Introduction To MySQL

Starting MySQL Command-Line Interface

MySQL is preinstalled on our provided VMWare image and automatically starts when you boot the virtual machine. Once you log in to the VM, you can start the MySQL command-line interface by typing `mysql`:

```
cs143@cs143:~$ mysql
```

Then you should receive the following prompt

```
mysql>
```

and be inside the MySQL command-line interface. All commands in this tutorial should be issued inside the MySQL command-line unless noted otherwise.

Choosing a Databases in MySQL

MySQL allows users to create *multiple* databases, so that a single MySQL server can host databases for many independent applications. Before you start issuing SQL commands to `mysql`, you first have to select the database that you will be using. In order to see what databases currently exist, run

```
    SHOW DATABASES;
```

You will see an output like

```
+--------------------+
| Database           |
+--------------------+
| information_schema |
| CS143              |
| TEST               |
+--------------------+
```

`information_schema` is a database that MySQL creates automatically and uses to maintain some internal statistics on datbases and tables. The other two databases, `CS143` and `TEST`, are what we created for the project (note database names are *case-sensitive* in MySQL). This two database setup mimics common development environments in the real world. The `CS143` database is your "production" database, meant for use in the final versions of your code. The `TEST` database is for any experimentation and for use during development and debugging. Select the `TEST` database for the rest of this tutorial by issuing the command

```
    USE TEST;
```

It is also possible to specify a database as a command line parameter to the `mysql` command:

```
cs143@cs143:~$ mysql TEST
```

Creating a Table

Once you select a database, you can execute any SQL command. For example, you can create a table using the `CREATE TABLE` command:

```
CREATE TABLE <tableName> (
    <list of attributes and their types>
);
```

Note that all reserved keywords (like `CREATE` and `TABLE`) are *case-insensitive* and identifiers (like table names and attribute names) are *case-sensitive* in MySQL by default. That is, a table named `STUDENT` is different from the `student` table.

You may enter a command on one line or on several lines. If your command runs over several lines, you will be prompted with " `->` " until you type the semicolon that ends any command. An example table-creation command is:

```
CREATE TABLE tbl(a int, b char(20));
```

This command creates a table named `tbl` with two attributes. The first, named `a`, is an integer, and the second, named `b`, is a character string of length (up to) 20.

When you create a table, you can declare a (set of) attribute(s) to be the primary key like:

```
CREATE TABLE <tableName> (..., a <type> PRIMARY KEY, b, ...);
```

or

```
CREATE TABLE <tableName> (<attrs and their types>, PRIMARY KEY(a,b,c));
```

Inserting and Retrieving Tuples

Having created a table, we can insert tuples into it. The simplest way to insert is with the `INSERT` command:

```
INSERT INTO <tableName>
    VALUES( <list of values for attributes, in order> );
```

For instance, we can insert the tuple `(10, 'foobar')` into relation `tbl` by

```
INSERT INTO tbl VALUES(10, 'foobar');
```

Once tuples are inserted, we can see the tuples in a relation with the command:

```
SELECT *
FROM <tableName>;
```

For instance, after the above create and insert statements, the command

```
SELECT * FROM tbl;
```

produces the result

```
+------+--------+
| a    | b      |
+------+--------+
|   10 | foobar |
+------+--------+
```

Creating Index

Having created a table, we can create an index on some attributes of the table. The command for creating an index is:

```
CREATE INDEX <indexName> ON <tableName>(<list of attributes>);
```

For instance, we can create an index on `b` attribute of table `tbl` by

```
CREATE INDEX IdxOnAttrB ON tbl(b);
```

Later, if you want to drop an index, you use the following command

```
DROP INDEX <indexName> ON <tableName>;
```

like

```
DROP INDEX IdxOnAttrB ON tbl;
```

Note that in MySQL, an index is automatically created on primary keys and unique attributes.

Bulk Loading Data

Instead of inserting tuples one at a time, it is possible to create a file that contains all tuples that you want to load in batch. The command for bulking loading tuples from a file is the following:

```
LOAD DATA LOCAL INFILE <dataFile> INTO TABLE <tableName>;
```

where `<dataFile>` is the name of the file that contains the tuples. Each line in the data file corresponds to one tuple and columns are separated by a tab character (\t). You can specify a NULL value in the data file using `\N`.

For example, the following data file

```
1	first
2	second
3	\N
```

will insert three tuples, (1, 'first'), (2, 'second'), and (3, NULL) to a table. If you want to use, say, commas to separate columns, not tabs, add `FIELDS TERMINATED BY ','` to the `LOAD` command as follows:

```
LOAD DATA LOCAL INFILE <dataFile> INTO TABLE <tableName>
FIELDS TERMINATED BY ',';
```

If some columns in the data file is enclosed with, say, double quotes, you need to add `OPTIONALLY ENCLOED BY '"'` as well:

```
LOAD DATA LOCAL INFILE <dataFile> INTO TABLE <tableName>
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"';
```

Notes on CR/LF issue: If your host OS is Windows, you need to pay particular attention to how each line of a text file ends. By convention, Windows uses a pair of CR (carriage return) and LF (line feed) characters to terminate lines. On the other hand, Unix (including Linux and Mac OS X) use only a LF character. Therefore, problems arise when you are feeding a text file generated from a Windows program to a program running in Unix (such as mysql in Ubuntu). Since the end of the line of the input file is different from what the tools expect, you may encounter unexpected

behavior from these tools. If you encounter this problem, you may want to run `dos2unix` command from Ubuntu on your Windows-generated text file. This command converts CR and LF at the end of each line in the input file to just LF. Type `dos2unix --help` to learn how to use this command.

Getting Rid of Your Tables

To remove a table from your database, execute

```
DROP TABLE <tableName>;
```

We suggest you execute

```
DROP TABLE tbl;
```

after trying out the sequence of commands in this tutorial to avoid leaving a lot of garbage tables around.

Getting Information About Your TABLES

You can get the set of all tables within the current database by the following command:

```
SHOW TABLES;
```

Once you know the list of tables, it is also possible to learn more about the table by issuing the command:

```
DESCRIBE <tableName>;
```

Executing SQL From a File

Instead of typing and running SQL commands at a terminal, it is often more convenient to type the SQL command(s) into a file and cause the file to be executed.

To run the commands in `foo.sql` (in the current working directory), type:

```
SOURCE foo.sql;
```

in `mysql`. Files like `foo.sql` that have SQL commands to be executed are often referred to as a (batch) script file. You can also execute the script file directly from the Unix shell by redirecting the input to `mysql` like the following:

```
cs143@cs143:~$ mysql TEST < foo.sql
```

Again, pay attention to the CR/LF issue if your host OS is windows and if you create your SQL batch script file from Windows. Run `dos2unix` on the file if necessary.

Recording Your MySQL Session In a File

`mysql` provides the command `TEE` to save the queries that you executed and their results to a file. At the `mysql>` prompt, you say:

```
TEE foo.txt;
```

and a file called `foo.txt` will appear in your current directory and will record all user input and system output, until you exit `mysql` or type:

```
NOTEE;
```

Note that if the file `foo.txt` existed previously, new output will be appended to the file.

Quitting `mysql`

To leave `mysql`, type

```
QUIT;
```

MySQL Users and Privileges

By default, when you run `mysql`, you connect to MySQL as a user with the same name of your Unix account: `cs143`. It is also possible to connect to MySQL as a different user by specifying the `-u` option:

```
cs143@cs143:~$ mysql <database> -u <username> -p
```

Here, `<username>` should be replaced with the username that you want to use, and the option `-p` asks `mysql` to prompt for the password of the user. So far in this tutorial, we haven't had to use the `-p` option because the default user `cs143` has empty password.

In our VM, we created another user `root` with password `password`. For example, if you run

```
cs143@cs143:~$ mysql TEST -u root -p
```

`mysql` will ask for a password, for which you have to type "`password`" (without quotes). After successful login, you are now connected to the `TEST` database as the user `root`.

The user `root` is the "superuser" or the "database administrator" of our MySQL installation and has full access to all databases and tables, including the ability to create new users, change user privileges, and so on. Because the root user has unrestricted access, for security purposes, it is best to connect to MySQL as `root` only when you need to perform one of these administrative tasks. For your project work, the default user account `cs143` (no password) will be sufficeint, which has full access to the `CS143` and `TEST` databases but nothing else.

If you are logged in to MySQL as the `root` user, you can create a new MySQL user with the `CREATE USER` command:

```
CREATE USER <user> IDENTIFIED BY '<password>';
```

where `<user>` is the name of the new user and `<password>` is its password. To create a user with an empty password, you can simply type `CREATE USER <user>;` skipping the `IDENTIFIED BY` part. All user data is stored in the `user` table in the `mysql` database, issuing the following query as `root` will give you the list of current users:

```
SELECT * FROM mysql.user;
```

Note that prefixing the table name `user` with the database name `mysql` with a dot allows you to access a table in a database not in currently in use. (`mysql` is the database where MySQL maintains administrative records.) Once a new user is created, you will have to grant appropriate privileges to the user with the `GRANT` command. For example, the command

```
GRANT ALL ON TEST.* to cs143@localhost;
```

will give all privileges for the `TEST` database to the user `cs143` (on `localhost`). To see the privileges that you have, run:

```
SHOW GRANTS;
```

which will produce an output like:

```
+-----------------------------------------------------------+
| Grants for cs143@localhost                                |
+-----------------------------------------------------------+
| GRANT USAGE ON *.* TO 'cs143'@'localhost'                 |
| GRANT ALL PRIVILEGES ON `CS143`.* TO 'cs143'@'localhost' |
| GRANT ALL PRIVILEGES ON `TEST`.* TO 'cs143'@'localhost'  |
+-----------------------------------------------------------+
```

As the `root` user, you can also create and drop databases using `CREATE DATABASE` and `DROP DATABASE` command like the following:

```
CREATE DATABASE TESTDB;
DROP DATABASE TESTDB;
```

As a final note, keep in mind that MySQL maintains its own username and password pairs independently of the underlying OS. Thus, it is possible to create a MySQL user that does not exist in the underlying OS and vice versa. This means that your default MySQL user `cs143` is not related to your Ubuntu account `cs143` except the shared name. In fact, as you may have noticed, `cs143` in MySQL has a different password (empty) from `cs143` of the Ubuntu account (`"password"`).

Help Facilities

`mysql` provides internal help facilities for MySQL commands. To see a list of commands for which help is available, type `help` or `help contents` in `mysql`. To look up help for a particular topic (listed in the contents), type `help` followed by the topic.

This document was adapted from Getting Started With Oracle by Junghoo "John" Cho for CS143 at UCLA.