COMP1004 Lab Week 4: SQL II

INTRODUCTION

The lab for week 4 contains a set of exercises the purpose of which is to acquaint you with the basic SELECT, as well as with more complex SELECT statements including Subqueries. Remember that there is often more than one solution to a query problem; the most important thing is that your results are correct.

We will start with a variety of queries you can accomplish using SQL's SELECT statement. If you have completed the exercises of Week 3 then your database should contain tables for Actor and Movie records. In order to add some variety for the queries, we should add a few more Movies. If you'd like more practice with INSERT and UPDATE operations then try to match your tables to these:

```
SELECT * FROM Actor;
```

actID	actName
1	Arethan Franklin
2	Barry Nelson
3	Bullet Prakash
4	Daniel Craig
5	James Bond
6	Jonny Walker
7	Laura Dern
8	Mr. Ryan Reynolds

```
SELECT * FROM Movie;
```

mvID	actID	mvTitle	mvPrice	mvYear	mvGenre	mvNumScenes
1	2	Die hard with a Vengeance	12.24	1999	Action	NULL
2	2	Black Snake Moan	9.99	2007	Adventure	NULL
3	1	Snake on a Plane	9.99	2011	Comedy	NULL
4	3	Freeway of Love	9.99	2018	Drama	NULL
5	4	I knew you were waiting for me	12.25	1997	Comedy	NULL
6	5	The Black Panther	10.99	2018	Action	NULL
7	6	The Jungle book	9.99	2015	Adventure	NULL
8	7	Infinity War	8.5	1975	Horror	NULL
9	7	Coming to Europe	12.99	2001	Adventure	NULL
10	8	The midnight	10.99	2019	Drama	NULL

You may have different ID values, **do not worry about this**. If you feel you don't need to practice any more insertions or updates, then you can recreate the exact copy of these tables by using the file **setupWeek4.txt**. To display the content of each table use the SELECT statements:

```
SELECT * FROM Actor;

SELECT * FROM Movie;
```

SQL SELECT

SQL's SELECT statement has a number of different options, some of which we've already seen. The simplest form of a SELECT statement returns all of the information from a single table. The general format is:

```
SELECT * FROM table-name;
```

There are more options to build up more complex queries. In this exercise we shall cover:

1. SELECTing specific columns from a table;
2. Using WHERE clauses to select specific rows;
3. SELECTing from multiple tables;

Exercise: Write a SELECT statement to output all of the information from the Actor table

SELECTING SPECIFIC COLUMNS

Often tables in a database have many columns, and we only require a few. We can select useful ones and use aliases to make them more understandable. The aliases give the columns new names to make them easier to understand. Let's try this with our Movie table:

```
SELECT mvTitle AS Title, mvGenre AS Genre FROM Movie;
```

If you run the command above, you will see that the two columns returned are now named "Title" and "Genre", rather than "mvTitle" and "mvGenre".

Exercise: Write SQL Statements to do the following:

- List the names of the actors in the database
- List the genres of the Movies in the database and call the result" genre"
- List the titles and prices of the Movies in the database

You will find that the genre output contains multiple rows for the same genres. You can avoid this by using the `DISTINCT` keyword in your select statement:

```
SELECT DISTINCT mvGenre AS Genre FROM Movie;
```

Exercise: Write a SELECT statement to find the actor IDs and genres from the Movie table, without duplicate entries.

USING WHERE CLAUSES

By using a WHERE clause, you can return rows from tables that match specific criteria. Some examples of simple WHERE clauses using a single test are:

- `SELECT * FROM Movie WHERE mvPrice >10.0;`
- `SELECT * FROM Movie WHERE mvGenre='Action';`
- `SELECT * FROM Movie WHERE mvYear <> 1999;`

Exercise: Write SQL statements to do the following:

- Find a list of all information on Adventure Movies **[Output 1]**
- Find a list of the titles of all Movies that cost less than 10.00 **[Output 2]**

More complex conditions can be created using the operators AND, OR and NOT. Brackets can be used to control the order of evaluation. For example:

```
SELECT * FROM Movie WHERE (mvPrice < 10.00) AND NOT
(mvGenre = 'Comedy');
```

Will return a list of Movies that cost less than 10.00, but are not Action Movies.

Exercise: Write SQL queries to:

- Find a list of all information on Movies that have the genre 'Drama' or cost more than 10.00 **[Output 3]**
- Find a list of titles of Movies that are Action movies released between year 2000 and 2019. **[Output 4]**
- Find a list of the titles of Movies that cost less than 10.00, and are either "Action", or "Adventure". **[Output 5]**

SELECTING FROM MULTIPLE TABLES

It is often useful to combine information from several tables in a single query. Usually we will use a where clause to clarify which results we want, that is not always the case however. The following example will select every combination of rows from Actor and Movie:

```
SELECT * FROM Actor, Movie;
```

This is a CROSS JOIN, sometimes referred to as a product. In this case, the select query will return 10 * 8 rows, or 80 rows. This is far too many to be useful in this case, we should use a WHERE clause.

Exercise: Write an SQL query to:

- Return all the information from Actor and Movie where the actID records are the same for both tables **[Output 6].**
- Find a list of the titles of all Movies by Barry Nelson **[Output 7].**

SUBQUERIES (USING EXISTS, IN, ANY/ALL OPTIONS)

Combining multiple tables can lead to extremely complex queries very quickly. It is sometimes easier to use subqueries to pass values into other queries. For example, suppose we want to find out a list of Movies that cost the same as ('Freeway of Love');

```
SELECT mvTitle FROM Movie WHERE mvPrice =
     (SELECT mvPrice FROM Movie WHERE mvTitle = 'Freeway of
     Love');
```

The output of the query should return the table below;

mvTitle
Black Snake Moan
Snake on a Plane
Freeway of Love
The Jungle book

It is important to remember that there are various ways of comparing values with the results of a subquery. What you do is dependent not only on what results you want, but also how many values you can expect the subquery to return. In general:

1. If the subquery returns a single value, you can use the normal conditional operators (=, <, >, <>, etc.);
2. If the subquery returns multiple values, you can use IN, ANY, ALL and NOT to compare a single value to a set;
3. You can use (NOT) EXISTS to see if a set is empty or not

For example, to find a list of all Movies of the same genre as any that the actor "Laura Dern%" has led, we can use the following:

```
SELECT mvTitle FROM Movie WHERE mvGenre IN
     (SELECT mvGenre FROM Movie, Actor WHERE Actor.actID =
          Movie.actID AND actName LIKE "Laura Dern%");
```

The output of the query should return the table below;

mvTitle
Black Snake Moan
The Jungle book
Infinity War
Coming to Europe

Exercise: Write the following queries:

- Use a subquery to find a list of Movies that have the same genre as 'Comedy' **[Output 8].**
- Use IN to find a list of the titles of movies that are the same price as any 'Action' movie **[Output 9].**
- Use ANY to find the titles of Movies that cost more than at least one other Movie **[Output 10].**

Use ALL to find a list of Movies titles that cost more or the same as all other Movies **[Output 11].**

- Use EXISTS to find a list of Actors who led a “Drama” movie **[Output 12].**

Check carefully your last output (corresponding to Output 12). Is it actually true that the list of Actor you obtained led a “Drama” movie? Do you need to add an extra condition?

Exercise: Write the following queries:

- Find the names of Actors who led movies cheaper than 10 pounds **[Output 13].**
- Find Title of Movies led by actors who have their names start with ‘B’ **[Output 14**].
- Find all information about Movies which are not Drama having price less than 9:00 pounds **[Output 15].**

Answer Sheet for COMP1004 Lab Week 4

Name: ID:

Please provide your SQL Code for Exercises 1-15 described above.

OUTPUT for Exercise 1 **[0.5 marks]**

OUTPUT for Exercise 2 **[0.5 marks]**

OUTPUT for Exercise 3 **[0.5 marks]**

OUTPUT for Exercise 4 **[0.5 marks]**

OUTPUT for Exercise 5 **[0.5 marks]**

OUTPUT for Exercise 6 **[0.5 marks]**

OUTPUT for Exercise 7 **[0.5 marks]**

OUTPUT for Exercise 8 **[0.5 marks]**

OUTPUT for Exercise 9 **[0.5 marks]**

OUTPUT for Exercise 10 **[0.5 marks]**

OUTPUT for Exercise 11 **[0.5 marks]**

OUTPUT for Exercise 12 **[0.5 marks]**

OUTPUT for Exercise 13 **[0.5 marks]**

OUTPUT for Exercise 14 **[0.5 marks]**

OUTPUT for Exercise 15 **[0.5 marks]**