Week 3 Exercise

Part 1 - Docker & MySQL

Part 1a - Initial Run

In this first part, we will try running MySQL in a Docker container and logging into it. Open a terminal and change directory (`cd`) to the week 03 exercise folder.

Now enter this command:

```
docker -v
```

You should get a response similar to this (though the numbers may vary):

```
Docker version 18.03.0-ce, build 0520e24
```

If that does not work, then Docker may not be running. Go to your applications in Finder or Explorer and start Docker. After it is running, try the above command again.

Assuming that Docker is installed and running, now try running a MySQL container with the following command. It will have to download the MySQL container image the first time you do this, which could take a few minutes depending on bandwidth.

```
docker run -d --name wk03ex -e MYSQL_ROOT_PASSWORD=crud mysql
:5
```

After it downloads the image, it will run the container, which has an instance of MySQL running inside it.

To confirm that it is running do this:

```
docker ps
```

You should see that one MySQL container is running.

OK, now let’s try logging in to that container and then logging into the MySQL database.

```
docker exec -it wk03ex bash
```

That will log you into the container and give you a shell prompt as the root administrator for the virtual server.

Now log in to MySQL:

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

It will prompt you to enter a password. The password is `crud`.

At the `mysql` prompt enter

```
show databases;
```

You should get something like this

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

These are the system databases it uses to keep track of the databases that you create.

Let’s create out own database.

```
mysql> create database wk03;
Query OK, 1 row affected (0.00 sec)

mysql> use wk03;
Database changed
```

OK, now let’s create a table and enter some data. Paste in these SQL statements

```
CREATE TABLE account(
    id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(64) NOT NULL,
    balance DOUBLE(14,2) NOT NULL DEFAULT 0.00
    );

INSERT INTO account(name) VALUES
    ('Aida Jones'),
    ('Belissa Mayo');

UPDATE account SET balance=100.00 WHERE id=1;

SELECT * FROM account;
```

You should get:

```
+----+--------------+---------+
| id | name         | balance |
+----+--------------+---------+
|  1 | Aida Jones   |  100.00 |
|  2 | Belissa Mayo |    0.00 |
+----+--------------+---------+
2 rows in set (0.00 sec)
```

Part 1b - Shutting Down

OK, now log out of MySQL by typing `exit`

Now log out of the container by typing `exit` again

You should now be back in your local week 03 exercise folder.

Shut down the container with this command

```
docker stop wk03ex
```

Now take a look at the docker container list again

```
docker ps
```

It should show nothing. Does this mean your database was wiped out? No. The container still exists, it’s just not running. Try adding the `-a` flag to get all the containers.

```
docker ps -a
```

You should now see that your `wk03ex` container is still there. Your data is still in it too. It will stay there unless you delete the container (with `docker rm wk03`, but don’t do that!).

A real setup with docker would use “volumes” to store the data on the host machine (i.e. your computer’s file system) instead of inside the container, but we don’t need to do that for our class assignments.

Part 1c - Logging back in

OK, now let’s try turning it all back on again

```
docker run -d --name wk03ex -e MYSQL_ROOT_PASSWORD=crud mysql:5
docker exec -it wk03ex bash
mysql -u root -p
```

Enter password and continue

```
use wk03;
show tables;
```

Your `account` table should still be there, and ready to use. So now let’s move on to actual testing.

Part 1d - Make a transaction

OK, let’s write a SQL statement that uses a transaction. Let’s transfer money from one account to another and wrap it in a transaction so that we ensure all operations are executed. Aida wants to give Belissa $25.

Open the file `wk03ex.sql` and fill out the section for part 2a: create a transactino that transfers $25 from Aida to Belissa.

```
START TRANSACTION;
    UPDATE account
    SET balance=balance-25.00
    WHERE id=1;

    UPDATE account
    SET balance=balance+25.00
    WHERE id=2;
COMMIT;
```

OK, now check the balances:

```
SELECT * FROM account;
```

You should get

```
+----+--------------+---------+
| id | name         | balance |
+----+--------------+---------+
|  1 | Aida Jones   |   75.00 |
|  2 | Belissa Mayo |   25.00 |
+----+--------------+---------+
2 rows in set (0.00 sec)
```

Great! There’s not much more we can do with transactions without using some programming to show off concurrency, so we’ll leave transactions for now. Let’s move on to triggers.

Part 2 - Triggers

Part 2a - Create tables

Let’s create a new set of tables that have data about objects which are part of collections.

Create a table `collection` that has an `id` primary key and a `name` that is required and unique. It should also have a field for `total_value` that is a double (14 digits max, 2 decimal places) and has a default value of 0.00.

Now create a table `object` that also has an `id`, `name`, and `value`.

Part 2b - Insert some data

Insert some data into the tables. I have created some data for you.

```
INSERT INTO collection (name)
VALUES ('photographs'),
       ('paintings'),
       ('sculpture');

INSERT INTO object (name, value, collectionid)
```

```
VALUES ('Spring', 64000000.00, 2),
       ('Irises', 54000000.00, 2),
       ('some selfie 1', 1.23, 1),
       ('some selfie 2', 2.23, 1),
       ('The Thinker', 16000000.00, 3),
       ('The Thinker (replica)', 199.99, 3);

UPDATE collection a
INNER JOIN (
    SELECT collectionid, sum(value) total_value
    FROM object
    GROUP BY collectionid) b ON a.id=b.collectionid
SET a.total_value=b.total_value;
```

Part 2c - Create some triggers

Now create a trigger that updates a collections `total_value` when a new object is created.

Now insert a value and see what happens.

```
INSERT INTO object (name, value, collectionid)
VALUES ('Obama selfie 1', 2000.00, 1);

SELECT * FROM collection;
```

Expected results:

```
+----+-------------+--------------+
| id | name        | total_value  |
+----+-------------+--------------+
|  1 | photographs |         3.46 |
|  2 | paintings   | 118000000.00 |
|  3 | sculpture   |  16004199.99 |
+----+-------------+--------------+
3 rows in set (0.00 sec)
```

Looks like we inserted the selfie into the wrong collection. We should update it.

But wait, our trigger only works on insert. If we update the object table, the collection values will still be incorrect.

Create a trigger to handle updates to object.

Now test it out

```
UPDATE object SET collectionid=1 WHERE name="Obama selfie 1";

SELECT * FROM collection;
```

Expected results:

```
+----+-------------+--------------+
| id | name        | total_value  |
+----+-------------+--------------+
|  1 | photographs |      2003.46 |
|  2 | paintings   | 118000000.00 |
|  3 | sculpture   |  16002199.99 |
+----+-------------+--------------+
3 rows in set (0.00 sec)
```

While you’re at it, create a trigger to handle delets from object.

Now test it out

```
DELETE FROM object WHERE name="Obama selfie 1";

SELECT * FROM collection;
```

Expected results:

```
+----+-------------+--------------+
| id | name        | total_value  |
+----+-------------+--------------+
|  1 | photographs |         3.46 |
|  2 | paintings   | 118000000.00 |
|  3 | sculpture   |  16002199.99 |
+----+-------------+--------------+
3 rows in set (0.00 sec)
```