SQL Database Administration

Lesson 10

Lesson outline

- Start, Stop, and Restart MySQL Server
- Users, Roles, and Privileges
- Show commands
- Backup and Restore
- Database maintenance

Start, Stop, and Restart MySQL Server

Start MySQL Server on Linux

On Linux, you can start the server with the following commands using service, init.d, and systemd

Start MySQL Server using service

```
sudo service mysql start
```

Start MySQL Server on Linux

On Linux, you can start the server with the following commands using service, init.d, and systemd

Start MySQL Server using service

```
sudo service mysql start
```

Start MySQL Server using using init.d

```
sudo /etc/init.d/mysql start
```

Start MySQL Server using systemd

```
sudo systemctl start mysqld
```

Start MySQL Server on Windows

On Windows, you can start the MySQL Server using the mysqld program as follows:

First, open the Run dialog by pressing Windows+R keyboards:
Second, type cmd and press Enter:
Third, type mysqld and press Enter:

```
mysqld
```

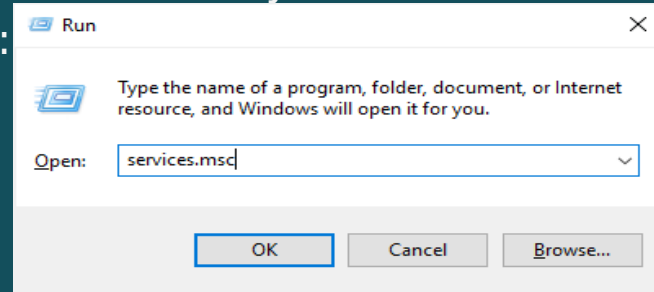If the bin folder is not in the Windows path environment, you can navigate to the bin folder e.g., C:\Program Files\MySQL\MySQL Server 8.0\bin\ and use the mysqld command.
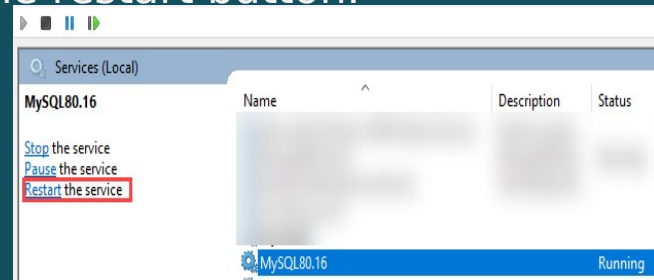
Restart MySQL Server on Windows

If MySQL installed as a Window service, you follow these steps to restart the MySQL Server:

First, open the Run window by using the Windows+R keyboard.

Second, type services.msc and press Enter:

Third, select the MySQL service and click the restart button.

Restart MySQL Server on Linux

You use the following command to restart the MySQL server On Linux:

```
service mysql restart
```

Or you can use the init.d to start the MySQL service:

```
/etc/init.d/mysqld restart
```

Stop MySQL Server on Windows

On Windows, you can stop MySQL Server using the program mysqladmin.

The program mysqladmin locates in the folder <path_to_installation_dir>\bin, where path_to_installation_dir is the path to the installation directory e.g., C:\Program Files\MySQL\MySQL Server 8.0\

Typically, you should add the pathname of the MySQL bin directory to Windows path environment variable to access any MySQL programs in the bin directory faster.

To stop MySQL, you follow these steps:

First, launch the Command Prompt by pressing Windows+R to open the Run box and type cmd and press Enter.

Second, navigate to the bin folder of the MySQL if it is not in the Window path environment.

```
mysqladmin -u root -p shutdown
Enter password: ********
```

Stop MySQL Server on Windows

It prompts for a password of the root account. You need to enter the password and press the Enter keyboard.

The program will stop the MySQL Server.

Stop MySQL Server on Linux

To stop MySQL Server on Linux, you use the following command:

```
/etc/init.d/mysqld stop
```

Some Linux distributions provide server command:

```
service mysqld stop
```

Or

```
service mysql stop
```

MySQL CREATE USER syntax

The CREATE USER statement creates a new user in the database server.
Here is the basic syntax of the CREATE USER statement:

```
CREATE USER [IF NOT EXISTS] account_name
IDENTIFIED BY 'password';
```

In this syntax:

First, specify the account name after the CREATE USER keywords. The account name has two parts: username and hostname, separated by the @ sign:

```
username@hostname
```

MySQL CREATE USER syntax

The username is the name of the user. And hostname is the name of the host from which the user connects to the MySQL Server.

The hostname part of the account name is optional. If you omit it, the user can connect from any host.

An account name without a hostname is equivalent to:

```
username@%
```

If the username and hostname contains special characters such as space or -, you need to quote the username and hostname separately as follows:

```
'username'@'hostname'
```

MySQL CREATE USER syntax

Besides the single quote ('), you can use backticks (`) or double quotation mark (").

Second, specify the password for the user after the IDENTIFIED BY keywords.

The IF NOT EXISTS option conditionally create a new user only if it does not exist.

Note that the CREATE USER statement creates a new user without any privileges. To grant privileges to the user, you use the GRANT statement.

MySQL CREATE USER example

First, connect to the MySQL Server using the mysql client tool:

```
mysql -u root -p
```

Enter the password for the root account and press Enter:

```
Enter password: ********
```

Second, show users from the current MySQL Server:

```
mysql> select user from mysql.user;
```

MySQL CREATE USER example

Here is the output:

```
+------------------+
| user             |
+------------------+
| mysql.infoschema |
| mysql.saession   |
| mysql.sys        |
| root             |
+------------------+
```

Third, create a new user called bob:

```
mysql> create user bob@localhost identified by 'Secure1pass!';
```

Fourth, show all users again:

```
mysql> select user from mysql.user;
```

MySQL CREATE USER example

The output will be:

```
+------------------+
| user             |
+------------------+
| bob              |
| mysql.infoschema |
| mysql.session    |
| mysql.sys        |
| root             |
+------------------+
5 rows in set (0.00 sec)
```

The user bob has been created successfully.
Fifth, open a second session and log in to the MySQL as bob:

```
mysql -u bob -p
```

Input the password for bob and press Enter:

```
Enter password: ********
```

MySQL CREATE USER example

Sixth, show the databases that bob has access:

```
mysql> show databases;
```

Here is the list of databases that bob can access

```
+--------------------+
| Database           |
+--------------------+
| information_schema |
+--------------------+
1 row in set (0.01 sec)
```

Seventh, go to the session of the user root and create a new database called bobdb:

```
mysql> create database bobdb;
```

MySQL CREATE USER example

Eight, select the database bobdb:

```
mysql> use bobdb;
```

Ninth, create a new table called lists:

```
mysql> create table lists(
    -> id int auto_increment primary key,
    -> todo varchar(100) not null,
    -> completed bool default false);
```

Notice that when you press Enter, instead of showing the mysql> command, the mysql tool shows the -> that accepts new clause of the statement.

Tenth, grant all privileges on the bobdb to bob:

```
mysql> grant all privileges on bobdb.* to bob@localhost;
```

MySQL CREATE USER example

Note that you will learn how to grant privileges to a user in the GRANT tutorial.

Eleventh, go to the bob's session and show databases:

```
mysql> show databases;
```

Now, bob can see the bobdb:

```
+---------------------+
| Database            |
+---------------------+
| bobdb               |
| information_schema  |
+---------------------+
2 rows in set (0.00 sec)
```

Twelfth, select the database bobdb:

```
mysql> use bobdb;
```

MySQL CREATE USER example

Thirteenth, show the tables from the bobdb database:

```
mysql> show tables;
```

The user bob can see the lists table

```
+-----------------+
| Tables_in_bobdb |
+-----------------+
| lists           |
+-----------------+
1 row in set (0.00 sec)
```

Fourteenth, insert a row into the lists table:

```
mysql> insert into lists(todo) values('Learn MySQL');
```

MySQL CREATE USER example

Fifteenth, query data from the lists table:

```
mysql> select * from lists;
```

This is the output:

```
+----+--------------+-----------+
| id | todo         | completed |
+----+--------------+-----------+
|  1 | Learn MySQL  |         0 |
+----+--------------+-----------+
1 row in set (0.00 sec)
```

So the user bob can do everything in the bobdb database.
Finally, disconnect from the MySQL Server from both sessions:

```
mysql> exit
```

MySQL GRANT - Introduction to the MySQL GRANT statement

The CREATE USER statement creates one or more user accounts with no privileges. It means that the user accounts can log in to the MySQL Server, but cannot do anything such as selecting a database and querying data from tables.

To allow user accounts to work with database objects, you need to grant the user accounts privileges. And the GRANT statement grants a user account one or more privileges.

The following illustrates the basic syntax of the GRANT statement:

```
GRANT privilege [,privilege],..
ON privilege_level
TO account_name;
```

MySQL GRANT - Introduction to the MySQL GRANT statement

In this syntax:

First, specify one or more privileges after the GRANT keyword. If you grant multiple privileges, you need to separate privileges by commas.

This example grants the SELECT privilege on the table employees in the sample database to the user acount bob@localhost:

```
GRANT SELECT
ON employees
TO bob@localhost;
```
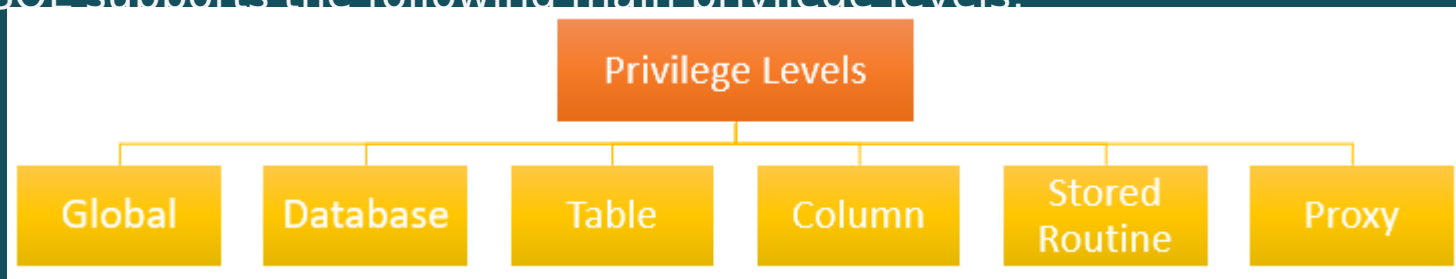
MySQL GRANT - Introduction to the MySQL GRANT statement

The following example grants UPDATE, DELETE, and INSERT privileges on the table employees to bob@localhost:

```
GRANT INSERT, UPDATE, DELETE

ON employees

TO bob@localhost;
```

Second, specify the privilege_level that determines the level to which the privileges apply.

MySQL supports the following main privilege levels:

MySQL GRANT - Introduction to the MySQL GRANT statement

Global privileges apply to all databases in a MySQL Server. To assign global privileges, you use the *.* syntax, for example:

```
GRANT SELECT
ON *.*
TO bob@localhost;
```

The account user bob@localhost can query data from all tables in all database of the current MySQL Server.

Database privileges apply to all objects in a database. To assign database-level privileges, you use the ON database_name.* syntax, for example:

```
GRANT INSERT
ON classicmodels.*
TO bob@localhost;
```

MySQL GRANT - Introduction to the MySQL GRANT statement

In this example, bob@localhost can insert data into all tables in the classicmodels database.

Table privileges apply to all columns in a table. To assign table-level privileges, you use the ON database_name.table_name syntax, for example:

```
GRANT DELETE
ON classicmodels.employees
TO bob@localhsot;
```

In this example, bob@localhost can delete rows from the table employees in the database classicmodels.

MySQL GRANT - Introduction to the MySQL GRANT statement

If you skip the database name, MySQL uses the default database or issues an error if there is no default database.

Column privileges apply to single columns in a table. You must specify the column or columns for each privilege, for example:

```
GRANT
    SELECT (employeeNumner,lastName, firstName,email),
    UPDATE(lastName)
ON employees
TO bob@localhost;
```

MySQL GRANT - Introduction to the MySQL GRANT statement

In this example, bob@localhost can select data from four columns employeeNumber, lastName, firstName, and email and update only the lastName column in the employees table.

Stored routine privileges apply to stored procedures and stored functions, for example:

```
GRANT EXECUTE
ON PROCEDURE CheckCredit
TO bob@localhost;
```

MySQL GRANT - Introduction to the MySQL GRANT statement

In this example, bob@localhost can execute the stored procedure CheckCredit in the current database.

Proxy user privileges allow one user to be a proxy for another. The proxy user gets all privileges of the proxied user. For example:

```
GRANT PROXY
ON root
TO alice@localhost;
```

MySQL GRANT - Introduction to the MySQL GRANT statement

In this example, alice@localhost assumes all privileges of root.

Finally, specify the account name of the user that you want to grant privileges after the TO keyword.

Notice that in order to use the GRANT statement, you must have the GRANT OPTION privilege and the privileges that you are granting. If the read_only system variable is enabled, you need to have the SUPER privilege to execute the GRANT statement.

MySQL GRANT statement examples

Typically, you use the CREATE USER statement to create a new user account first and then use the GRANT statement to grant privileges to the user.

First, create a new user called super@localhost by using the following CREATE TABLE statement:

```
CREATE USER super@localhost
IDENTIFIED BY 'Secure1Pass!';
```

Second, show the privileges assigned to super@localhost user by using the SHOW GRANTS statement.

```
SHOW GRANTS FOR super@localhost;
```

Grants for super@localhost
▶	GRANT USAGE ON *.* TO `super`@`localhost`

MySQL GRANT statement examples

The USAGE means that the super@localhost can log in the database but has no privilege.

Third, grant all privileges in all databases in the current database server to super@localhost:

```
GRANT ALL
ON classicmodels.*
TO super@localhost;
```

Fourth, use the SHOW GRANTS statement again:

```
SHOW GRANTS FOR super@localhost;
```

Grants for super@localhost
GRANT USAGE ON *.* TO `super`@`localhost`
GRANT ALL PRIVILEGES ON `classicmodels`.* TO `super`@`localhost`

Permissible privileges for GRANT statement

The following table illustrates all permissible privileges that you can use for the GRANT and REVOKE statement:

Read here: https://www.mysqltutorial.org/mysql-grant.aspx end of page

Introduction to MySQL SHOW GRANTS statement

The MySQL SHOW GRANTS statement returns all privileges and roles granted to an account user or role.

Here is the basic syntax of the SHOW GRANTS statement:

```
SHOW GRANTS
[FOR {user | role}
[USING role [, role] ...]]
```

In this syntax:

First, specify the name of the user account or role that you want to display the privileges that are previously granted to the user account or role after the FOR keyword. If you skip the FOR clause, the SHOW GRANTS returns the privileges of the current user.

Introduction to MySQL SHOW GRANTS statement

Second, use the USING clause to examine the privileges associated with roles for the user. The roles that you specify in the USING clause must previously granted to the user.

To execute the SHOW GRANTS statement, you need to have SELECT privilege for the mysql system database, except to show privileges and roles for the current user.

MySQL SHOW GRANTS statement examples

Let's take some examples of using the MySQL SHOW GRANTS statement.

A) Using MySQL SHOW GRANTS to display the privileges granted for the current user

The following statement uses the SHOW GRANTS statement to display the privileges granted for the current user:

```
SHOW GRANTS;
```

It is equivalent to the following statement:

```
SHOW GRANTS FOR CURRENT_USER;
```

and

```
SHOW GRANTS FOR CURRENT_USER();
```

MySQL SHOW GRANTS statement examples

Both CURRENT_USER and CURRENT_USER() return the current user.

B) Using MySQL SHOW GRANTS to display the privileges granted for a user
First, create a new database named vehicles:

```
CREATE DATABASE vehicles;
```

Second, select the database vehicles :

```
USE vehicles;
```

Third, create a new table called cars in the vehicles database:

```
CREATE TABLE cars (
    id INT AUTO_INCREMENT,
    make VARCHAR(100) NOT NULL,
    model VARCHAR(100) NOT NULL,
    PRIMARY KEY (id)
);
```

MySQL SHOW GRANTS statement examples

Fourth, create a new user called musk@localhost:

```
CREATE USER musk@localhost

IDENTIFIED BY 'Super1Pass!';
```

Fifth, show the default privileges granted to the user musk@localhost:

```
SHOW GRANTS

FOR musk@localhost;
```

Grants for musk@localhost
▶	GRANT USAGE ON *.* TO `musk`@`localhost`

The GRANT USAGE is the synonym of no privilege. By default, when a new user created, it has no privilege.

Sixth, grant all privileges on the vehicles database to the user musk@localhost:

```
GRANT ALL

ON vehicles.*

TO musk@localhost;
```

MySQL SHOW GRANTS statement examples

Finally, show the privileges granted for the user musk@localhost:

```
SHOW GRANTS
FOR musk@localhost;
```

	Grants for musk@localhost
▶	GRANT USAGE ON *.* TO `musk`@`localhost`
	GRANT ALL PRIVILEGES ON `vehicles`.* TO `musk`@`localhost`

C) Using MySQL SHOW GRANTS to display the privileges granted for a role
First, create a new role called writer@localhost:

```
CREATE ROLE writer@localhost;
```

Second, show privileges granted for the role writer@localhost:

```
SHOW GRANTS
FOR writer@localhost;
```

	Grants for writer@localhost
▶	GRANT USAGE ON *.* TO `writer`@`localhost`

MySQL SHOW GRANTS statement examples

Third, grant SELECT, INSERT, UPDATE, and DELETE privileges on the vehicles database to the writer@localhost:

```
GRANT
    SELECT,
    INSERT,
    UPDATE,
    DELETE
ON vehicles.*
TO writer@localhost;
```

Fourth, show privileges granted for the role writer@localhost:

```
SHOW GRANTS
    FOR writer@localhost;
```

Grants for writer@localhost
GRANT USAGE ON *.* TO `writer`@`localhost`
GRANT SELECT, INSERT, UPDATE, DELETE ON `vechicles`.* TO `writer`@`localhost`

MySQL SHOW GRANTS statement examples

D) Using MySQL SHOW GRANTS with USING clause example

First, create a new account user called jame@localhost:

```
CREATE USER jame@localhost
IDENTIFIED BY 'Secret@Pass1';
```

Second, grant the EXECUTE privilege to the user jame@localhost:

```
GRANT EXECUTE
ON vehicles.*
TO jame@localhost;
```

Third, grant the role writer@localhost to the user jame@localhost:

```
GRANT writer@localhost
TO jame@localhost;
```

MySQL SHOW GRANTS statement examples

to be continued

MySQL SHOW DATABASES: List All Databases in MySQL

To list all databases on a MySQL server host, you use the SHOW DATABASES command as follows:

```
SHOW DATABASES;
```

For example, to list all database in the local MySQL database server, first login to the database server as follows:

```
>mysql -u root -p
Enter password: **********
mysql>
```

MySQL SHOW DATABASES: List All Databases in MySQL

And then use the SHOW DATABASES command:

```
mysql> SHOW DATABASES;

+--------------------+
| Database           |
+--------------------+
| classicmodels      |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| test               |
+--------------------+
6 rows in set (0.00 sec)
```

MySQL SHOW DATABASES: List All Databases in MySQL

The SHOW SCHEMAS command is a synonym for SHOW DATABASES, therefore the following command returns the same result as the one above:

```
SHOW SCHEMAS;
```

If you want to query the database that matches a specific pattern, you use the LIKE clause as follows:

```
SHOW DATABASES LIKE pattern;
```

```
SHOW DATABASES LIKE '%schema';

+--------------------+
| Database (%schema) |
+--------------------+
| information_schema |
| performance_schema |
+--------------------+
2 rows in set (0.00 sec)
```

For example, the following s   base that ends with the string 'schema';

MySQL SHOW DATABASES: List All Databases in MySQL

For example, the following statement returns database that ends with the string 'schema';

```
SHOW DATABASES LIKE '%schema';

+---------------------+
| Database (%schema)  |
+---------------------+
| information_schema  |
| performance_schema  |
+---------------------+
2 rows in set (0.00 sec)
```

It is important to note that if the MySQL database server started with --skip-show-database, you cannot use the SHOW DATABASES statement unless you have the SHOW DATABASES privilege.

MySQL SHOW DATABASES: List All Databases in MySQL

If the condition in the LIKE clause is not sufficient, you can query the database information directly from the schemata table in the information_schema database.

For example, the following query returns the same result as the SHOW DATABASES command.

```sql
SELECT schema_name
FROM information_schema.schemata;
```

The following SELECT statement returns databases whose names end with 'schema' or 's'.

```sql
SELECT schema_name
FROM information_schema.schemata
WHERE schema_name LIKE '%schema' OR
      schema_name LIKE '%s';
```

MySQL SHOW DATABASES: List All Databases in MySQL

It returns the following result set:

```
+--------------------+
| SCHEMA_NAME        |
+--------------------+
| information_schema |
| performance_schema |
| sys                |
| classicmodels      |
+--------------------+
4 rows in set (0.00 sec)
```

So far, you have learned how to show all databases in the MySQL server using the SHOW DATABASES command or querying from the schemata table in the information_schema database.

MySQL SHOW TABLES: List Tables In a MySQL Database

To list tables in a MySQL database, you follow these steps:

1. Login to the MySQL database server using a MySQL client such as mysql
2. Switch to a specific database using the USE statement.
3. Use the SHOW TABLES command.

The following illustrates the syntax of the MySQL SHOW TABLES command:

```
SHOW TABLES;
```

MySQL SHOW TABLES examples

The following example shows you how to list the table in the `classicmodels` database.

Step 1. Connect to the MySQL database server:

```
>mysql -u root -p
Enter password: **********
mysql>
```

Step 2. Switch to `classicmodels` database:

```
mysql> use classicmodels;
Database changed
mysql>
```

MySQL SHOW TABLES examples

Step 3. Show tables in the `classicmodels` database:

```
> show tables;

+-------------------------+
| Tables_in_classicmodels |
+-------------------------+
| customers               |
| employees               |
| offices                 |
| orderdetails            |
| orders                  |
| payments                |
| productlines            |
| products                |
+-------------------------+
8 rows in set (0.00 sec)
```

MySQL SHOW TABLES examples

The `SHOW TABLES` command allows you to show if a table is a base table or a view. To include the table type in the result, you use the following form of the `SHOW TABLES` statement.

```
SHOW FULL TABLES;
```

Let's create a view in the `classicmodels` database called `contacts` that includes first name, last name and phone from the `employees` and `customers` tables for the demonstration.

```
CREATE VIEW contacts
AS
SELECT lastName, firstName, extension as phone
FROM employees
UNION
SELECT contactFirstName, contactLastName, phone
FROM customers;
```

MySQL SHOW TABLES examples

As you can see, all the tables are the base tables except for the `contacts` table which is a view.

For the database that has many tables, showing all tables at a time may not be intuitive.

Fortunately, the `SHOW TABLES` command provides you with an option that allows you to filter the returned tables using the `LIKE` operator or an expression in the `WHERE` clause as follows:

```
SHOW TABLES LIKE pattern;
```

```
SHOW TABLES WHERE expression;
```

MySQL SHOW TABLES examples

For example, to shows all tables in the `classicmodels` database that start with the letter `p`, you use the following statement:

```
> SHOW TABLES LIKE 'p%';

+------------------------------+
| Tables_in_classicmodels (p%) |
+------------------------------+
| payments                     |
| productlines                 |
| products                     |
+------------------------------+
3 rows in set (0.00 sec)
```

MySQL SHOW TABLES examples

Or to show the tables that end with the string `'es'`, you use the following statement:

```
> SHOW TABLES LIKE '%es';

+-------------------------------+
| Tables_in_classicmodels (%es) |
+-------------------------------+
| employees                     |
| offices                       |
| productlines                  |
+-------------------------------+
3 rows in set (0.00 sec)
```

MySQL SHOW TABLES examples

The following statement illustrates how to use the `WHERE` clause in the `SHOW TABLES` statement to list all the views in the `classicmodels` database.

```
> SHOW FULL TABLES WHERE table_type = 'VIEW';

+-----------------------------+------------+
| Tables_in_classicmodels     | Table_type |
+-----------------------------+------------+
| contacts                    | VIEW       |
+-----------------------------+------------+
1 row in set (0.00 sec)
```

Sometimes, you want to see the tables in the database that you are not connected to. In this case, you can use the `FROM` clause of the `SHOW TABLES` statement to specify the database from which you want to show the tables.

MySQL SHOW TABLES examples

The following example demonstrates how to show tables that start with `'time'`;

```
> SHOW TABLES FROM mysql LIKE 'time%';

+---------------------------+
| Tables_in_mysql (time%)   |
+---------------------------+
| time_zone                 |
| time_zone_leap_second     |
| time_zone_name            |
| time_zone_transition      |
| time_zone_transition_type |
+---------------------------+
5 rows in set (0.00 sec)
```

MySQL SHOW TABLES examples

The following statement is equivalent to the statement above but it uses `IN` instead of `FROM`.

```
SHOW TABLES IN mysql LIKE 'time%';
```

It's important to note that if you don't have privileges for a base table or view, it won't show up in the result set of the `SHOW TABLES` command.

MySQL SHOW COLUMNS and DESCRIBE: List All Columns in a Table

To show all columns of a table, you use the following steps:

1. Login to the MySQL database server.
2. Switch to a specific database.
3. Use the DESCRIBE statement.

The following example demonstrates how to display columns of the orders table in the classicmodels database.

Step 1. Login to the MySQL database.

```
>mysql -u root -p

Enter password: **********

mysql>
```

MySQL SHOW COLUMNS and DESCRIBE: List All Columns in a Table

Step 2. Issue the USE command to switch to the database to classicmodels:

```
mysql> USE classicmodels;

Database changed

mysql>
```

MySQL SHOW COLUMNS and DESCRIBE: List All Columns in a Table

Step 3. Use the DESCRIBE statement.

```
mysql> DESCRIBE orders;

+----------------+-------------+------+-----+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+-----+---------+-------+
| orderNumber    | int(11)     | NO   | PRI | NULL    |       |
| orderDate      | date        | NO   |     | NULL    |       |
| requiredDate   | date        | NO   |     | NULL    |       |
| shippedDate    | date        | YES  |     | NULL    |       |
| status         | varchar(15) | NO   |     | NULL    |       |
| comments       | text        | YES  |     | NULL    |       |
| customerNumber | int(11)     | NO   | MUL | NULL    |       |
+----------------+-------------+------+-----+---------+-------+
7 rows in set (0.01 sec)
```

MySQL SHOW COLUMNS and DESCRIBE: List All Columns in a Table

In practice, you use the DESC statement which is a shorthand of the DESCRIBE statement. For example, the following statement is equivalent to the DESCRIBE above:

```
DESC orders;
```

MySQL SHOW COLUMNS and DESCRIBE: List All Columns in a Table

MySQL SHOW COLUMNS command

The more flexible way to get a list of columns in a table is to use the MySQL SHOW COLUMNS command.

```
SHOW COLUMNS FROM table_name;
```

To show columns of a table, you specific the table name in the FROM clause of the SHOW COLUMNS statement. To show columns of a table in a database that is not the current database, you use the following form:

```
SHOW COLUMNS FROM database_name.table_name;
```

MySQL SHOW COLUMNS and DESCRIBE: List All Columns in a Table

Or

```
SHOW COLUMNS FROM table_name IN database_name;
```

For example, to get the columns of the orders table, you use the SHOW COLUMNS statement as follows:

```
SHOW COLUMNS FROM orders;
```

As you can see the result of this SHOW COLUMNS command is the same as the result of the DESC statement.

To get more information about the column, you add the FULL keyword to the SHOW COLUMNS command as follows:

```
SHOW FULL COLUMNS FROM table_name;
```

MySQL SHOW COLUMNS and DESCRIBE: List All Columns in a Table

For example, the following statement lists all columns of the payments table in the classicmodels database. (Cross section shown below)

```
mysql> SHOW FULL COLUMNS FROM payments \G;
*************************** 1. row ***************************
     Field: customerNumber
      Type: int(11)
 Collation: NULL
      Null: NO
       Key: PRI
   Default: NULL
     Extra:
Privileges: select,insert,update,references
   Comment:
*************************** 2. row ***************************
```

MySQL SHOW COLUMNS and DESCRIBE: List All Columns in a Table

As you can see, the SHOW FULL COLUMNS command adds the collation, privileges, and comment columns to the result set.

The SHOW COLUMNS command allows you to filter the columns of the table by using the LIKE operator or WHERE clause:

```
SHOW COLUMNS FROM table_name LIKE pattern;


SHOW COLUMNS FROM table_name WHERE expression;
```

MySQL SHOW COLUMNS and DESCRIBE: List All Columns in a Table

For example, to show only columns that start with the letter c, you use the LIKE operator as follows:

```
mysql> SHOW COLUMNS FROM payments LIKE 'c%';
+----------------+-------------+------+-----+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+-----+---------+-------+
| customerNumber | int(11)     | NO   | PRI | NULL    |       |
| checkNumber    | varchar(50) | NO   | PRI | NULL    |       |
+----------------+-------------+------+-----+---------+-------+
2 rows in set (0.01 sec)
```

MySQL SHOW PROCESSLIST

Sometimes, you may get the "too many connections" error returned by the MySQL Server. To find out the reasons, you can use the SHOW PROCESSLIST command.

The SHOW PROCESSLIST command returns all currently running threads. You then can terminate the idle threads with the KILL statement.

The following shows the syntax of the SHOW PROCESSLIST command:

```
SHOW [FULL] PROCESSLIST;
```

MySQL SHOW PROCESSLIST

Accounts with the PROCESS privilege can view all threads. Otherwise, they can view only threads associated with their accounts.

The following shows an example of the output of the SHOW PROCESSLIST command:

```
mysql>SHOW PROCESSLIST;

+----+-----------------+-----------------+---------------+---------+------+----------------
| Id | User            | Host            | db            | Command | Time | State
+----+-----------------+-----------------+---------------+---------+------+----------------
|  4 | event_scheduler | localhost       | NULL          | Daemon  | 2246 | Waiting on emp
| 14 | root            | localhost:50924 | NULL          | Query   |    0 | starting
| 15 | car             | localhost:50933 | classicmodels | Sleep   |    2 |
+----+-----------------+-----------------+---------------+---------+------+----------------
3 rows in set (0.00 sec)
```

MySQL SHOW PROCESSLIST

The output of the SHOW PROCESSLIST command consists of the following columns:

Id

The client process's Id

User

The username associated with the thread.

Host

The host to which the client is connected

DB

The default database if one selected otherwise NULL

MySQL SHOW PROCESSLIST

Command
The command type

Time
The number of seconds that the current thread has been in its current state.

State
The thread state which represents an action, event, or state that indicates what thread is executing.

MySQL SHOW PROCESSLIST

Info

The statement is being executed, or NULL if it is not executing any statement. If you do not use the FULL keyword in the SHOW PROCESSLIST command, then only the first 100 characters of each statement are returned in the Info column.

MySQL Backup: Backing Up Using mysqldump Tool

The mysqldump tool allows you to make a backup of one or more databases by generating a text file that contains SQL statements which can re-create the databases from scratch.

The mysqldump tool is located in the root/bin directory of the MySQL installation directory.

To access the mysqldump tool, you navigate to the root/bin folder and use the mysqldump command with the following options.

MySQL Backup: Backing Up Using mysqldump Tool

Here are the common mysqldump options:
add-drop-table

Includes a DROP TABLE statement for each table in the database.
add-locks

Includes LOCK TABLES and UNLOCK TABLES statements before and after each INSERT statement. It improves the data restoration speed from the dump file.
all-databases

Creates a dump of all databases on the MySQL server.

MySQL Backup: Backing Up Using mysqldump Tool

create-options

Includes ENGINE and CHARSET options in the CREATE TABLE statement for each table.
databases

Creates a dump of one or more databases.
disable-keys

Instructs MySQL to disable index updates during data load for MyISAM tables. MySQL will create indexes after mysqldump completes loading data. This option improves the speed of restoration.

MySQL Backup: Backing Up Using mysqldump Tool

extended-insert

Combines single-row INSERT statements into a single statement that insert multiple table rows; This option also helps speed up data restoration.
flush-logs

Flushes the server logs before dumping the data. This is useful in conjunction with incremental backups.
lock-tables

Ensures that the dump is a consistent snapshot by locking all the tables in a database during the dump.

MySQL Backup: Backing Up Using mysqldump Tool

no-data

create a dump file that contains statements necessary for re-creating the database structure only (only CREATE DATABASE, CREATE TABLE...), not the data (no INSERT statements).

opt
The mysqldump tool uses the opt by default.

The opt option enables the following options: add-drop-table, add-locks, create-options, disable-keys, extended-insert, lock-tables, quick, and set-charset.
To disable this option, you use skip-opt. If you want to skip each individual option, you use skip-<option_name>. For example, to skip the disable-keys option, you use skip-disable-keys option.

MySQL Backup: Backing Up Using mysqldump Tool

quick
Instructs mysqldump to not buffer tables in memory before writing to the file. This option speeds up dumps from big tables.

result-file
Specifies the path to the output dump file.

set-charset
Specifies the character set such as latin1 or utf8 of the database.

tables
Creates a dump of one or more tables.
where
Dumps only rows which satisfies a condition in the WHERE clause.

MySQL Backup: Backing Up Using mysqldump Tool

Using the mysqldump tool to make a backup of databases

Let's take some examples of using the mysqldump tool to backup database examples.

1) Using the mysqldump tool to make a backup of a single database

The following command backs up a single database from a MySQL Server:

> mysqldump --user=<username> --password=<password> --result-file=<path_to_backup_file> --databases <database_name>

MySQL Backup: Backing Up Using mysqldump Tool

In this syntax:

- The <username> is the user account that will login to the MySQL Server.
- The <password> is the password for the <username>.
- The <path_to_backup_file> is the path to the backup file.
- The --databases is an option that instructs the mysqldump tool to create a dump of the specified databases.
- The <database_name> is the name of the database that you want to back up.

MySQL Backup: Backing Up Using mysqldump Tool

For example, the following command creates a backup of the database classicmodels to the file c:\backup\classicmodels.sql:

```
> mysqldump --user=root --password=Supe!rPass1 --result-file=c:\backup\classicmodels.sql --databases classicmodels
```

2) Using the mysqldump tool to make a backup of multiple databases
To make a backup of multiple databases, you specify a list of the database names after the --database option:

```
> mysqldump --user=<username> --password=<password> --result-file=<path_to_backup_file> --databases <dbname1>[,<dbname2>, ...]
```

MySQL Backup: Backing Up Using mysqldump Tool

For example, the following command makes a backup of the classicmodels and world databases:

```
>  mysqldump  --user=root  --password=Supe!rPass1  --result-file=c:\backup\classicmodels_world.sql --databases classicmodels world
```

3) Using the mysqldump tool to make a backup of all databases
To make a backup of all databases in a MySQL Server, you use the –all-database option:

```
>  mysqldump  --user=<username>  --password=<password>  --result-file=<path_to_backup_file> --all-databases
```

MySQL Backup: Backing Up Using mysqldump Tool

The following statement makes a backup of all databases in the current MySQL server:

```
> mysqldump --user=root --password=Supe!rPass1 --result-file=c:\backup\all_databases.sql --all-databases
```

4) Using the mysqldump tool to make a backup of specific tables from a database

To make a backup of specific tables from a database, you use the following command:

```
> mysqldump --user=<username> --password=<password> --result-file=<path_to_backup_file> <database_name> <table_name>
```

MySQL Backup: Backing Up Using mysqldump Tool

You can also specify multiple tables after the database name, one after the other:

```
> mysqldump --user=<username> --password=<password> --result-file=<path_to_backup_file> <database_name> <table1> <table2> <table3>
```

For example, to make a backup of the employees table from the classicmodels database, you use the following command:

```
> mysqldump --user=root --password=Supe!rPass1 --result-file=c:\backup\employees.sql classicmodels employees
```

MySQL Backup: Backing Up Using mysqldump Tool

5) Using a mysqldump tool to make a backup of database structure only
To make a backup of the database structure only, you use the --no-data option:

```
>   mysqldump   --user=<username>   --password=<password>   --result-file=<path_to_backup_file> --no-data --databases <database_name>
```

The statement will generate a dump file that contains the SQL statement necessary to re-create the database structure. And the dump file does not contain INSERT statements.

For example, the following statement makes a backup of the database structure of the database classicmodels:

```
>   mysqldump   --user=root   --password=Supe!rPass1   --result-file=c:\backup\ classicmodels-ddl.sql --no-data --databases classicmodels
```

MySQL Backup: Backing Up Using mysqldump Tool

5) Using a mysqldump tool to make a backup of database structure only

To make a backup of the database structure only, you use the --no-data option:

```
> mysqldump --user=<username> --password=<password> --result-file=<path_to_backup_file> --no-data --databases <database_name>
```

The statement will generate a dump file that contains the SQL statement necessary to re-create the database structure. And the dump file does not contain INSERT statements.

For example, the following statement makes a backup of the database structure of the database classicmodels:

```
> mysqldump --user=root --password=Supe!rPass1 --result-file=c:\backup\classicmodels-ddl.sql --no-data --databases classicmodels
```

MySQL Restore: Restoring from a Dump File

Setting up a sample database

First, create a new database called mydb:

```
CREATE DATABASE mydb;
```

Second, use the mydb database:

```
USE mydb;
```

Third, create a new table tests with one column:

```
CREATE TABLE tests(
    id INT PRIMARY KEY
);
```

MySQL Restore: Restoring from a Dump File

Fourth, insert some rows into the tests table:

```
INSERT INTO tests(id)

VALUES(1),(2),(3);
```

Finally, use the mysqldump to dump the mydb database:

> *mysqldump --user=root --password=Supe!rPass1 --result-file=c:\backup\ mydb.sql --databases mydb*

Note that you must have the c:\backup directory available on your computer or server.

MySQL Restore: Restoring from a Dump File

Restoring an SQL dump file using the SOURCE command
To restore the mydb.sql SQL dump file, you follow these steps:
First, connect to MySQL server:

```
C:\>mysql -u root -p

Enter password: ********
```

Second, drop the mydb database:

```
mysql>drop database mydb;
```

Third, use the SOURCE command to load the dump file:

```
mysql>source c:\backup\mydb.sql
```

MySQL Restore: Restoring from a Dump File

The command created a database mydb, select the database, and execute other SQL statements. In addition, it showed all the possible warnings and errors.

It is recommended that you use the SOURCE command to restore a dump file because the SOURCE command returns very detailed information including warnings and errors.

Maintaining MySQL Database Tables

MySQL provides several useful statements that allow you to maintain database tables effectively. Those statements enable you to analyze, optimize, check, and repair database tables.

Analyze table statement

MySQL query optimizer is an important component of the MySQL server that creates an optimal query execution plan for a query. For a particular query, the query optimizer uses the stored key distribution and other factors to decide the order in which tables should be joined when you performing the join, and which index should be used for a specific table.

However, the key distributions can be sometimes inaccurate e.g., after you have done a lot of data changes in the table including insert, delete, or update. If the key distribution is not accurate, the query optimizer may pick a bad query execution plan that may cause a severe performance issue.

Maintaining MySQL Database Tables

To solve this problem, you can run the ANALYZE TABLE statement for the table e.g., the following statement analyzes the payments table in the sample database.

```
ANALYZE TABLE payments;
```

Table	Op	Msg_type	Msg_text
classicmodels.payments	analyze	status	OK

If there is no change to the table since the ANALYZE TABLE statement ran, MySQL will not analyze the table again. If you run the above statement again:

```
ANALYZE TABLE payments;
```

Table	Op	Msg_type	Msg_text
classicmodels.payments	analyze	status	Table is already up to date

Maintaining MySQL Database Tables

It is saying that the table is already up to date.
Optimize table statement

While working with the database, you do a lot of changes such as insert, update and delete data in the table that may cause the physical storage of the table fragmented. As a result, the performance of database server is degraded.

MySQL provides you with a statement that allows you to optimize the table to avoid this defragmenting problem. The following illustrates how to optimize a table:

```
OPTIMIZE TABLE table_name;
```

Maintaining MySQL Database Tables

It is recommended that you execute this statement for the tables that are updated frequently. For example, if you want to optimize the orders table to defragment it, you can perform the following statement:

```
OPTIMIZE TABLE orders;
```

	Table	Op	Msg_type	Msg_text
▶	classicmodels.orders	optimize	status	OK

Check table statement
Something wrong can happen to the database server e.g., the server was shutdown unexpectedly, error while writing data to the hard disk, etc. These situations could make the database operate incorrectly and in the worst case, it can be crashed.

Maintaining MySQL Database Tables

MySQL allows you to check the integrity of database tables by using the CHECK TABLE statement. The following illustrates the syntax of the CHECK TABLE statement:

```
CHECK TABLE table_name;
```

The CHECK TABLE statement checks both table and its indexes. For example, you can use the CHECK TABLE statement to check the orders table as follows:

```
CHECK TABLE orders;
```

Table	Op	Msg_type	Msg_text
classicmodels.orders	check	status	OK

The CHECK TABLE statement only detects problems in a database table but it does not repair them. To repair the table, you use the REPAIR TABLE statement.

Maintaining MySQL Database Tables

Repair table statement

The REPAIR TABLE statement allows you to repair some errors occurred in database tables. MySQL does not guarantee that the REPAIR TABLE statement can repair all errors that the tables may have.

The following is the syntax of the REPAIR TABLE statement:

```
REPAIR TABLE table_name;
```

Suppose there are some errors in the orders table and you need to fix them, you can use the REPAIR TABLE statement as the following query:

```
REPAIR TABLE employees;
```

Maintaining MySQL Database Tables

MySQL returns what it has done to the table and shows you whether the table was repaired or not.

Table	Op	Msg_type	Msg_text
classicmodels.orders	repair	status	OK

Quiz

Complete the commands on *mysqldump* and for each command save a separate file for the backup

Assignment

Congratulations!