Table of Contents

1. Installation

https://downloads.mariadb.org/mariadb/repositories/

```
yum install MariaDB-server Mariadb-client
systemctl enable mariadb.service
systemctl start mariadb.service
mysql -u root
```

2. Sercurity

```
mysql_secure_installation
mysql -u root -p
      my-file
      [client]
      user=myusername
      password=mypassword
mysql --defaults-file=/path/to/my-file
```

3. Administering MariaDB

The privileges or rights that we can grant to users break down into three main categories:

1. Global administrative user privileges
2. Database, table, and column user privileges
3. Miscellaneous user privileges and limits

Full documentation of the privileges:https://mariadb.com/kb/en/grant/

4. Creating user

```
CREATE USER 'username'@'host' IDENTIFIED BY 'password';
```

```
CREATE USER 'boyd'@'%' IDENTIFIED BY 'bomber';
CREATE USER 'tom'@'localhost' IDENTIFIED BY 'retail';
CREATE USER 'richard'@'powr.example.net' IDENTIFIED BY 'nuclear';
CREATE USER 'robert'@'%.example.net' IDENTIFIED BY 'pilot';
CREATE USER 'dallin'@'192.168.1.1' IDENTIFIED BY 'judge';
CREATE USER 'russell'@'192.168.1.%' IDENTIFIED BY 'surgeon';
```

```
CREATE USER 'russell'@'192.168.1.0/255.255.255.0' IDENTIFIED BY 'business';
```

5. Granting permissions

By default, new users do not have the permission to do anything except logging in, which is not very usefull.

```
GRANT <privileges> ON <database> TO <user>;
```

```
GRANT ALL ON *.* TO 'robert'@'%' WITH GRANT OPTION;
GRANT SELECT,INSERT,UPDATE,DELETE ON serv.* TO 'jeffrey'@'localhost';
GRANT SELECT ON edu.staff TO 'david'@'localhost' WITH GRANT OPTION;
The limit will apply to every database that can be accessed by the quentin user and not
GRANT ALL ON logan.* TO 'quentin'@'localhost' WITH MAX_QUERIES_PER_HOUR 100;
```

6. Revoking permissions

to remove privileges

```
REVOKE <privileges> ON <database> FROM <user>;
```

```
REVOKE DELETE,GRANT OPTION ON cust.* FROM 'todd'@'%';
REVOKE ALL,GRANT OPTION FROM 'neil'@'%.example.com';
```

7. Showing permissions

```
SHOW GRANTS FOR <user>;
```

```
SHOW GRANTS FOR 'dieter'@'10.2.200.4';
```

8. Setting and changing passwords

```
SET PASSWORD FOR <user> = PASSWORD('<password>');
```

```
SET PASSWORD FOR 'henry'@'%' = PASSWORD('niftypassword');
```

9. Removing users

```
DROP USER <user>;
```

```
DROP USER 'tom'@'%';
```

10. Using MariaDB

10.1. Connecting to MariaDB

```
mysql [-u <username>] [-p] [-h <host>] [<database>]
```

10.2. Using USE to select a database

```
USE <database>
```

10.3. Using SHOW to list all database

```
SHOW DATABASES;
```

10.4. Creating and deleting databases

When MariaDB is installed, the installar creates four databases
mysql: to keep track of users
test: for experimentation and learning
performance_schema: semi-vritual database, read-only, stores performance
information_schema: semi-virtual database, read-only, stores other statistics

In MariaDB, a schema and a database are quite the same thing

10.5. Creating a database

```
CREATE DATABASE <databasename>;
CREATE DATABASE my_database;
CREATE DATABASE IF NOT EXIST my_database;
```

10.6. Deleting a database

```
DROP DATABASE <databasename>;
DROP DATABASE my_database;
DROP DATABASE IF EXIST my_database;
```

The process by which we refine our table definitions and split our data off into multiple tables is called normalization.
https://mariadb.com/kb/en/recap-the-relational-model

11. Using CREATE TABLE

11.1. Using CREATE TABLE - basic syntax

```
basic CREATE TABLE table_name (<column_definition>);

The <column_definitions> part has the following basic syntax:
<column_name> <data_type>
[NOT NULL | NULL]
[DEFAULT <default_value>]
[AUTO_INCREMENT]
[UNIQUE [KEY] | [PRIMARY] KEY]
[COMMENT '<string>']
The parts in angle brakets(<>) are the bits that we fill in.
The parts in square brakets([]) are optional
The pipe character(|) means "or"
```

11.2. Using CREATE TABLE - datatypes

Different datatypes exist because various types of data are most efficiently stored in different ways.
Common datatypes include numeric(numbers), strings(text), and dates.
numberic: INTEGER, FLOAT
string: CHAR, TEXT, VARCHAR
date: DATE,TIME,DATETIME

11.3. Using CREATE TABLE - an example

```
CREATE TABLE employees (
id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
surname VARCHAR(100),
givenname VARCHAR(100),
pref_name VARCHAR(50),
birthday DATE COMMENT 'approximate birthday OK'
);
```

11.4. Using SHOW to display the command used to create a table

```
SHOW CREATE TABLE employee \G
SHOW CREATE TABLE employee;
```

11.5. Using DESCRIBE to explore the structure of a table

```
DESCRIBE employees;  # COMMENT is not displayed
DESCRIBE employees birthday;  # specific column
```

12. Using ALTER TABLE

12.1. Using ALTER TABLE - basic syntax

```
ALTER TABLE table_name <alter_definition>[, alter_definition] ...;
```

The <alter_definition> part of the command can ADD, MODIFY, and DROP columns from tables.

12.2. Using ALTER TABLE - adding a column

```
ADD <column_name> <column_definition> [FIRST | AFTER <column_name>]
```

The FIRST option puts the new column as the first column of a row.
The after option lets us specify which column the new column appers after.
By default, the column will be added after the current last column.

```
ALTER TABLE employees ADD username VARCHAR(20) AFTER pref_name;
```

12.3. Using ALTER TABLE - modifying a column

```
MODIFY <column_name> <column_definition>
ALTER TABLE employees MODIFY pref_name VARCHAR(25);
```

12.4. Using ALTER TABLE - droping a column

```
DROP <column_name>
ALTER TABLE employees DROP username;
```

12.5. Using DROP TABLE

```
DROP TABLE <table_name>
DROP TABLE mytable;
DROP TABLE IF EXISTS mytable;
```

13. Using MariaDB - Inserting, Updating, and Deleting

13.1. Using INSERT

```
INSERT [INTO] <table_name> [(<column_name>[, <column_name>, ...])] {VALUES | VALUE} ({e
```

Inserting complete rows

```
INSERT INTO employees VALUES
(NULL, "Perry", "Lowell Tom", "Tom", "1988-08-05");
```

For auto-incremented columns, such as the id column in our employees table, we have to put something but we can't put in our own value because MariaDB handles that. So, we use NULL as a placeholder to let MariaDB know that we are not providing a value and then MariaDB provides its own value.

```
INSERT INTO employees VALUES
(NULL, "Pratt","Parley", NULL, NULL),
```

```
(NULL, "Snow","Eliza", NULL, NULL);
```

A better and safer method is to always specify the columns that we are inserting data into, even if it is every column in the table.

Inserting partial rows

```
INSERT INTO employees (surname,givenname) VALUES
("Taylor","John"),
("Woodruff","Wilford"),
("Snow","Lorenzo");
INSERT INTO employees (pref_name,givenname,surname,birthday) VALUES
("George","George Albert","Smith","1970-04-04");
INSERT employees (surname) VALUE ("McKay");
```

VALUE because of the single column

Inserting from another table

```
INSERT [INTO] <table_1> [(<column_name>[, <column_name>,...])]
SELECT <column_name>[, <column_name>,...]
FROM <table_2>;
INSERT INTO employees (surname, givenname, birthday)
SELECT lastname, firstname, bday
FROM names;
```

Inserting from a file
default: tab separated columns file
basic format:

```
LOAD DATA [LOCAL] INFILE '<filename>'
INTO TABLE <tablename>
[(<column_name>[, <column_name>,...]];
```

suppose we have a file named new_employees ,which has three columns which correspond to the birthday , surname , and givenname columns in our employees table:
1971-08-09 Anderson Neil
1985-01-24 Christofferson Todd

```
LOAD DATA INFILE '/tmp/new_employees'
INTO TABLE employees
(birthday, surname, givenname);
```

13.2. Using UPDATE

basic syntax:

```
UPDATE <table_name>
SET column_name1={expression|DEFAULT}
[, column_name2={expression|DEFAULT}] ...
[WHERE <where_conditions>];

UPDATE employees SET
```

```
pref_name = "John", birthday = "1958-11-01"
WHERE surname = "Taylor" AND givenname = "John";

UPDATE employees SET
pref_name = "Will", birthday = "1957-03-01"
WHERE surname="Woodruff";

UPDATE employees SET
birthday = "1964-04-03"
WHERE surname = "Snow";

UPDATE employees SET
birthday = "1975-04-12"
WHERE id = 2;
```

13.3. Using DELETE

```
DELETE FROM <table_name> [WHERE <where_conditions>];
DELETE FROM employees
WHERE givenname="Spencer" AND surname="Kimball";
```

14. Using MariaDB - Retrieving Data

14.1. basic syntax:

```
SELECT <what> FROM <table_name>
[WHERE <where-conditions>]
[ORDER BY <column_name>];
```

14.2. Retrieving everything

```
SELECT * FROM employees;
```

the data will be retrieved and displayed in the order in which it is stored in the table.

14.3. Retrieving selected columns

```
SELECT givenname,surname FROM employees;
```

14.4. Filtering and searching data

Filtering by exact values

```
SELECT * FROM employees
WHERE birthday >= '1970-01-01';
```

14.5. Using logical operators

Using the AND operator

```
SELECT * FROM employees
WHERE surname = 'Snow'
AND givenname LIKE 'Eli%';
```

Using the OR operator

```
SELECT * FROM employees
WHERE givenname = 'Neil'
OR givenname = 'John';
```

14.6. Evaluation order

In SQL, AND operators are evaluated first, followed by the OR operations.

```
ELECT * FROM employees
WHERE
givenname = 'John'
OR givenname = 'Tom'
AND surname = 'Snow';
ELECT * FROM employees
WHERE
(givenname = 'John'
OR givenname = 'Tom')
AND surname = 'Snow';
```

14.7. Using the IN operator

```
SELECT * FROM employees
WHERE
surname = 'Snow'
OR surname = 'Smith'
OR surname = 'Pratt';
SELECT * FROM employees WHERE surname IN ('Snow','Smith','Pratt');
```

14.8. Using the NOT operator

```
SELECT * FROM employees WHERE
surname NOT IN ('Snow','Smith','Pratt');
```

14.9. Searching with LIKE

```
SELECT * FROM employees
WHERE surname LIKE "McK%";
```

%: wildcard

14.10. Sorting data

```
SELECT * FROM employees
WHERE birthday >= '1970-01-01'
ORDER BY surname;
SELECT * FROM employees
WHERE birthday >= '1970-01-01'
ORDER BY surname,givenname , birthday;
```

14.11. Joining data

```
CREATE TABLE phone (
id serial PRIMARY KEY,
emp_id int,
type char(3),
cc int(4),
number bigint,
ext int);
SELECT surname,givenname,type,cc,number,ext
FROM employees JOIN phone
ON employees.id = phone.emp_id;
The first table specified in the FROM clause is called the left table, and the second is
SELECT surname,givenname,type,cc,number,ext
FROM employees LEFT JOIN phone
ON employees.id = phone.emp_id;
```

15. Summarizing data

15.1. The AVG function

The AVG function is used for obtaining the average of the data in a column.
The CURDATE function doesn't take any arguments and when called, it simply returns the current date.
The TIMESTAMPDIFF function takes three arguments-the unit to count by and two dates, and then outputs the difference between the two. The unit is one of several time units, including MINUTE , HOUR , DAY , WEEK , MONTH , QUARTER , and YEAR .

```
SELECT AVG(TIMESTAMPDIFF(YEAR,birthday,CURDATE()))
FROM employees;
```

15.2. The COUNT function

```
SELECT COUNT(*) FROM employees;
SELECT COUNT(pref_name) FROM employees;
```

to count how many do have a set preferred name

15.3. The MIN and MAX functions

the oldest employee

```
SELECT * FROM employees
WHERE birthday = (SELECT MIN(birthday) from employees);
```

the youngest employee

```
SELECT * FROM employees
WHERE birthday = (SELECT MAX(birthday) FROM employees);
```

15.4. The SUM function

```
SELECT SUM(TIMESTAMPDIFF(YEAR,birthday,CURDATE()))
FROM employees;
```

15.5. Using GROUP BY with summarized data

to find out which surnames are the most popular

```
SELECT surname, COUNT(*)
FROM employees
GROUP BY surname;
```

15.6. Using HAVING to filter GROUP BY

```
SELECT surname, COUNT(*)
FROM employees
GROUP BY surname
HAVING COUNT(*) > 1;
```

16. Backing up, importing, and restoring data

16.1. Basic backups with mysqldump

```
mysqldump [-u username] [-p] database_name [table_name]
mysqldump -r root -p test > test.sql
```

16.2. Restoring backups made with mysqldump

```
mysql -u root -p test < test.sql
```

16.3. Making tab-delimited backups with mysqldump

```
mysqldump --tab /tmp/ -u root -p test employees
```

When using this option, mysqldump will create two files. A tablename.sql
file with the SQL commands to recreate the table, and a tablename.txt file with the actual data in tab-delimited format.
The SQL file is owned by whichever user we used to run the mysqldump command. The TXT file, on the other

hand, is owned by the mysql user, so whatever directory we specify needs to have permissions so that both users can write to it.

16.4. Restoring and importing data with mysqlimport

```
mysqlimport [--local] [-u username] [-p] database_name filename
```

The filename attribute must be the name of the table we want to import into.

```
mysqlimport --local -u root -p test /tmp/employees.txt
```

16.5. Making backups of MyISAM tables with mysqlhotcopy

The mysqlhotcopy backup program is actually a Perl script. It can take backups quickly, but only if our tables use the MyISAM or ARCHIVE storage engines.
An easy way to show the storage engines being used by the tables in our database is with the following SELECT statement:

```
SELECT TABLE_NAME,ENGINE
FROM information_schema.tables
WHERE TABLE_SCHEMA="test";
```

We can change test to the name of whichever database we want to check.
mysqlhotcopy db_name [/path/to/new_directory]
The default storage engine for MariaDB is InnoDB , so this script is less useful.

16.6. Making backups of XtraDB and InnoDB tables with xtrabackup

The xtrabackup backup program is made specifically for use with XtraDB and InnoDB tables. It can take quick, full backups of our databases while MariaDB is running.
To take a backup, we do the following:

```
xtrabackup --backup --datadir=/var/lib/mysql/ --target-dir=/path/to/backup/
```

InnoDB and XtraDB tables are stored across several files, and backups made with xtrabackup are the same. This is why while taking a backup with xtrabackup , we specify a directory and not a file name with the –target-dir option.
After making a raw backup, we need to prepare the backup so that it can be restored if necessary.
xtrabackup –prepare –target-dir=/path/to/backup/
The first time that we run xtrabackup with the –prepare option, our backed up data will be cleaned up and put into a logical order. The second time that the –prepare option is used, xtrabackup will create some log files that help speed up restoring our data, if it turns out that we need to do that. Running –prepare a third, fourth, or any more number of times won't do anything, but is safe to do so in case we can't remember if we've run it for the second time.

16.7. Restoring backups made with xtrabackup

The easiest way to restore from a backup made with xtrabackup is to use a utility, such as rsync or the cp command, to copy all the files in the backup directory to our MariaDB data directory. Before doing so, we must

stop MariaDB and then run the rsync or cp command.

```
rsync -avP /path/to/backup/ /var/lib/mysql/
```

After the files are copied back to the MariaDB data directory, and before we start MariaDB, it's a good idea to make sure that the ownership of the files is correct.

```
chown -R mysql:mysql /var/lib/mysql/
```

Author: Mingming Li
Created: 2023-03-15 Wed 21:44
Validate