Databases Coursework 2

Jonathan Alderson

1. Created a database for table 1. It should look like this. This database was initially created using the command.

   ```
   CREATE TABLE shopping( Product text PRIMARY KEY,Quantity
   int);
   ```

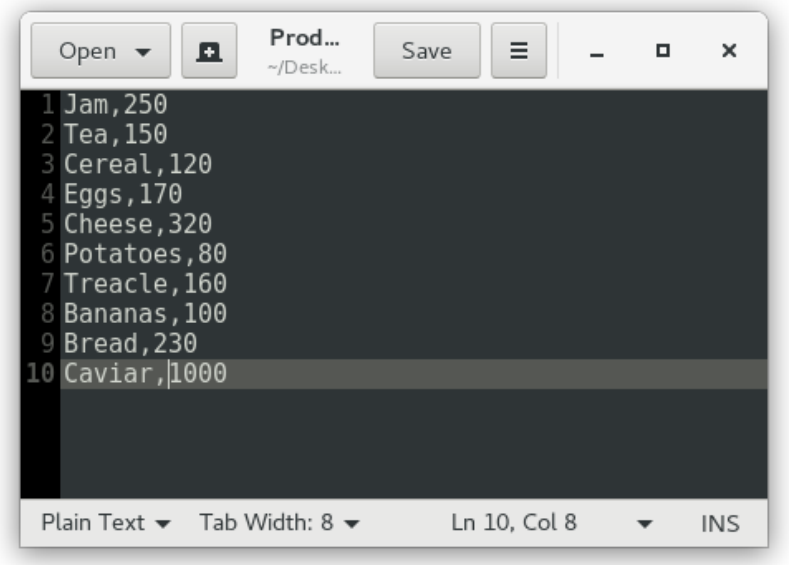
   Then each column was added by using the commands, INSERT INTO and VALUES.

   ```
   INSERT INTO shopping(Product,Quantity)
   VALUES(“Jam”,1);
   INSERT INTO shopping(Product,Quantity)
   VALUES(“Bread”,2);
   INSERT INTO shopping(Product,Quantity)
   VALUES(“Tea”,5);
   INSERT INTO shopping(Product,Quantity)
   VALUES(“Cereal”,1);
   ```

   This gave me the resulting table.

   ```
   Product     Quantity
   ----------  ----------
   Jam         1
   Bread       2
   Tea         5
   Cereal      1
   ```

2. For this task, we had to create a text file, and insert all the data we would need into it. Then by using the .import command, we would be able to import the data into the file instead of having to put it in manually. I first created a text file called ProductData.txt. It looks like



I then created a new table called productData by using the commands

```
CREATE TABLE products( Product text PRIMARY KEY, Price INT);
```

Then I used the .import command to load all the data from the file.

First of all I had to use

```
.separator ","
```

Or else I would get some nasty errors.
Then I could run the command

```
.IMPORT ProductData products
```

The table is shown on the left and is now complete.

Product	Quantity
Jam	250
Tea	150
Cereal	120
Eggs	170
Cheese	320
Potatoes	80
Treacle	160
Bananas	100
Bread	230
Caviar	1000

3. To find how many products there are that are in both shopping and in products. We can use the commands

```
SELECT Product from shopping INTERSECT SELECT Product from
products;
```

This gives the output.

```
Bread
Cereal
Jam
Tea
```

We can then use the command

```
SELECT COUNT(*) FROM (SELECT Product from shopping INTERSECT
SELECT Product from products);
```

Then we get 4.
So this all works great.

4. To see how many items were bought on the shopping trip. We use the command.

```
SELECT SUM(Quantity) FROM (select Quantity from shopping WHERE
Product IN (SELECT Product FROM shopping INTERSECT SELECT
Product FROM products));
```

This took a little while to do. But basically, this command first finds all the rows in the shopping table that have the same product name as in the product table. It then finds all the rows in the shopping table which have the same product name as in the union we just found. Then sums up the quantities of those rows. To get the final answer.

5. This task was much more straight forward, to do this one. The command is just

```
SELECT Product FROM products WHERE Price > 120;
```

This didn't take long at all, it just gives the names of the products in the products table whose corresponding Price value is greater than 120.

6. The output this produced is

```
Product     Quantity * Price
----------  ----------------
Jam         250
Tea         750
Cereal      120
Bread       460
```

This output shows the cost of the item you want to buy, multiplied by it's cost. SO if you bought 5 lots of tea, at 150 each. You would have spent 750. This is like a shopping receipt.

7. To find the total cost of all of this. We can change the command it takes the output from task And performs a count() on the row containing the prices. It will look like this. After trying for what felt like a lifetime. I realised that what I was looking for was far simpler than I had realised. Causing me to be more than a little silly when I discovered the answer was.

```
SELECT SUM(Quantity * Price) FROM products INNER JOIN shopping
ON products.Product = shopping.Product;
```

8. The following statement

```
SELECT Quantity * Price FROM products INNER JOIN shopping ON
products.Product = shopping.Product WHERE shopping.Product =
'Tea';
```

It gives the output

```
750
```

This command finds out how much tea you have bought, and multiplied it by the cost of the tea. Since the tea is 15, and you have bought 5 teas, it will show the result to be 750.

9. To answer this question, we simply need to change the prior statement so that instead of just checking to see if the product name is 'Tea' we can do a union so that it does it for 'Jam' or 'Tea'. Then we wrap the output in the SUM() command, to return one value instead of two.

```
SELECT SUM(Quantity * Price) FROM products INNER JOIN shopping
ON products.Product = shopping.Product WHERE shopping.Product
= 'Tea' OR shopping.Product = 'Jam';
```

Part B

10. To do this task. First the two tables were created, then the .seperator was used so that the .import command could be used. From then. The command was used which found the number of students taking each module and then ordered by the number of students taking it. Here is what my .sql test file looked like at the time.

```
/* printing blank space */

.print ''
.print ''
.print ''

/* Question 10 */
.print 'All Modules sorted by the number of students taking
them'
.print ''
CREATE TABLE Teaches(Lecturer text , Module int);
CREATE TABLE Studies(Student text, Module int, Grade int );
.separator ','
.import PartBData/TeachesData Teaches
.import PartBData/StudiesData Studies

SELECT Module, COUNT(*) FROM Studies GROUP BY Module ORDER BY
COUNT(*) DESC;
.print 'Task ten complete'
```

The output from this task was.

```
Module      COUNT(*)
----------  ----------
COMP1300    3
COMP1500    3
COMP1600    3
COMP2300    2
COMP2700    2
COMP1400    1
COMP2200    1
COMP3400    1
COMP3440    1

Task ten complete
```

11. Task 11 took a very long time. The closest I seemed to get for a long time was only being able to do this for one lecturer at a time. But after using multiple statements inside each other I finally got it to work after a very long time.

```
Lecturer    COUNT(Lecturer)
----------  ---------------
Doran       6
Jones       4
McCarthy    4
Smith       4
```

12. Task 12 also took a very long time. I am unsure weaher I have done this in the most efficient way. But it works correctly and gives a nice output. It is also an understandable expression.

```
SELECT  Lecturer,Module,COUNT(Student) FROM (SELECT
Lecturer,Module,Student FROM (SELECT DISTINCT
Teaches.Lecturer,Teaches.Module, Studies.Student FROM Studies
INNER JOIN Teaches ON Studies.Module = Teaches.Module)) GROUP
BY Lecturer,Module ORDER BY Lecturer;
```

This gives the output

```
Lecturer    Module      COUNT(Student)
----------  ----------  --------------
Doran       COMP1600    3
Doran       COMP2300    2
Doran       COMP2700    2
Doran       COMP3440    1
Jones       COMP1300    3
Jones       COMP1500    3
Jones       COMP2200    1
McCarthy    COMP1600    3
McCarthy    COMP3440    1
Smith       COMP1300    3
Smith       COMP1400    1
Smith       COMP3400    1
```

13. Finally for question 13. I found this one easier than the last two, since the answer built off the previous one. To do this, I used the EXCEPT command in SQL. This is the equivelent of a complement in set theory. I initially selected all Modules from the Teaches table. Then did an EXCEPT with this table and a table containing all he results, but only selecting those with a grade less than 40. This would result in a table where every person in the module had passed. Finally we do a count on the whole line to return the number of modules.

This query will show all the modules in which every student passed.

```
SELECT DISTINCT Module from Teaches EXCEPT SELECT  Module FROM
(SELECT DISTINCT Teaches.Module, Studies.Grade FROM Studies
INNER JOIN Teaches ON Studies.Module = Teaches.Module WHERE
Grade < 40);
```

This query will show the number of modules in which every student passed.

```
SELECT COUNT(*) FROM (SELECT DISTINCT Module from Teaches
EXCEPT SELECT  Module FROM (SELECT DISTINCT Teaches.Module,
Studies.Grade FROM Studies INNER JOIN Teaches ON
Studies.Module = Teaches.Module WHERE Grade < 40));
```

The output looks like

```
Number of modules in which everyone passed

Module
----------
COMP1300
COMP1400
COMP2200
COMP2300
COMP2700
COMP3440

Task 12 Complete
```

And for the second line

```
COUNT(*)
----------
6

Task 12 Complete
```