G51DBI 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 CD and Artist information, we will not be handling Track information.

ADDING INFORMATION TO THE DATABASE

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

Artist	**Title**	**Price**	**Genre**
Muse	Black Holes and Revelations	9.99	Rock
Muse	The Resistance	11.99	Rock
Mr. Scruff	Ninja Tuna	9.99	Electronica
Deadmau5	For Lack of a Better Name	9.99	Electro House
Mark Ronson	Record Collection	11.99	Alternative Rock
Mark Ronson	Version	12.99	Pop
Animal Collective	Merriweather Post Pavilion	12.99	Electronica

Each CD is going to require two entries – one in the CD table, and one in Artist. 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 artist's ID before we can insert it in the CD table. This means that there are 4 stages to add a CD:

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

   ```
   SELECT artID FROM Artist WHERE artName = 'Muse';
   ```

2. If there isn't, make a new entry with a new artID. Since our database initially is empty, the SELECT above will return no results. This tells us that we must add Muse 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 artID as one will be generated for us. Overall, to insert Muse into Artist we type:

```
INSERT INTO Artist (artName) VALUES ('Muse');
```

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

   ```
   SELECT artID FROM Artist WHERE artName = 'Muse';
   ```

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

   ```
   INSERT INTO CD (artID, cdTitle, cdPrice, cdGenre) VALUES (1,
   'Black Holes and Revelations', 9.99, 'Rock');
   ```

Exercise: Add all of the CDs from the table above into the database. Try to do this, using both ways ("straightforward" and "smart") described above. Remember to add the artist names first in the Artist 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 Artist;
```

which will return the following table:

artID	artName
5	Animal Collective
3	Deadmau5
4	Mark Ronson
2	Mr. Scruff
1	Muse

Exercise: Query both the Artist and CD 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 artist called "Mr. Scruff" to "Mrs. Scruff" you would use:

```
UPDATE Artist SET artName = 'Mrs. Scruff' WHERE artName = 'Mr.
Scruff';
```

It is very important to use a correct `WHERE` statement, to avoid renaming all your artists. 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 CDs by 1.00 pound, we would use:

```
UPDATE CD SET cdPrice = cdPrice + 1.00;
```

Exercise: Change the name of the artist “Muse” to “Amuse”. 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 CDs by 1.00, and then write a command to undo this change. Next, reduce the price of all CDs that cost more than 10.00 pounds by 50 pence [Output 3].

Exercise: Change the artist on the CD “Record Collection” to “Mark Ronson & The Business Intl” while leaving the artist on the CD “Version” unchanged (this will involve creating a new artist) [Output 4].

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

```
SELECT * FROM Artist;

SELECT * FROM CD;
```

artID	artName
1	Amuse
5	Animal Collective
3	Deadmau5
4	Mark Ronson
6	Mark Ronson & The Business Intl
2	Mr. Scruff

cdID	artID	cdTitle	cdPrice	cdGenre	cdNumTracks
1	1	Black Holes and Revelations	9.99	Rock	NULL
2	1	The Resistance	11.49	Rock	NULL
3	2	Ninja Tuna	9.99	Electronica	NULL
4	3	For Lack of a Better Name	9.99	Electro House	NULL
5	6	Record Collection	11.49	Alternative Rock	NULL
6	4	Version	12.49	Pop	NULL
7	5	Merriweather Post Pavilion	12.49	Electronica	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 artist Animal Collective, the artID reference in CD for their album would be invalid. MySQL (using the InnoDB engine) will not allow this:

```
DELETE FROM Artist WHERE artName='Animal Collective';
```

You will get the following error:

```
ERROR Code: 1451. Cannot delete or update a parent row: a foreign
key constraint fails (`usr`.`CD`, CONSTRAINT `fk_cd_art` FOREIGN
KEY (`artID`) REFERENCES `Artist` (`artID`))
```

Exercise: Try removing the artist “Animal Collective” 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 artist completely, you will need to first delete all of their CDs [Output 5].

When you’ve successfully done this, your tables should look like the following:

```
SELECT * FROM Artist;
```

artID	artName
1	Amuse
3	Deadmau5
4	Mark Ronson
6	Mark Ronson & The Business Intl
2	Mr. Scruff

```
SELECT * FROM CD;
```

cdID	artID	cdTitle	cdPrice	cdGenre	cdNumTracks
1	1	Black Holes and Revelations	9.99	Rock	NULL
2	1	The Resistance	11.49	Rock	NULL
3	2	Ninja Tuna	9.99	Electronica	NULL
4	3	For Lack of a Better Name	9.99	Electro House	NULL
5	6	Record Collection	11.49	Alternative Rock	NULL
6	4	Version	12.49	Pop	NULL