COMP1004 Lab Week 3: SQL I– Part B

INTRODUCTION

This exercise will cover the process of adding, updating and removing records from your database tables. The database we will be working on is the one set up in Exercises 1 and 2 of Part A, so be sure to have done that first. In this exercise we will be concentrating entirely on Movie and Actor information, we will not be handling Scene information.

ADDING INFORMATION TO THE DATABASE

Suppose that we want to add the following Movie information into the database:

Actor	**Title**	**Price**	**Genre**
Arethan	Die hard with a Vengeance	12.24	Action
Barry	Black Snake Moan	9.99	Adventure
Bullet	Snake on a Plane	9.99	Sci-fi
Daniel	Freeway of Love	8.05	Drama
James	I knew you were waiting for me	12.24	Comedy
Jonny	Black Panther	9.99	Drama
Laura	The Jungle Book	9.99	Crime
Ryan	Infinity War	7.99	Horror
Sebastian	Coming to Europe	8.54	Adventure

Each Movie is going to require two entries – one in the Movie table, and one in Actor. We will consider 2 ways of entering the data into the tables: **The first way** is by using the INSERT command in a **straightforward** manner for both tables.

The second way is a bit **smarter**: we don't want to put each artist in more than once, and we need to know the actor's ID before we can insert it in the Movie table. This means that there are 4 stages to add a Movie:

1. Check to see if there is an entry for the actor involved. To do this we use a SELECT query to try to find their ID. For example for the first Movie:

```
SELECT actID FROM Actor WHERE actName = 'Barry';
```

2. If there isn't, make a new entry with a new actID. Since our database initially is empty, the SELECT above will return no results. This tells us that we must add Barry into the artist table. We add them by name. Remember that we also need to specify a unique IDs for each artist. One approach would be to make them up, but this becomes unwieldy for anything other than a tiny database. So, as we have seen, we can make use of the MySQL AUTO_INCREMENT functionality. In this way we do not need specify an actID as one will be generated for us. Overall, to insert Barry into Actor we type:

```
INSERT INTO Actor (actName) VALUES ('Barry');
```

3. If there is, retrieve their actID; To do this, we must re-run our query:

```
SELECT actID FROM Actor WHERE actName = 'Barry';
```

4. Make an entry in the Movie table with the actID from step 3. For example, now that we know the ID for Barry, we can add their first Movie:

```
INSERT INTO Movie (actID, mvTitle, mvPrice, mvGenre)
VALUES (1, 'Black Snake Moan', 12.24, 'Action');
```

Exercise: Add all of the Movies from the table above into the database. Try to do this, using both ways ("straightforward" and "smart") described above. Remember to add the actor names first in the Actor table **[Output 1].**

QUERYING A TABLE

You can see what is in your database at any time using SQL's SELECT statement. This will be looked at in much more detail in the next Exercise. For now, you can read the entire contents of your table using the following command:

```
SELECT * FROM Actor;
```

which will return the following table:

actID	actName
6	Arethan
1	Barry
2	Bullet
4	Daniel
5	James
3	Jonny
8	Laura
7	Ryan

Exercise: Query both the Actor and Movie tables to ensure you have added all information correctly.

UPDATING INFORMATION IN A TABLE

To update information in your table, you can use SQL's UPDATE command. You should almost always specify which rows to change using a WHERE clause, to avoid changing all rows. For example, to change the actor called "Barry" to "Barry Nelson" you would use:

```
UPDATE Actor SET actName = 'Barry Nelson' WHERE actName =
'Barry';
```

It is very important to use a correct `WHERE` statement, to avoid renaming all your actors. You may wish to run a `SELECT` statement using the same `WHERE` clause to check the rows the `UPDATE` will operate on.

You can also use the old value to calculate a new value using a function. For example, to increase the price of all Movie by 1.00 pound, we would use:

```
UPDATE Movie SET mvPrice = mvPrice + 1.00;
```

Exercise: Change the name of the artist "James" to "James Bond". You might want to check your change with a `SELECT` query first **[Output2].**

Exercise: If you haven't already done it, increase the price of all Movie by 1.00, and then write a command to undo this change. Next, reduce the price of all Movies that cost more than 10.00 pounds by 50 pence **[Output 3].**

After these exercises, your tables should look like these (your ID values may be different, don't worry about that):

```
SELECT * FROM Actor;
```

```
SELECT * FROM Movie;
```

actId	actName
6	Arethan
1	Barry Nelson
2	Bullet
4	Daniel
5	James Bond
3	Jonny
8	Laura
7	Ryan

mvID	acID	mvTitle	mvPrice	mvYear	mvGenre	mvNumScenes
1	1	Die Hard With a Vengeance	11.74	NULL	Action	NULL
2	2	Black Snake Moan	9.99	NULL	Adventure	NULL
3	3	Snake on a Plane	9.99	NULL	Sci-fi	NULL
4	4	Freeway of Love	8.05	NULL	Drama	NULL
5	5	I knew you were waiting for me	11.74	NULL	Comedy	NULL
6	2	Black Panther	9.99	NULL	Drama	NULL
7	6	The Jungle Book	9.99	NULL	Crime	NULL
8	7	Infinity War	7.99	NULL	Horror	NULL

DELETING ROWS FROM TABLES

SQL's `DELETE` statement removes rows from tables. Like `UPDATE`, you will almost always want

to use a `WHERE` clause to specify which rows to delete. If you don't use a `WHERE` clause, the command will delete all rows.

Sometimes foreign key constraints will mean rows cannot be removed. This occurs where a foreign key in another table references a row you are trying to remove, and the foreign key has been specified as `ON DELETE RESTRICT` (the default). For example, if we were to delete the actor Bullet, the actID reference in Movie for their film would be invalid. MySQL (using the InnoDB engine) will not allow this:

```
DELETE FROM Actor WHERE actName='Bullet';
```

You will get the following error:

```
ERROR Code: 1451. Cannot delete or update a parent row: a
foreign key constraint fails (`usr`.`Movie`, CONSTRAINT
`fk_cd_act` FOREIGN KEY (`actID`) REFERENCES `Actor` (`actID`))
```

Exercise: Try removing the actor "Bullet" from your database. MySQL should prevent you from doing so. If you can, there is likely to be something wrong with your foreign key constraints, or your choice of storage engine. To delete the actor completely, you will need to first delete all of their Movies.