values of the first row it finds and discards the rest. Any ORDER BY clauses will only apply to the returned column value, not to the discarded ones.

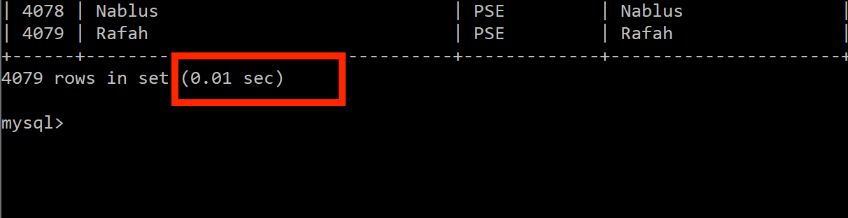
**MySQL Query Optimization**

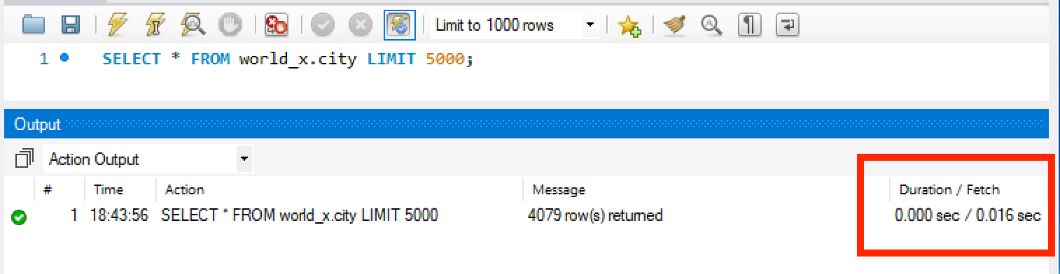
We optimize for speed and resources. Optimized queries can run faster and require less computing power.

### **Tips for MySQL Queries Optimization**

Fast queries are about response time. The goal is to have queries return the required result in the shortest time possible. How much time does a query take to execute? Most of the tools used to query a MySQL database give details on time taken to run a query.

The most straightforward query cost metrics used in MySQL are query response time, the number of rows scanned, and the number of rows returned. The more the number of rows read, the higher the cost of the query. The screenshots below show the time taken to run a query in both CLI and MySQL Workbench.

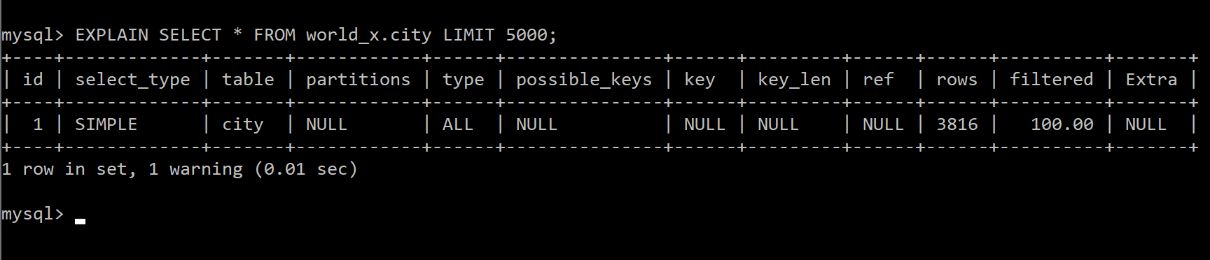




#### **Optimizing Queries with EXPLAIN**

The EXPLAIN statement provides information about how MySQL executes a statement. According to the [MySQL documentation](https://dev.mysql.com/doc/refman/8.0/en/using-explain.html), EXPLAIN works alongside SELECT, DELETE, INSERT, REPLACE, and UPDATE statements. It displays information from a built-in MySQL optimizer regarding the statement execution plan and the number of rows scanned in each table. Thus we can determine the cost of the query. The query below shows how EXPLAIN works with the SELECT statement.

EXPLAIN SELECT \* FROM world\_x.city LIMIT 5000;

 MySQL EXPLAIN query output

#### **MySQL Query Log**

In MySQL, slow queries are logged in an built-in query log. Once you find the slow queries in the query log, use the EXPLAIN statement to determine why the queries are slow and optimize them.

#### **Optimizing Database Schema**

The database structure is very crucial in performance optimization. There are several ways in which we can optimize database structure, including:

* Limiting the number of columns: MySQL has a limit of 4096 columns per table. Use fewer columns for better performance. If possible, do not use more than a hundred columns unless your business logic requires that. Tables with more columns require more CPU time to process.
* Normalize Tables: Normalizing keeps all data non-redundant. The database that is in this state is called 3NF (third normal form). The 3NF ensures that lengthy values such as names, addresses, categories, and contact details are not repeated. Instead, they are represented as IDs across multiple smaller tables. For more details on database normalization, refer [here](https://www.guru99.com/database-normalization.html).
* Use the Most Appropriate Data Types: There are more than 20 different data types in MySQL designed for different uses. Some of the data types include Timestamp, DateTime, Integer, ENUM, Float, Double, Char, LongText, and Text. Tables should be designed to minimize space used on a disk. Tables that occupy less disk space results in smaller indexes that can be processed in a shorter duration. For example, if a table will host less than 100 records, you should use the TINYINT data type for the unique ID as it takes less space than INT.
* Avoid Null Values. Declare columns to be NOT NULL where possible. This enables better use of indexes. NULL values increase the processing power needed for testing whether each value is NULL, making SQL operations slower.

#### **Use Indexes**

Think of records as content in a book. If you want to learn on a particular subtopic, you would go to the index pages, look for the subtopic you want, then get the page where the subtopic is. Indexes work the same way. They are used to find rows with specific column values much faster. Without using an index, MySQL must begin searching in the first row and go through the whole table to find the required records. Tables with a huge amount of data are more costly to query.

With the use of an index, MySQL can faster determine the position to seek in the middle of the data file. This is done without going through all the rows and is much faster than reading every row sequentially. Refer to the [MySQL developer](https://dev.mysql.com/doc/refman/8.0/en/mysql-indexes.html) guide for more information on indexes.

You can create a single-column or multiple column indexes, as shown below, respectively.

CREATE INDEX tablename\_columnname\_idx ON tablename (columnname);

CREATE INDEX tablename\_column1name\_column2name\_idx ON tablename (column1name, column2name);

#### **Use Wildcards at the End of a Phrase**

In MySQL, wildcards are used in conjunction with the LIKE operator and NOT LIKE operator. They are used to search for data matching some search criteria. You can learn more about wildcards [here](https://www.guru99.com/wildcards.html).

Wildcards result in the most expansive scan when searching for data, which is very inefficient. Leading wildcards are the most inefficient, especially when combined with ending wild cards. In such a case, MySQL has to search all the records for a match. Thus you should avoid leading wild cards. See the queries below, one is using a leading wildcard and another one is using an ending wildcard.

SELECT \* FROM city WHERE name LIKE '%Al%';

SELECT \* FROM city WHERE name LIKE 'Al%';

#### **Specify Columns in SELECT Function**

SELECT \* (select all) is used as a shortcut to query all columns available in a table. This requires more resources than using a SELECT statement with only the columns you need for that specific query. For example, a customer table with 20 different columns and a hundred thousand entries. If you want to select a city with ID and Name only; try to use

SELECT ID, Name, District FROM city;

instead of

SELECT \* FROM city;

The second example will take more time to run to completion.

#### **Avoid SELECT DISTINCT**

DISTINCT is used to remove duplicate rows with SELECT statements. The DISTINCT command requires more sorting and reading of the database, which requires more processing power. DISTINCT can be replaced with GROUP BY to get the same results. See the two queries below.

SELECT col1, col2 FROM table GROUP BY col1, col2;

SELECT DISTINCT col1, col2 FROM table;

#### **Use LIMIT**

Sometimes we need a specified number of rows from a result set. The LIMIT clause is used in the query to specify the number of rows instead of fetching the whole result set. Fetching the entire result set requires more resources compared to fetching a specified number of rows. See the queries below, one without LIMIT, another one with the LIMIT clause.

SELECT ID, Name, District FROM city;

SELECT ID, Name, District FROM city LIMIT 10;

#### **MySQL Query Caching**

MySQL Query Caching provides database caching functionality. The SELECT statement text and the retrieved result are stored in the cache. When you make a similar query to the one already in the cache, MySQL will respond and give a query already in the cache. In this way, fewer resources are used, and your query runs faster.

This works best with a database where more select queries are made. Once the table is updated, the cached query and result become invalid. Thus, caching may not work with an application that updates the table frequently.

The command below is used to check if query cache is enabled in MySQL.

SHOW VARIABLES LIKE 'have\_query\_cache';

If the query cache is not set, set the query cache by following guidelines on [MySQL Documentation](https://dev.mysql.com/doc/refman/5.6/en/query-cache-configuration.html#:~:text=To%20set%20the%20size%20of,default%20for%20query_cache_type%20of%200.).

#### **Converting OUTER JOINs to INNER JOINs**

An INNER JOIN returns rows that contain columns from both tables. Unlike INNER JOIN, OUTER JOIN returns rows where no matches have been found on both tables. Therefore, OUTER JOIN does more work than INNER JOIN, increasing total execution time.

Use INNER JOIN whenever possible. It would be a waste of performance to use OUTER JOIN when you don’t need the data outside specified columns. We have a sample database with two tables as follows:

1. student - student\_id, first\_name, last\_name
2. orders - id, date, amount, customer\_id

An INNER JOIN query to the table would be as it is shown below.

SELECT C.id, C.name, O.amount, O.date FROM customers C

INNER JOIN orders O ON O.customer\_id = C.id;

An OUTER JOIN query to the table would be as it is shown below. NOTE: MySQL does not support FULL OUTER JOIN, but other SQL dialects such as [PostgreSQL](https://www.postgresql.org/) do.

SELECT C.id, C.name, O.amount, O.date FROM customers C

FULL OUTER JOIN orders O ON O.customer\_id = C.id;

#### **Optimize LIKE Statements with UNION Clause**

The OR operator is used to combine two Boolean expressions and return true when either of the conditions is met. When using comparison operator ‘or’ in a query, MySQL optimizer may incorrectly choose a full table scan to retrieve the result set. This makes the query run slower. A UNION clause runs faster and gives the same result.

Consider the query below:

SELECT \* FROM city WHERE Name LIKE 'C%' or District LIKE 'C%';

Below are the optimized versions of the query above using the UNION ALL and UNION operators, respectively.

SELECT \* FROM city WHERE Name LIKE 'C%' UNION ALL SELECT \* FROM city WHERE District LIKE 'C%';

SELECT \* FROM city WHERE Name LIKE 'C%' UNION SELECT \* FROM city WHERE District LIKE 'C%';

In the first query above, we used UNION ALL, while the second one, we have used UNION. By default, UNION returns distinct rows while UNION ALL allows duplicate rows. UNION generally runs faster than UNION ALL.

### **Conclusion**

MySQL development is ongoing. More tips to optimize queries are developed every day. This article is a guide on how to make better queries and make more stable database applications. Query with no doubt.

**MySQL Complex Query**

### **What is a complex MySQL query?**

Complex MySQL queries search data using more than one parameter and may comprise a combination of several joins across multiple tables and quite a few nested subqueries (queries nested within another query). Complex queries also frequently involve heavy use of AND and OR clauses.

Complex queries are typically used to retrieve complex data from multiple tables. Advanced queries can also be used for reporting, joining multiple tables, nesting queries, and transaction locking.

#### **Complex MySQL query with multiple SELECT statements**

For analytical purposes, it’s often necessary to fetch data from several different tables to form a single result table. Thus, complex MySQL queries with multiple SELECT statements are the most common advanced queries used by DBAs and developers. When you combine the results of multiple SELECT statements, you can choose what to include in the result table. This is exactly what makes them so popular.

In case I spent too many words so far, let’s remind ourselves of the original question – “How to write a complex SELECT query?”. And let’s start with a complex query.

|  |  |
| --- | --- |
|  | SELECT  country.country\_name\_eng,  SUM(CASE WHEN call.id IS NOT NULL THEN 1 ELSE 0 END) AS calls,  AVG(ISNULL(DATEDIFF(SECOND, call.start\_time, call.end\_time),0)) AS avg\_difference  FROM country  LEFT JOIN city ON city.country\_id = country.id  LEFT JOIN customer ON city.id = customer.city\_id  LEFT JOIN call ON call.customer\_id = customer.id  GROUP BY  country.id,  country.country\_name\_eng  HAVING AVG(ISNULL(DATEDIFF(SECOND, call.start\_time, call.end\_time),0)) > (SELECT AVG(DATEDIFF(SECOND, call.start\_time, call.end\_time)) FROM call)  ORDER BY calls DESC, country.id ASC; |

And this is what query returns:

[](https://www.sqlshack.com/wp-content/uploads/2020/02/complex-sql-select-query-e1580832567657.jpeg)

As you can see, we have a complex query and 2 rows in the result. Without any comments, we can’t easily say what does this query does and how it works.

# Reference

1. <https://www.javatpoint.com/mysql-transaction>
2. <https://www.tutorialspoint.com/mysql/mysql-transactions.htm>
3. <https://www.w3resource.com/mysql/mysql-transaction.php>
4. <https://www.thegeekdiary.com/beginners-guide-to-mysql-user-management/>
5. <https://www.w3resource.com/mysql/mysql-security.php>
6. <https://qxf2.com/blog/mysql-query-execution/#:~:text=Query%20execution%20is%20not%20that,sends%20them%20to%20the%20client>.
7. <https://www.section.io/engineering-education/mysql-query-performance-optimization-tips/>
8. <https://www.sqlshack.com/learn-sql-how-to-write-a-complex-select-query/>
9. <https://blog.devart.com/how-to-write-complex-mysql-queries.html#:~:text=What%20is%20a%20complex%20MySQL,of%20AND%20and%20OR%20clauses>.