EXERCISE 1

Introduction to exercise 1

Aggregation functions (like COUNT, MAX, MIN, SUM, AVG, and so on) can be used to convert collections of values into single values. Take a look at the execution of this query:

```sql
SELECT Gender, AVG(Age) AS AverageAge
FROM RandomPeople
WHERE Gender = 'F'
GROUP BY Gender;
```

LOGICAL EXECUTION DIAGRAM:

(FROM RandomPeople) WHERE Gender = 'F'

RandomPeople		
Name	Gender	Age
Beyoncé	F	37
Laura Marling	F	28
Darren Hayes	M	46
Bret McKenzie	M	42
Jack Monroe	NB	30

⇓

GROUP BY Gender

Name	Gender	Age
{Beyoncé, Laura Marling}	F	{37, 28}

⇓

AVG(Age) AS AverageAge

Name	Gender	AverageAge
{Beyoncé, Laura Marling}	F	AVG({37,28})

⇓

SELECT Gender, AverageAge

Name	Gender	AverageAge
{Beyoncé, Laura Marling}	F	32.5

⇓

RESULT

Gender	AverageAge
F	32.5

After executing GROUP BY, we ended up with multiple values for **Name** (Beyonce and Laura Marline) and **Age** (37 and 28), within a single group. We then executed the aggregation function **AVG(Age).** **So, we ended up with one value for Age instead of many, in each group.** On the other hand, we didn't apply any aggregation function to the Name column. This is why we can SELECT AVG(Age), but we cannot SELECT Name without causing an error. Of course, we can SELECT **Gender**, because writing GROUP BY Gender ensures there is only one value of Gender for each group.

In every question of this exercise we will use aggregation functions. We will use them in the SELECT clause and in the HAVING clause. We cannot ever use aggregation functions in the WHERE clause, and in fact this is the whole point of the HAVING clause. The HAVING clause is essentially a WHERE clause for aggregation functions.

If we use an aggregation function in the SELECT clause with a GROUP BY, then we get one value for each group. If we want to discard some of the groups, conditioning on the value of an aggregation function, then we use the aggregation function in the HAVING clause. If we use an aggregation function in the SELECT clause without a GROUP BY, then we get one value for the whole table.

1.
```
SELECT S.businessID, S.businessName, COUNT(ES.businessID) AS ...
FROM ..., ...
WHERE ...
GROUP BY ...;
```

Hint: A business has sponsored an event if the businessID appears alongside the eventID in the EventSponsors table. For each business sponsor (businessID), we want to count the number of events (eventID).

Remarks: The EventSponsors table is a many-to-many relation between Events and Sponsors.

2.
```
SELECT S.businessID, S.businessName
FROM Sponsors S, EventSponsors ES
WHERE ...
GROUP BY ...
HAVING ... > ...;
```

Hint: A business has sponsored more than 3 events if the businessID appears in the EventSponsors table more than 3 times. Whenever we condition on an aggregation function (e.g., COUNT) we need to put the condition in HAVING and not in WHERE.

```
3.
SELECT E.eventID, E.eventName, (COUNT(...)) as total
FROM Events E, Tickets T
WHERE ...
GROUP BY ...;
```

Hint 1: A ticket is sold by a promoter if the eventID appears in the Ticket table alongside a promoterID that is not NULL. Beware, some sold tickets are not sold by promoters (in which case the promoterID in the Ticket table is NULL). So, it is not sufficient to use ticketNumber as in COUNT(T.ticketNumber) because then you will end up counting sold tickets that were not sold by promoters. Also beware, you should include events even if no tickets have been sold (so that the total count will display 0). In this case, an eventID in the Events table will have no matching eventID in the Tickets table, so you need to use an outer join when joining Tickets with Events.

```
4.
SELECT ..., ..., ..., ...
FROM Events E, ..., ...
WHERE E.eventID = ... AND ...
GROUP BY ..., ..., ..., ...
HAVING COUNT(...) = (SELECT MAX(COUNT(...))
                     FROM ...
                     WHERE ... = E.eventID
                     GROUP BY ...);
```

*Hint: The alias **E** appears in the main query and in the nested subquery. So, this is a correlated nested query. This means that the nested subquery will execute once for each eventID. For each eventID we want to find the promoter whose promoterID appears the most times in the Tickets table.*

Remarks: In the nested subquery we use MAX(COUNT(...)). This means we first apply COUNT to give us one value for each group, and then we apply MAX to the result, giving one value for the whole subquery. This is why we can write "COUNT(...) = MAX(COUNT(...))" without the "=" symbol causing an error due to too many values on the right hand side.

```
5.
SELECT ...
...
HAVING ... >= (SELECT ...
               ...);
```

Hint: This will also need to be a correlated nested query, like question 4 above. For each eventID we want to find the promoter whose promoterID appears in the Ticket table more than the average of the counts of promoterIDs in the Tickets table.

Exercise 2

You need to replace the X in `X:\outputfile.txt` with the drive letter that you want to save to. For example `U:\outputfile.txt` will save to your student drive. You can also replace the word "outputfile" with any name you want, and replace the sentence "query goes here" with any query that you want (for testing). Check with the demonstrator to make sure that you got it right. You will need to do this correctly to submit Assignment 2 (the 3rd Assignment).

Exercise 3

1.
```
SELECT ...
...
WHERE UPPER(...) LIKE '...' OR UPPER(...) LIKE '...';
```

2.
```
SELECT ...
...
WHERE LOWER(...) ...;
```

3.
```
SELECT ...
...
AND TO_CHAR(..., ...) BETWEEN '...' AND '...';
```

Hint: An event is held on its booking date which appears in the EventVenues table. We use TO_CHAR() to extract just the year, as in 'YYYY', from the booking dates. An event may be held on more than one day, so each combination of eventID and eventName in the result might not be distinct from the others.

4.
```
SELECT ...
...;
```

Hint: An event requires security if the attribute securityRequired (from the Events table) holds the value 'Y'. The number of people expected to attend the event is given as venueCapacityRequired in the Events table. Past events are those events whose bookingDate is before today's date, and today's date is stored by Oracle in the variable SYSDATE.

5.
```
UPDATE Venues
SET ...
WHERE ...;
```

Hint: To increase the cost per day by 10% we can multiply it by 1.10. You can use the venueID of 'V00001' to find the Local Town Hall.

```
6.
SELECT ...
FROM ...
WHERE ...;
```

Hint: This one is straight forward and is just here to get you ready for question 7. You can use the eventID ‘E00018’ to find the School of SOng and Dance’s production of Fame.

```
7.
SELECT ... - (SELECT ...
              FROM ..., ...
              WHERE ... = 'E00018'
              AND ...)
       AS "Number of Tickets Remaining"
FROM ...
WHERE ... = 'E00018';
```

Hint: The event was held twice (on two different nights). So, the total number of tickets originally available was twice the value in venueCapacityRequired. They sold some tickets, and you can use the subquery to find out how many tickets were sold.

```
8.
SELECT ...
...;
```

Hint: The event type is stored in the variable typeOfEvent in the Events table.

```
9.
SELECT totalCostVenue + totalCostEquip + totalCostCatSec AS "Total Cost"
FROM (SELECT ... AS totalCostVenue
      FROM ..., ...
      WHERE ... AND ... = 'E00007'),
     (SELECT SUM(... * ... * ...) AS totalCostEquip
      FROM EventEquipments EE
      WHERE ... = 'E00007'),
     (SELECT ... AS totalCostCatSec
      FROM ...
      WHERE ... = 'E00007');
```

Hint: We are treating the results of the nested queries as tables and joining them. Each of the three nested queries produces one value. So, when we join them we don’t need a join clause. Lucky us. The venue costs are stored in Venues.costPerDay. The equipment cost is given as the product of the unitPrice, quantity and noOfDays from the EventEquipments table. The cost of catering and security is stored in totalCostCatSec in the EventServices table.

10.

```
SELECT totalCostVenue + NVL(totalCostEquip,0) + NVL(totalCostCatSec,0) AS
"Total Cost"
FROM
(SELECT ...),
(SELECT ...),
(SELECT ...);
```

Hint: This is very similar to the question 9, but Mimed Moments production doesn't actually need security or catering. After you get your query working, remove the NVL function and see what happens. Make sure you understand why this happens.

11.

```
SELECT R1.eventID, ... AS "Total Cost"
FROM  (SELECT ..., ... AS totalCostVenue
       FROM ..., ...
       WHERE ...
       AND TO_CHAR(..., ...) = '...'
       GROUP BY ...) R1,
      (SELECT ..., ... AS totalCostEquip
       FROM ...
       GROUP BY ...) R2,
      (SELECT ..., ... AS totalCostCatSec
       FROM ...
       GROUP BY ...) R3
WHERE ... = ...(+)
AND ... = ...(+);
```

Hint: Here we have given the aliases R1, R2, and R3 to the results of the subqueries. Unlike the last two questions (questions 9 and 10), the subqueries here do not produce only one value each. Each subquery produces a whole table of values. So, when we join the subquery results (R1, R2 and R3) we do need a join clause (in fact we need two of them). This means we need to SELECT an attribute in each subquery that we can join on.

MAKE SURE YOU UNDERSTAND QUESTION 11 TO PREPARE FOR NEXT WEEKS LAB.