Appendix

Consider the following relational table `OrderDetails`.

Table 1: Table: OrderDetails

OrderDetailID	OrderID	ProductID	Quantity
1	10248	11	12
2	10248	42	10
3	10249	14	9

Here are the following SQL queries that involve `sum`, `count` and `avg`. These may be useful for the exam.

- ```
  SELECT SUM(Quantity)
  FROM OrderDetails;
  ```

  The output of the above SQL query is a table with one column called `Sum(Quantity)` with a single row containing 31 i.e. the sum function sums up every single item in that column.

- ```
  SELECT AVG(Quantity)
  FROM OrderDetails;
  ```

  The output of the above SQL query is a table with one column called `AVG(Quantity)` with a single row containing 10.33 i.e. the AVG function returns the average of all the items in that column.

- ```
  SELECT COUNT(Quantity)
  FROM OrderDetails;
  ```

  The output of the above SQL query is a table that has one column called `Count(Price)` and one row with the value 3 i.e. the count function just counts all the tuples that were present from `OrderDetails`.

Note: You can always add a `WHERE` clause to the above examples if you like to perform 'selection' and select only certain tuples that meet some condition as specified in the `WHERE` clause. Now, let us look at some other example involving 'Group By' and 'Having'.

- ```
  SELECT COUNT(OrderID), OrderID
  FROM OrderDetails
  GROUP BY OrderID
  HAVING COUNT(OrderID) > 1
  ```

  The best way to understand this is first start with 'Group By'. What 'Group By' does is that it looks at the column 'OrderID' and figures out how many unique items are there in that column. We know from the table above that there are 2 unique items i.e. $10248, 10249$. Ok, now let us look at the `SELECT` statement. The output of this entire SQL query will have two columns i.e. `Count(OrderID)` and `OrderID` where the column `OrderID` will contain those unique values fetched from 'Group By'. Now in the first column `CountOrderDetailID`, you are simply going to count how many tuples in the entire table have the `OrderID` as 10248 and 10249. Finally, we only want those values that have `COUNT(OrderID)` of > 1. Here is the final output produced:

  ```
  COUNT(OrderID), OrderID
  2                 10248
  ```

- Let us look at another similar example:

  ```
  SELECT COUNT(OrderID), OrderID
  FROM OrderDetails
  GROUP BY OrderID
  HAVING COUNT(OrderID) > 0
  ```

Here is the final output produced:

```
COUNT(OrderID), OrderID
2                 10248
1                 10249
```

IN operator

The IN operator is a shorthand for multiple OR conditions.

- ```
  SELECT * FROM Customers
  WHERE Country IN (`Germany', `France', `UK');
  ```

  The above SQL statement selects all customers that are located in "Germany", "France" and "UK":

- ```
  SELECT * FROM Customers
  WHERE Country NOT IN (`Germany', `France', `UK');
  ```

  The above SQL statement selects all customers that are NOT located in "Germany", "France" or "UK":

- ```
  SELECT * FROM Customers
  WHERE Country IN (SELECT Country FROM Suppliers);
  ```

  The above SQL statement selects all customers that are from the same countries as the suppliers:

NOT IN operator

Very similar to the IN operator, however, the NOT negates it. The following statement returns a list of tracks whose genre id is not in a list of (1,2,3).

- ```
  SELECT
   trackid,
   name,
   genreid
  FROM
   tracks
  WHERE
   genreid NOT IN (1,2,3);
  ```

  You can also replace the hard coded values of (1,2,3) with some other SELECT statement that returns back a relation of one column.

NOT EXISTS

The SQLite EXISTS condition can also be combined with the NOT operator. For example,

- ```
      SELECT *
  FROM departments
  WHERE NOT EXISTS (SELECT *
                    FROM employees
                    WHERE departments.department_id = employees.department_id);
  ```

This SQLite EXISTS example will return all records from the departments table where there are no records in the employees table for the given department_id.
  ```

HTML template file

```
<html>
    <head>
        <title>
        </title>
    </head>
    <body>

    </body>
</html
```

The element $< b >$ is used for bold and the element $< i >$ is used for italics