COMP1004 Lab Week 5: SQL III

INTRODUCTION

This exercise will cover queries you can accomplish using SQL's SELECT statement as seen during the lasts lectures. We will use tables created in previous exercises. Your tables should be similar to the following ones:

```
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

Remember that there is often more than one solution to a query problem; also, the ID numbers will vary depending on how your database is set up, and the order of output is also unimportant.

ORDER BY

You can order results in the output should you wish the output to be in a more readable form. The ORDER BY clause allows you to specify columns whose data will order the rows, and whether the ordering is ascending or descending. For example, to order Movies by title alphabetically, you would use:

```
SELECT * FROM Movie ORDER BY mvTitle;
```

By default, results are sorted in ascending order when an `ORDER BY` is used. To specify which we require, we use ASC or DESC:

```
SELECT * FROM Movie ORDER BY mvTitle ASC;

SELECT * FROM Movie ORDER BY mvTitle DESC;
```

You can order first by one column, then by another. This can be done for any number of columns. For example, to order first by genre, then by title we would use:

```
SELECT * FROM Movie ORDER BY mvGenre, mvTitle;
```

Exercise: Write SQL queries to:

- List the titles and prices of Movies in order of price from highest to lowest **[Output 1]**
- List the titles and prices of Movies in order of price from lowest to highest **[Output 2].**
- List the titles, genres and prices Movies in alphabetical order by genre, then by price from the highest price to the lowest one **[Output 3].**

AGGREGATE FUNCTIONS, GROUP BY AND HAVING

The last feature of SQL SELECT seen in this module is the use of aggregate functions. Used with GROUP BY and HAVING clauses, these can produce summary information from a table, or set of tables. The common functions you will need to know for this module are:

- `COUNT([columns])` returns the number of rows in that column. You can use `COUNT(*)` to count the number of rows returned by a select statement. You can also use `COUNT(DISTINCT column)` to count the number of distinct values for a specific column.
- `SUM(column)` returns the sum of all values in a column.
- `MAX(column)` returns the maximum value.
- `MIN(column)` returns the minimum value.
- `AVG(column)` returns the average value.

The mathematical functions above require the data to be of a numeric type like REAL or INT.

Exercise: Write queries to do the following:

- Find the lowest price of any Movie **[Output 4].**
- Find the number of Movies costing 9.99 **[Output 5].**
- Find the title of the most expensive action Movie(s) **[Output 6].**
- Find the number of different Genres in the Movie table **[Output 7].**

- List all the information about the cheapest Movie **[Output 8].**

As above, aggregate functions will normally work on an entire table, unless you use a GROUP BY clause. This is done by selecting some normal columns, and aggregate functions, then grouping the results. For example, to find the number of Movies in each genre:

```
SELECT mvGenre AS Genre, COUNT(*) FROM Movie GROUP BY mvGenre;
```

We can then use a HAVING clause, which is essentially the same as a where. The difference is that a WHERE is applied before GROUP BY, the HAVING clause is applied after rows have been grouped, and aggregate functions have been calculated.

To find the number of Movies in each Genre, but only where that count is greater than 2:

```
SELECT mvGenre AS Genre, COUNT(*) AS Count FROM Movie
     GROUP BY mvGenre HAVING Count > 2;
```

Notice that it is often convenient to give an aggregate function an alias.

Exercise: Write queries to do the following:

- Find a list of actor names, the number of movies they have led, and the average price for their movies. Only return results for actors with more than one movies **[Output 9].**
- Find a list of actor names, the number of movies by that actor and the average price for their movies but not including 'Drama' (you might like to use a WHERE in this one too) **[Output 10].**

Answer Sheet for COMP1004 Lab Week 5

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]**