Chapter 1

Managing MariaDB

1.1 Understanding Relational Databases

Relational Database Management Systems (RDBMS) store data for applications in tables and maintain relationships between them. Since System Administrators deal with applications on a daily basis, it becomes important that we know how to deal with databases as well. Apache fetches data from databases and several logging applications log data to databases as well.

In a database, data is stored in tables. To interact with the tables, we use (typically) some form of querying language, such as **SQL**. to show the entire contents of a table, we use:

```
1 SELECT * FROM videos
```

Here, `videos` is the name of the table we're querying and we want all attributes or fields (indicated by $*$) in the table displayed for every *record* (row) in the table (indicated by the lack of filtering options, i.e., show all records). Each record consists of several columns called *attributes*.

MariaDB is a community branch of MySQL. Before being acquired by Oracle, MySQL was the most used open-source RDBMS, which is why the original developers decided to *fork* the MySQL project's code base before the acquisition and started MariaDB. Thus, the commands are virtually identical between the two. Another open-source database is called PostgreSQL.

1.2 Creating a Base MariaDB Configuration

1.2.1 Installation of Mariadb server

First of all, we need to install a couple of software to be able to use **mariadb**. These are:

```
1 # yum -y install mariadb mariadb-libs mariadb-test
```

The last item is a test database to check the configuration of the mariadb server. Now that it is installed, we must start and enable the server:

```
# systemctl enable mariadb; systemctl start mariadb; systemctl status mariadb
```

The basic configuration for mysql (or in our case, mariadb) in a secure environment is done with the command:

```
# mysql_secure_installation

NOTE: RUNNING ALL PARTS OF THIS SCRIPT IS RECOMMENDED FOR ALL MariaDB
SERVERS IN PRODUCTION USE!  PLEASE READ EACH STEP CAREFULLY!

In order to log into MariaDB to secure it, we'll need the current
password for the root user.  If you've just installed MariaDB, and
you haven't set the root password yet, the password will be blank,
so you should just press enter here.

Enter current password for root (enter for none):
OK, successfully used password, moving on...

Setting the root password ensures that nobody can log into the MariaDB
root user without the proper authorisation.

You already have a root password set, so you can safely answer 'n'.

Change the root password? [Y/n] Y
New password:
Re-enter new password:
Password updated successfully!
Reloading privilege tables..
... Success!
```

The first requirement is that we set a password for the root user **of the database**. This isn't the Linux root user we're concerned about. First it asks us for the present password, but since we haven't set up the root user, we just press enter (An empty string is the default password).

Next we're asked if we want to use anonymous users. Anonymous users are used for testing purposes, and can gain access to the data. To remove anonymous users, we simply type '*Y*' when prompted.

Now, we're asked if remote root login should be disabled. Just like in the case of operating systems, the root user has immense power in administering the DB. So, it's a security practice to disable the remote login of root. If an administrator needs to access the database he/she can gain access to the physical server.

```
By default, a MariaDB installation has an anonymous user, allowing anyone
to log into MariaDB without having to have a user account created for
them.  This is intended only for testing, and to make the installation
go a bit smoother.  You should remove them before moving into a
production environment.

Remove anonymous users? [Y/n] Y
... Success!

Normally, root should only be allowed to connect from 'localhost'.  This
ensures that someone cannot guess at the root password from the network.

Disallow root login remotely? [Y/n] n
... skipping.
```

```
15
16 By default, MariaDB comes with a database named 'test' that anyone can
17 access.  This is also intended only for testing, and should be removed
18 before moving into a production environment.
19
20 Remove test database and access to it? [Y/n] n
21 ... skipping.
22
23 Reloading the privilege tables will ensure that all changes made so far
24 will take effect immediately.
25
26 Reload privilege tables now? [Y/n] Y
27 ... Success!
28
29 Cleaning up...
30
31 All done!  If you've completed all of the above steps, your MariaDB
32 installation should now be secure.
33
34 Thanks for using MariaDB!
```

Next we remove the test database for production environments, or keep it otherwise, if we want and reload the privilege table. After this the database should be operational. The mariadb database has a configuration file located at `/etc/my.cnf`.

1.2.2 Logging in to the Mariadb server

We can login to the server now using the username, hostname and password. While the username and the hostname we provide as arguments to their respective options, the `-p` option makes the system prompt for a password. To do this, we type:

```
1  # mysql -u root -h localhost -p
2  Enter password:
3  Welcome to the MariaDB monitor.  Commands end with ; or \g.
4  Your MariaDB connection id is 12
5  Server version: 5.5.56-MariaDB MariaDB Server
6
7  Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.
8
9  Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
10
11 MariaDB [(none)]>
```

The existing set of databases can be verified with:

```
1  > show databases;
2  +--------------------+
3  | Database           |
4  +--------------------+
5  | information_schema |
6  | mysql              |
7  | performance_schema |
8  | test               |
9  +--------------------+
10 4 rows in set (0.00 sec)
```

1.3 Creating Databases and Tables

1.3.1 Creating and using a new database

If we want to create a new database called *addressbook*, we simply type at the mysql prompt:

```
MariaDB [(none)]> CREATE DATABASE addressbook;
Query OK, 1 row affected (0.00 sec)

MariaDB [(none)]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| addressbook        |
| mysql              |
| performance_schema |
| test               |
+--------------------+
5 rows in set (0.00 sec)
```

Note that every command in MySQL and MariaDB must terminate with a semicolon to indicate where the command ends. We now have a database, but to indicate that all further queries should be directed towards it, we use:

```
MariaDB [(none)]> USE addressbook;
Database changed
MariaDB [addressbook]>
```

Since our database is newly created and contains no data yet, it's empty and has no tables. We can show all tables in a database by using:

```
MariaDB [addressbook]> SHOW TABLES;
Empty set (0.00 sec)
```

To switch to a different database, we simply *use* that database instead, by:

```
MariaDB [addressbook]> USE mysql;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [mysql]>
```

If we were to use the `SHOW TABLES` command on this database, we'd see:

```
MariaDB [mysql]> SHOW TABLES;
+---------------------------+
| Tables_in_mysql           |
+---------------------------+
| columns_priv              |
| db                        |
| event                     |
| func                      |
```

```
| general_log               |
| help_category             |
| help_keyword              |
| help_relation             |
| help_topic                |
| host                      |
| ndb_binlog_index          |
| plugin                    |
| proc                      |
| procs_priv                |
| proxies_priv              |
| servers                   |
| slow_log                  |
| tables_priv               |
| time_zone                 |
| time_zone_leap_second     |
| time_zone_name            |
| time_zone_transition      |
| time_zone_transition_type |
| user                      |
+---------------------------+
24 rows in set (0.01 sec)
```

This is an internal table used by MariaDB itself, and is thus already populated! To see the structure of any table we type:

```
MariaDB [mysql]> DESCRIBE host;
+-----------------------+---------------+------+-----+---------+-------+
| Field                 | Type          | Null | Key | Default | Extra |
+-----------------------+---------------+------+-----+---------+-------+
| Host                  | char(60)      | NO   | PRI |         |       |
| Db                    | char(64)      | NO   | PRI |         |       |
| Select_priv           | enum('N','Y') | NO   |     | N       |       |
| Insert_priv           | enum('N','Y') | NO   |     | N       |       |
| Update_priv           | enum('N','Y') | NO   |     | N       |       |
| Delete_priv           | enum('N','Y') | NO   |     | N       |       |
| Create_priv           | enum('N','Y') | NO   |     | N       |       |
| Drop_priv             | enum('N','Y') | NO   |     | N       |       |
| Grant_priv            | enum('N','Y') | NO   |     | N       |       |
| References_priv       | enum('N','Y') | NO   |     | N       |       |
| Index_priv            | enum('N','Y') | NO   |     | N       |       |
| Alter_priv            | enum('N','Y') | NO   |     | N       |       |
| Create_tmp_table_priv | enum('N','Y') | NO   |     | N       |       |
| Lock_tables_priv      | enum('N','Y') | NO   |     | N       |       |
| Create_view_priv      | enum('N','Y') | NO   |     | N       |       |
| Show_view_priv        | enum('N','Y') | NO   |     | N       |       |
| Create_routine_priv   | enum('N','Y') | NO   |     | N       |       |
| Alter_routine_priv    | enum('N','Y') | NO   |     | N       |       |
| Execute_priv          | enum('N','Y') | NO   |     | N       |       |
| Trigger_priv          | enum('N','Y') | NO   |     | N       |       |
+-----------------------+---------------+------+-----+---------+-------+
20 rows in set (0.00 sec)
```

1.3.2 Basic Commands

The basic MySQL commands allow us to *create*, *select* (view), *update* and *delete* records in the databases. To be able to use these commands, or *queries*, we need to know which attributes are present in the table, which is shown by the `DESCRIBE <tableName>` command.

Insert Query

Once we know the attributes, we can use SQL queries like an *insert query*, which is used to insert a new record:

```
1 INSERT INTO user(Host,User,Password) VALUES ('localhost',lisa,password);
```

The usage of uppercase for SQL keywords is optional, but it makes the commands more readable and makes it easier to distinguish entity (database,tables,etc.) names from the commands at a glance.

The above command *inserts* data into the table `user` for the attributes *Host,User and Password*: specifically the values `localhost,lisa,password`. These values will form a new record in the table `user`.

Delete Query

It is also possible to delete data from tables, but it would need a special `WHERE` clause that filter out only specific records to delete! The syntax is `DELETE FROM tableName WHERE attribute='value';`. Without the `WHERE` clause, all data from the table would be deleted indiscriminately! Thus, to delete the record for the user 'rick', we use:

```
1 DELETE FROM user WHERE user='rick';
```

Update Query

To update the value of an attribute for a record, we use an *update query* which looks like:

```
1 UPDATE user SET password = 'secret' WHERE user = 'lisa';
```

Select Query

To show only parts of, or the contents of the entire table, we use *select queries*. To see all attributes, we use `SELECT * FROM ...`, but to only see specific attributes:

```
1 SELECT host,user FROM user;
```

To see every record in the table where the username is '*Johnson*', we use:

```
1 SELECT * FROM user WHERE user='Johnson';
```

1.3.3 Querying a database

Let us create an example table `videos`. For this, first we create a new database called `videos` and then describe the table:

```
1 MariaDB [(none)]> CREATE DATABASE videos;
2 Query OK, 1 row affected (0.00 sec)
3
```

```
MariaDB [(none)]> USE videos;
Database changed
MariaDB [videos]> CREATE TABLE videos(
-> title VARCHAR(40),
-> actor VARCHAR(40),
-> year INT,
-> registration INT);
Query OK, 0 rows affected (0.04 sec)
```

The Datatype (varchar, int, etc.) need to be specified while creating the attributes in the table, with an optional number specifying the size (if applicable). Note that when the line isn't terminated with a ';', the prompt assumes there's more to the command, and let's us add additional data with a new -> prompt. Now we're ready to insert data into the table:

```
MariaDB [videos]> INSERT INTO videos
-> (registration, title, actor, year) VALUES
-> (1, 'Basic Instinct', 'Sharon Stone', 1992);
```

We can add more data to populate the table:

```
MariaDB [videos]> INSERT INTO videos (registration, title, actor, year) VALUES
↪  (2,'Terminator 1','Arnold Schwarzenegger',1984);
Query OK, 1 row affected (0.00 sec)

MariaDB [videos]> INSERT INTO videos (registration, title, actor, year) VALUES (3,'Pretty
↪  Woman', 'Julia Roberts', 1990);
Query OK, 1 row affected (0.01 sec)
```

We can now view the contents of the table using:

```
> SELECT * FROM videos;
+----------------+-----------------------+------+--------------+
| title          | actor                 | year | registration |
+----------------+-----------------------+------+--------------+
| Basic Instinct | Sharon Stone          | 1992 |            1 |
| Terminator 1   | Arnold Schwarzenegger | 1984 |            2 |
| Pretty Woman   | Julia Roberts         | 1990 |            3 |
+----------------+-----------------------+------+--------------+
3 rows in set (0.00 sec)
```

1.4 Managing Users and Permissions

Just like in the case of operating systems, different users should be assigned different privileges, each carefully controlled to restrict their access to their job roles and nothing beyond.

1.4.1 Adding a DB user

To add a user 'lisa' with the password 'password' on the host 'localhost', we use:

```
CREATE USER lisa@localhost IDENTIFIED BY 'password';
Query OK, 0 rows affected (0.01 sec)
```

This user is created as a record in the `mysql` database, and can be viewed with:

```
> USE mysql;
Database changed
> SELECT host,user,password FROM user WHERE user='lisa';
+-----------+------+-------------------------------------------+
| host      | user | password                                  |
+-----------+------+-------------------------------------------+
| localhost | lisa | *2470C0C06DEE42FD1618BB99005ADCA2EC9D1E19 |
+-----------+------+-------------------------------------------+
1 row in set (0.00 sec)
```

Instead of adding a user this way, it's also acceptable to use PAM (Pluggable Authentication Modules) as used during user authentication in Linux.

1.4.2 Dropping (deleting) a user

We can delete an existing user by:

```
> DROP USER lisa@localhost;
Query OK, 0 rows affected (0.00 sec)

> SELECT host,user,password FROM user WHERE user='lisa';
Empty set (0.00 sec)
```

1.4.3 Privileges

By default, the new users have no privileges (permissions) at all. Thus, for the user accounts to be useful, certain appropriate privileges must be granted to them. This can be done either on a per-table basis, or globally for a database! For example:

```
> GRANT UPDATE,DELETE,INSERT,SELECT ON addressbook.names TO lisa@localhost;
Query OK, 0 rows affected (0.00 sec)
```

To now give the user *lisa* SELECT privileges on the entire *addressbook* database, we use:

```
> GRANT SELECT ON addressbook.* TO lisa@localhost;
Query OK, 0 rows affected (0.00 sec)
```

To give the user permissions to edit the schema of the database, but not have access to the records within, we give him/her the permission:

```
GRANT CREATE,ALTER,DROP ON addressbook.* TO lisa@localhost;
```

Finally, to make a user Database-admin, i.e., give them all privileges, we use the command:

```
GRANT ALL PRIVILEGES ON *.* TO user@host;
```

After modifying privileges with the above command, to make them usable we need to apply them with the command:

```
> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.01 sec)
```

To see all the privileges that a user has, we use the command:

```
> SHOW GRANTS FOR lisa@localhost;
+-----------------------------------------------------------------------------------------+
| Grants for lisa@localhost                                                               |
+-----------------------------------------------------------------------------------------+
| GRANT USAGE ON *.* TO 'lisa'@'localhost' IDENTIFIED BY PASSWORD '*2470C0C06D..9'    |
| GRANT SELECT ON `addressbook`.* TO 'lisa'@'localhost'                                   |
| GRANT SELECT, INSERT, UPDATE, DELETE ON `addressbook`.`names` TO 'lisa'@'localhost'|
+-----------------------------------------------------------------------------------------+
3 rows in set (0.00 sec)
```

1.4.4 Advanced user options

Let us first create a user 'julia' who isn't bound to any certain host:

```
CREATE USER julia@'%' IDENTIFIED BY 'secret';
Query OK, 0 rows affected (0.01 sec)
```

Now we grant some privileges to the user with:

```
> GRANT SELECT,INSERT,UPDATE,DELETE ON videos.* TO julia@'%';
Query OK, 0 rows affected (0.01 sec)
```

Now we apply the privileges and check them:

```
> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)

MariaDB [videos]> SHOW GRANTS FOR julia@'%';
+-------------------------------------------------------------------------+
| Grants for julia@%                                                      |
+-------------------------------------------------------------------------+
| GRANT USAGE ON *.* TO 'julia'@'%' IDENTIFIED BY PASSWORD '*14E655..E7' |
| GRANT SELECT, INSERT, UPDATE, DELETE ON `videos`.* TO 'julia'@'%'      |
+-------------------------------------------------------------------------+
2 rows in set (0.00 sec)
```

To check what is currently in the database, we use:

```
> DESCRIBE videos;
+--------------+-------------+------+-----+---------+-------+
| Field        | Type        | Null | Key | Default | Extra |
+--------------+-------------+------+-----+---------+-------+
| title        | varchar(40) | YES  |     | NULL    |       |
| actor        | varchar(40) | YES  |     | NULL    |       |
| year         | int(11)     | YES  |     | NULL    |       |
| registration | int(11)     | YES  |     | NULL    |       |
+--------------+-------------+------+-----+---------+-------+
4 rows in set (0.02 sec)
```

Let us now insert another value into the database:

```
> INSERT INTO videos(registration,title,year,actor) VALUES (4, 'The Last Stand', 2013,
↪   'Arnold Schwarzennegger');
Query OK, 1 row affected (0.01 sec)
```

Now, to only see the results where the actor is *Arnold*, we use:

```
> SELECT * FROM videos WHERE actor='Arnold Schwarzenegger';
+----------------+----------------------+------+--------------+
| title          | actor                | year | registration |
+----------------+----------------------+------+--------------+
| Terminator 1   | Arnold Schwarzenegger | 1984 |            2 |
| The Last Stand | Arnold Schwarzenegger | 2013 |            4 |
+----------------+----------------------+------+--------------+
2 rows in set (0.00 sec)
```

To quit the MariaDB management interface, we can now use `quit`.

1.5 Backing up the Database

There can be two types of database backups: *logical* and *physical*. In the case of physical backups, the database must be stopped. The resultant backup is portable to other machine with a similar hardware and software set up. Depending on the backup options, this backup can contain logs and configurations as well.

Contrastingly, a logical backup is only a backup of the data contained within the tables of the database, obtained by querying the database itself. The main advantage is that such a backup can be created while the database is online, thus avoiding service disruption. However, it is relatively slow. Since the output is a bunch of SQL queries, the output is portable to other database vendors as well!

1.5.1 Creating a logical backup

To create a logical backup, all we need to do is use the `mysqldump` command from the shell:

```
# mysqldump -u root -p videos > ~/videos-db.dump
Enter password:
# cat ~/videos-db.dump
-- MySQL dump 10.14  Distrib 5.5.56-MariaDB, for Linux (x86_64)
--
-- Host: localhost    Database: videos
-- ------------------------------------------------------
-- Server version       5.5.56-MariaDB
...
CREATE TABLE `videos` (
`title` varchar(40) DEFAULT NULL,
`actor` varchar(40) DEFAULT NULL,
`year` int(11) DEFAULT NULL,
`registration` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
...
```

```
17 INSERT INTO `videos` VALUES ('Basic Instinct','Sharon Stone',1992,1),('Terminator
    ↪  1','Arnold Schwarzenegger',1984,2),('Pretty Woman','Julia Roberts',1990,3),('The
    ↪  Last Stand','Arnold Schwarzenegger',2013,4);
18 /*!40000 ALTER TABLE `videos` ENABLE KEYS */;
19 ...
20 -- Dump completed on 2018-03-28 19:09:32
```

To generate a backup (dump) of all databases, we'd use:

```
1 # mysqldump -u root -p --all-databases > all-db.dump
```

1.5.2 Physical Backups

The first requirement for physical backups is LVM. First we need to know which directory to backup, which can be obtained using:

```
1 # mysqladmin -u root -p variables | grep datadir
2 Enter password:
3 | datadir                                       | /var/lib/mysql/
```

Now, we need to check the available disk space on the VG with `vgs <vgName>` to ensure enough is available, since we're going to create a snapshot. Then, we **Freeze** the database (stopping it completely is also an option), using the commands:

```
1 > FLUSH TABLES WITH READ LOCK;
2 Query OK, 0 rows affected (0.00 sec)
```

The above is done to ensure absolutely no changes are made to the database during the physical backup process. Once frozen, we can create a LVM snapshot from a different terminal, using a command like:

```
1 # lvcreate -L 25 -s -n lvMariaDB-Snapshot /dev/vgData/lvMariaDB
2 Using default stripesize 64.00 KiB.
3 Logical volume "lvMariaDB-Snapshot" created.
```

The `-s` option tells lvcreate that the new LV will be a snapshot of some existing LV, which is `/dev/vgData/lvMariaDB` mounted on `/var/lib/mysql`. Once this step is done, the tables can now be unfrozen by:

```
1 UNLOCK TABLES;
```

Now, the snapshot `/dev/vgData/lvMariaDB-Snapshot` can be mounted and used to create a backup. Snapshots act like a *picture* of the state of that LV in that moment. To mount the snapshot:

```
1 # mkdir /mnt/snapshot
2 # mount /dev/vgData/lvMariaDB-Snapshot /mnt/snapshot
```

To create the actual backup, we create a `tar` archive, using:

```
1 # tar -cvf /root/mariadb.tar /mnt/snapshot
```

We can now unmount the snapshot and then delete the LV snapshot:

```
# umount /mnt/snapshot
# lvremove /dev/vgData/lvMariaDB-Snapshot
```

The important point to remember here is the fact that it's **not possible** to create a physical database backup without having the datadir set up on an LVM.

1.5.3 Restoring a Database from a Backup

Lgoical Backup

We just need to run the commands in the backup file on a sql terminal by:

```
# mysql -u root -p videos < ~/videos-db.dump
```

Physical Backup

In case of physical backups, we first need to stop the database, and then delete the existing contents of the `/var/lib/mysql` directory, and replace it with the contents of the backup:

```
# systemctl stop mariadb
# rm -rf /var/lib/mysql/*
# tar xvzf /root/mariadb.tar -C /
```

This extracts the files to `/var/lib/mysql` directory, thus restoring the database from the backups.