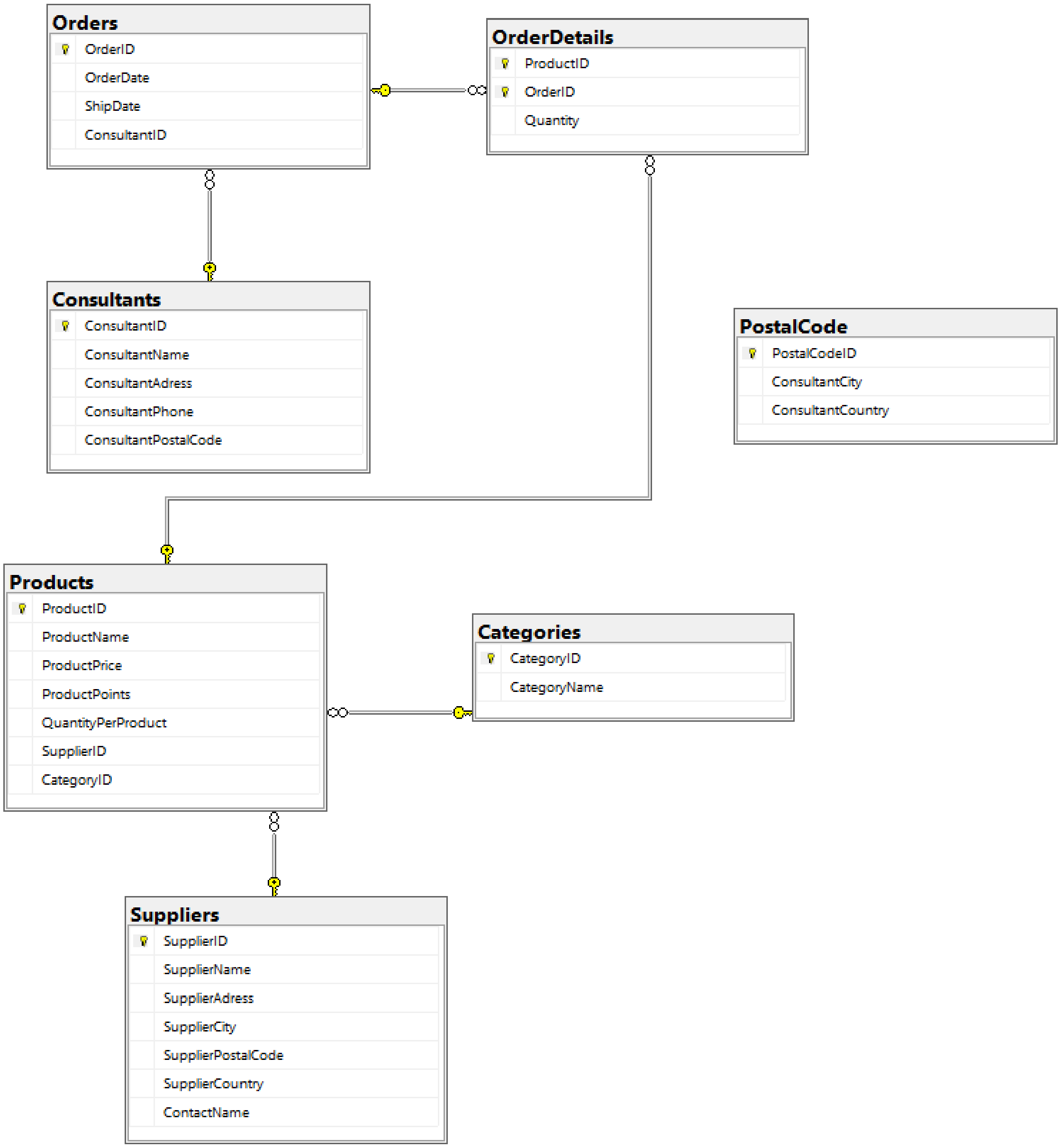
Example DML queries

Given the following ER Diagram. Solve the problems below in SQLite syntax.



1. Return top 5 consultants id's ordered by their postal code.[1]

```
Select ConsultantID
From Consultants
ORDER BY ConsultantPostalCode ASC
LIMIT 5
```

Result:

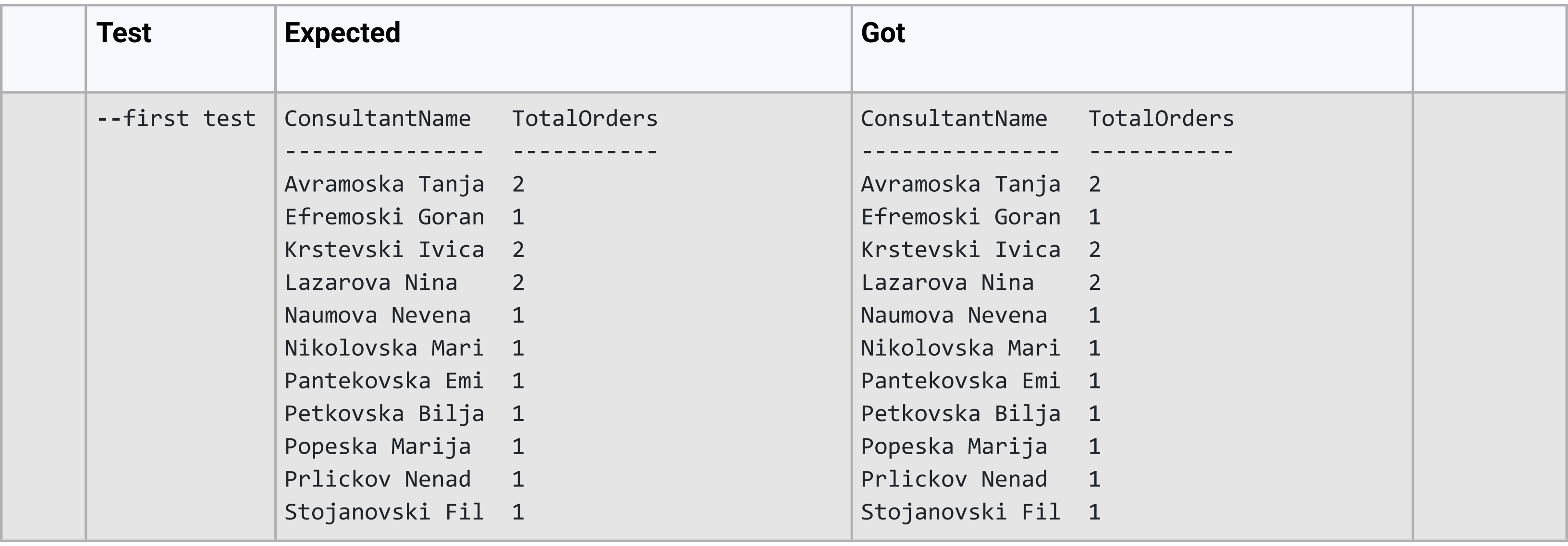
	Test	Expected	Got
	-- first test	ConsultantID ------------ 89002 89100 89555 89203 89075	ConsultantID ------------ 89002 89100 89555 89203 89075

[1] `LIMIT 5` in SQL restricts the query output to 5 rows. It's useful for managing large datasets.

2. Return the number of orders for each customer.[2]

```
Select ConsultantName, COUNT(OrderID) as TotalOrders
FROM Consultants c JOIN Orders o ON c.ConsultantID = o.ConsultantID
GROUP BY c.ConsultantID
ORDER BY ConsultantName ASC
```

Result:



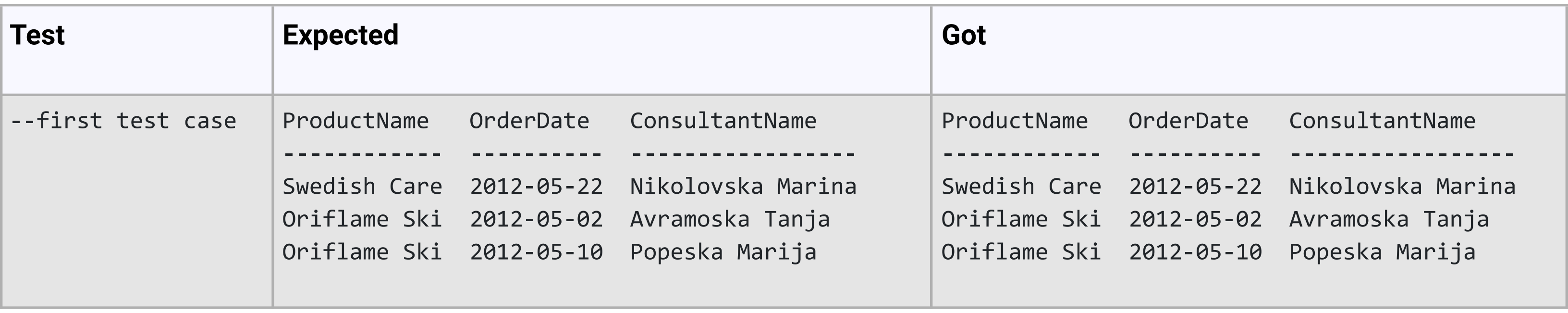
	Test	Expected	Got	
	--first test	ConsultantName TotalOrders --------------- ----------- Avramoska Tanja 2 Efremoski Goran 1 Krstevski Ivica 2 Lazarova Nina 2 Naumova Nevena 1 Nikolovska Mari 1 Pantekovska Emi 1 Petkovska Bilja 1 Popeska Marija 1 Prlickov Nenad 1 Stojanovski Fil 1	ConsultantName TotalOrders --------------- ----------- Avramoska Tanja 2 Efremoski Goran 1 Krstevski Ivica 2 Lazarova Nina 2 Naumova Nevena 1 Nikolovska Mari 1 Pantekovska Emi 1 Petkovska Bilja 1 Popeska Marija 1 Prlickov Nenad 1 Stojanovski Fil 1	

[2] `COUNT(OrderID)` counts the number of orders for each consultant.
`GROUP BY c.ConsultantID` groups the results by consultant ID.
It ensures that each consultant's orders are counted separately.
The `ORDER BY ConsultantName ASC` arranges the results alphabetically by consultant name.

3. Retrieve unique product names that contain 'Care', their order dates, and the names of the consultants who handled the orders.[3]

```sql
SELECT DISTINCT ProductName,  OrderDate  ,ConsultantName
FROM Products p
JOIN OrderDetails od ON p.ProductID = od.ProductID
JOIN Orders o ON od.OrderID = o.OrderID
JOIN Consultants c ON c.ConsultantID = o.ConsultantID
WHERE ProductName LIKE '%Care%'
```

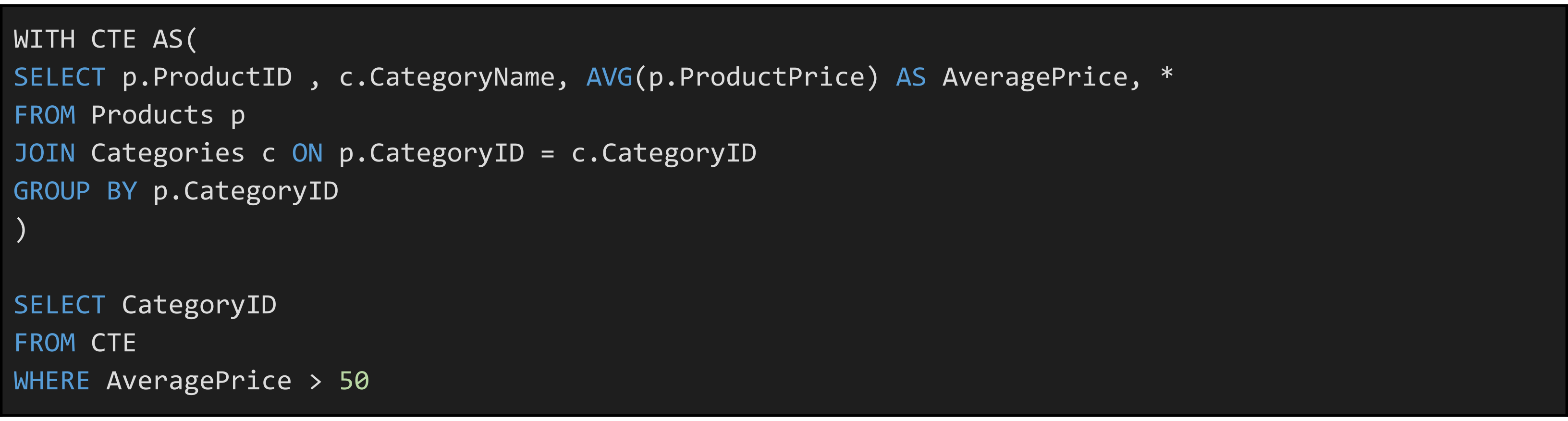
Result:



Test	Expected	Got
--first test case	ProductName OrderDate ConsultantName ------------ ---------- ----------------- Swedish Care 2012-05-22 Nikolovska Marina Oriflame Ski 2012-05-02 Avramoska Tanja Oriflame Ski 2012-05-10 Popeska Marija	ProductName OrderDate ConsultantName ------------ ---------- ----------------- Swedish Care 2012-05-22 Nikolovska Marina Oriflame Ski 2012-05-02 Avramoska Tanja Oriflame Ski 2012-05-10 Popeska Marija

[3] LIKE is used to search for a specified pattern in a column.
It allows wildcard characters like % and _.
% matches any sequence of characters.
_ matches any single character.
For example, LIKE 'a%' matches any string that starts with 'a'.

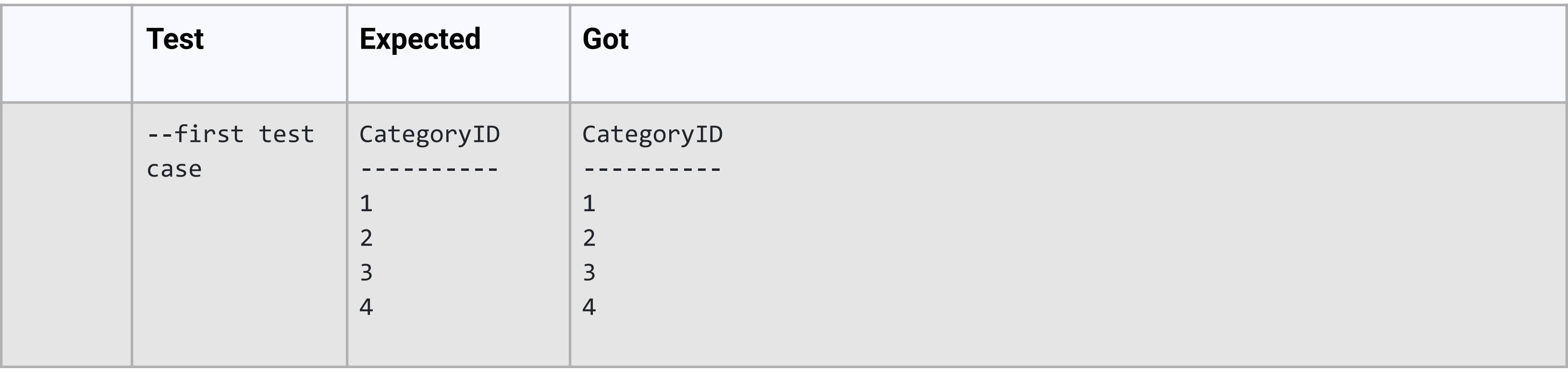
4. Retrieve the average price of products for each category where the average price is greater than 50.[4]



```
WITH CTE AS(
SELECT p.ProductID , c.CategoryName, AVG(p.ProductPrice) AS AveragePrice, *
FROM Products p
JOIN Categories c ON p.CategoryID = c.CategoryID
GROUP BY p.CategoryID
)

SELECT CategoryID
FROM CTE
WHERE AveragePrice > 50
```

Result:



	Test	Expected	Got
	--first test case	CategoryID ---------- 1 2 3 4	CategoryID ---------- 1 2 3 4

[4] The test here is kinda broken, it just shows categoryID, but i would assume the code is correct

5. Retrieve the consultants last name along with their associated order ids, ordered by their last name in descending order.[5]

```sql
SELECT SUBSTR(ConsultantName, 0, INSTR(ConsultantName, ' ')) AS ConsultantLastName , o.OrderID
FROM  Consultants c
JOIN Orders o ON c.ConsultantID = o.ConsultantID
ORDER BY ConsultantLastName DESC;
```

Result:

Test	Expected	Got

-- first test case	`ConsultantLastName  OrderID`
	`------------------  ----------`
	`Trpenoska           11`
	`Stojanovski         2`
	`Prlickov            10`
	`Popeska             4`
	`Petkovska           6`
	`Pantekovska         5`
	`Nikolovska          9`
	`Naumova             12`
	`Lazarova            7`
	`Lazarova            13`
	`Krstevski           8`
	`Krstevski           15`
	`Efremoski           3`
	`Avramoska           1`
	`Avramoska           14`
		`ConsultantLastName  OrderID`
		`------------------  ----------`
		`Trpenoska           11`
		`Stojanovski         2`
		`Prlickov            10`
		`Popeska             4`
		`Petkovska           6`
		`Pantekovska         5`
		`Nikolovska          9`
		`Naumova             12`
		`Lazarova            7`
		`Lazarova            13`
		`Krstevski           8`
		`Krstevski           15`
		`Efremoski           3`
		`Avramoska           1`
		`Avramoska           14`

[5] SUBSTR extracts a substring from a string.
The first parameter is the string to extract from.
The second parameter is the start position (0-based index) of the substring.
The third parameter is the length of the substring to extract.
For example, SUBSTR('hello', 1, 3) returns 'hel'.

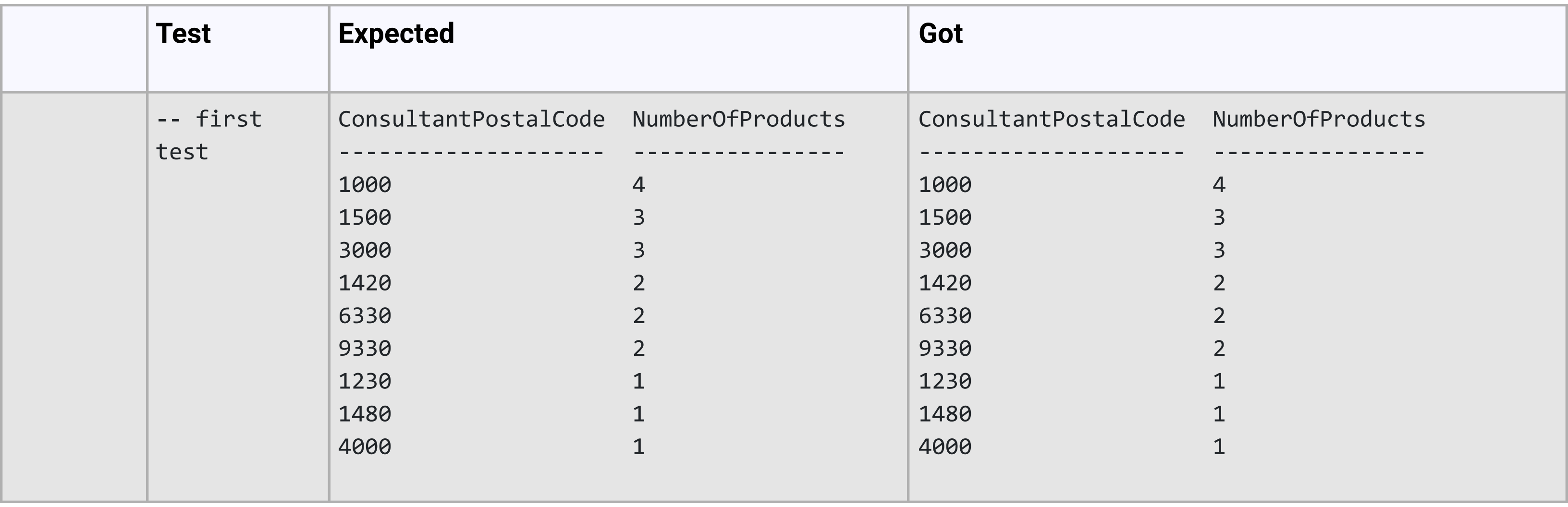
6.Return the number of products per PostalCode, ordered by the number of products in descending order.[6]

```
SELECT ConsultantPostalCode AS ConsultantPostalCode   ,COUNT(p.ProductID) AS NumberOfProducts

FROM Consultants c
JOIN Orders o ON c.ConsultantID = o.ConsultantID
JOIN OrderDetails od ON o.OrderID = od.OrderID
JOIN Products p ON od.ProductID = p.ProductID

GROUP BY c.ConsultantPostalCode
ORDER BY NumberOfProducts DESC
```

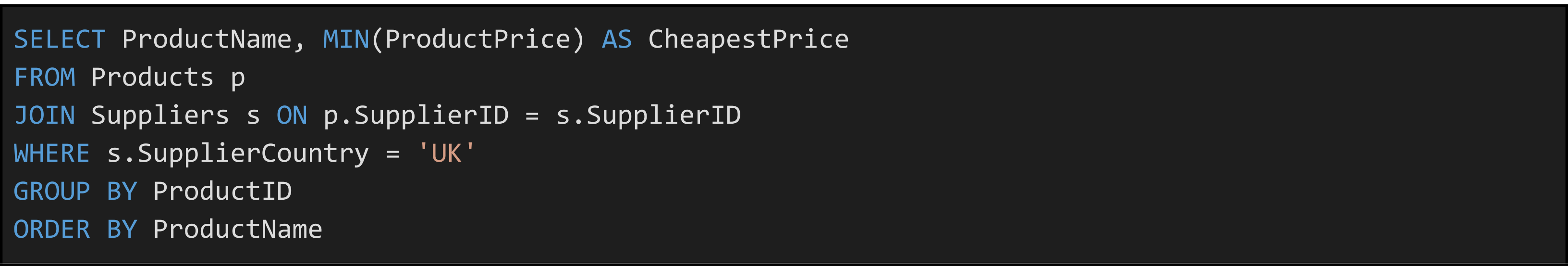
Result:



	Test	Expected	Got
	-- first test	ConsultantPostalCode NumberOfProducts -------------------- ---------------- 1000 4 1500 3 3000 3 1420 2 6330 2 9330 2 1230 1 1480 1 4000 1	ConsultantPostalCode NumberOfProducts -------------------- ---------------- 1000 4 1500 3 3000 3 1420 2 6330 2 9330 2 1230 1 1480 1 4000 1

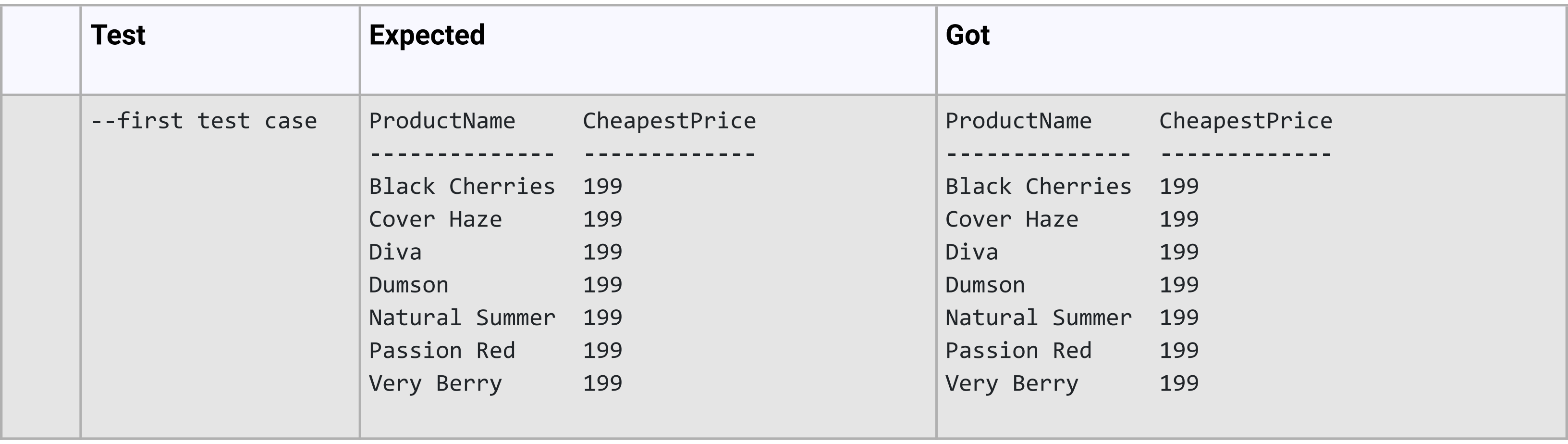
[6] Aggregate functions perform a calculation on a set of values and return a single value.
They summarize data, such as counting, summing, averaging, or finding minimum and maximum values.
COUNT is an aggregate function that counts the number of rows in a group.
GROUP BY is essential for categorizing rows into groups based on common values.
It allows aggregate functions to operate on each group separately.
In this query, GROUP BY groups the results by consultant postal code, enabling the COUNT function to count the number of products for each postal code group.

7. Find the cheapest price of each product supplied by suppliers in the UK.[7]



```sql
SELECT ProductName, MIN(ProductPrice) AS CheapestPrice
FROM Products p
JOIN Suppliers s ON p.SupplierID = s.SupplierID
WHERE s.SupplierCountry = 'UK'
GROUP BY ProductID
ORDER BY ProductName
```

Result:



	Test	Expected	Got
	--first test case	ProductName CheapestPrice -------------- ------------- Black Cherries 199 Cover Haze 199 Diva 199 Dumson 199 Natural Summer 199 Passion Red 199 Very Berry 199	ProductName CheapestPrice -------------- ------------- Black Cherries 199 Cover Haze 199 Diva 199 Dumson 199 Natural Summer 199 Passion Red 199 Very Berry 199

[7] `MIN` and `MAX` are aggregate functions in SQL.
`MIN` returns the smallest value in a set.
`MAX` returns the largest value in a set.
In this query, `MIN(ProductPrice)` finds the lowest price for each product.

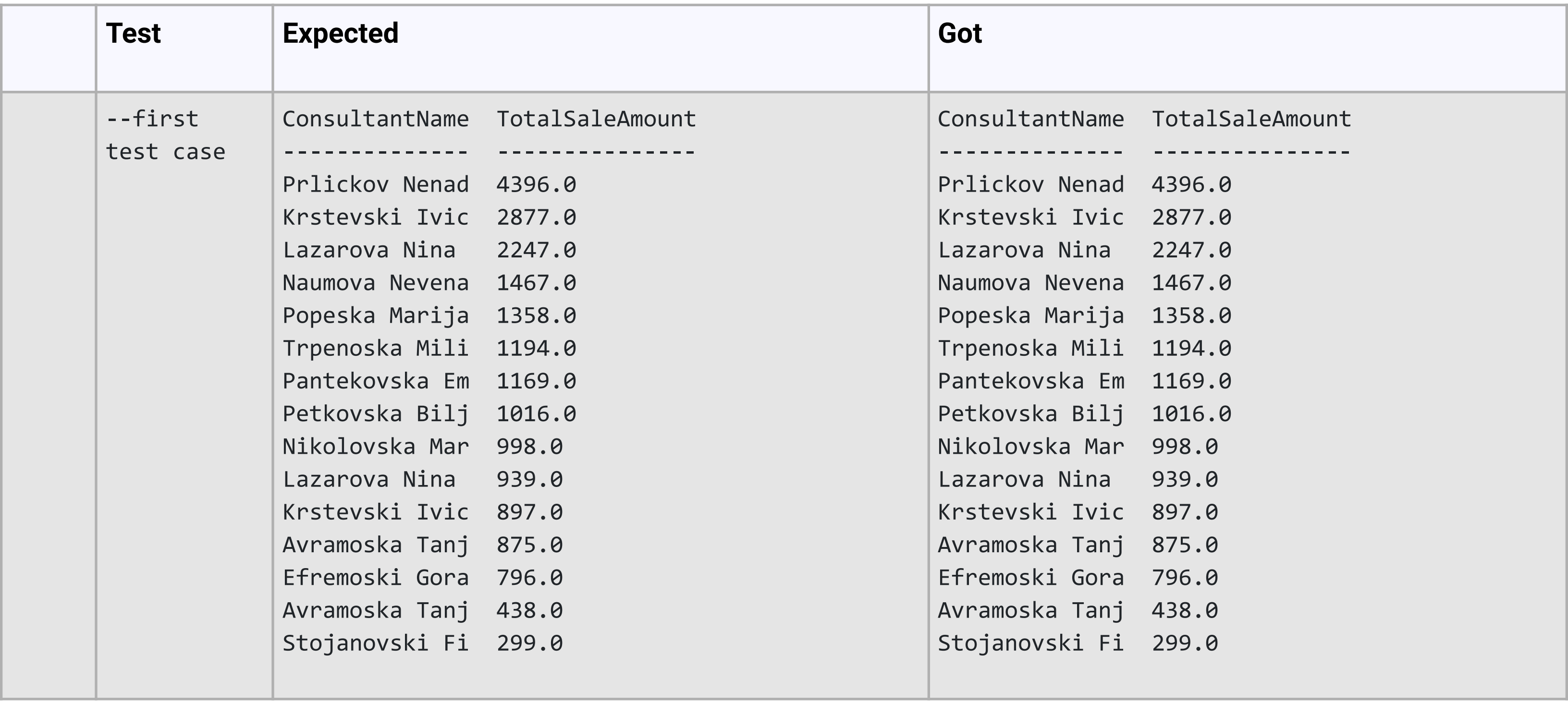
8. Retrieve the consultants name and total sale amount from orders, ordered by their total sale amount in descending order.

TotalSalesAmount should be calculated for each combination of ConsultantName and OrderId.

Expected columns: ConsultantName, TotalSaleAmount[8]

```sql
SELECT ConsultantName, SUM(od.Quantity*p.ProductPrice) AS TotalSaleAmount
FROM Orders o
JOIN OrderDetails od ON o.OrderID = od.OrderID
JOIN Products p ON od.ProductID = p.ProductID
JOIN Consultants c ON o.ConsultantID = c.ConsultantID
GROUP BY ConsultantName, o.OrderId
ORDER BY TotalSaleAmount DESC
```

Result:



	Test	Expected	Got
	--first test case	ConsultantName TotalSaleAmount -------------- --------------- Prlickov Nenad 4396.0 Krstevski Ivic 2877.0 Lazarova Nina 2247.0 Naumova Nevena 1467.0 Popeska Marija 1358.0 Trpenoska Mili 1194.0 Pantekovska Em 1169.0 Petkovska Bilj 1016.0 Nikolovska Mar 998.0 Lazarova Nina 939.0 Krstevski Ivic 897.0 Avramoska Tanj 875.0 Efremoski Gora 796.0 Avramoska Tanj 438.0 Stojanovski Fi 299.0	ConsultantName TotalSaleAmount -------------- --------------- Prlickov Nenad 4396.0 Krstevski Ivic 2877.0 Lazarova Nina 2247.0 Naumova Nevena 1467.0 Popeska Marija 1358.0 Trpenoska Mili 1194.0 Pantekovska Em 1169.0 Petkovska Bilj 1016.0 Nikolovska Mar 998.0 Lazarova Nina 939.0 Krstevski Ivic 897.0 Avramoska Tanj 875.0 Efremoski Gora 796.0 Avramoska Tanj 438.0 Stojanovski Fi 299.0

[8] SUM is used here because it calculates the total sale amount by multiplying the quantity of each product (od.Quantity) by its price (p.ProductPrice) and then summing up these amounts.
COUNT would count the number of rows for each combination of ConsultantName and OrderId, which wouldn't give the total sale amount.

9. Retrieve the three least selling products for each supplier based on the total quantity sold, considering the products that were ordered in the first half of May 2012. [9]

```sql
WITH CTE AS (
    SELECT s.SupplierName, p.ProductName, SUM(od.Quantity) AS TotalQuantitySold
    FROM Orders o
    JOIN OrderDetails od ON o.OrderID = od.OrderID
    JOIN Products p ON od.ProductID = p.ProductID
    JOIN Suppliers s ON p.SupplierID = s.SupplierID
    WHERE o.OrderDate BETWEEN '2012-05-01' AND '2012-05-15'
    GROUP BY s.SupplierName, p.ProductName
)

SELECT SupplierName, ProductName, TotalQuantitySold
FROM CTE
ORDER BY TotalQuantitySold ASC, SUBSTR(SupplierName, INSTR(SupplierName, ' ') + 1) ASC
LIMIT 3;
```

Result:

Test	Expected	Got
-- first test case	SupplierName ProductName TotalQuantitySold --------------- ----------- ----------------- Oriflame France Lucia 1.0 Oriflame German Precious Sp 1.0 Oriflame Sweden Oriflame Oi 1.0	SupplierName ProductName TotalQuantitySold --------------- ----------- ----------------- Oriflame France Lucia 1.0 Oriflame German Precious Sp 1.0 Oriflame Sweden Oriflame Oi 1.0

[9] CTE is used here to simplify complex queries by creating a temporary result set.
It first calculates the total quantity sold for each product from each supplier within a specified date range.
Then, it selects the supplier name, product name, and total quantity sold from the CTE.

10. Calculate the average sales amount per product category for all orders shipped in the second half of May 2012, and list categories with average sales above 1800 in descending order.[10]

```
WITH CTE AS(
SELECT c.CategoryName, AVG(ProductPrice*Quantity) AS Average_Sales_Amount
FROM Products p
JOIN Categories c ON p.CategoryID = c.CategoryID
JOIN OrderDetails od ON p.ProductID = od.ProductID
JOIN Orders o ON od.OrderID = o.OrderID
WHERE o.ShipDate BETWEEN '2012-05-15' AND '2012-05-31'
GROUP BY c.CategoryID
)

SELECT CategoryName, Average_Sales_Amount AS AverageSalesAmount
FROM CTE
WHERE Average_Sales_Amount > 1800
ORDER BY CategoryName DESC
```

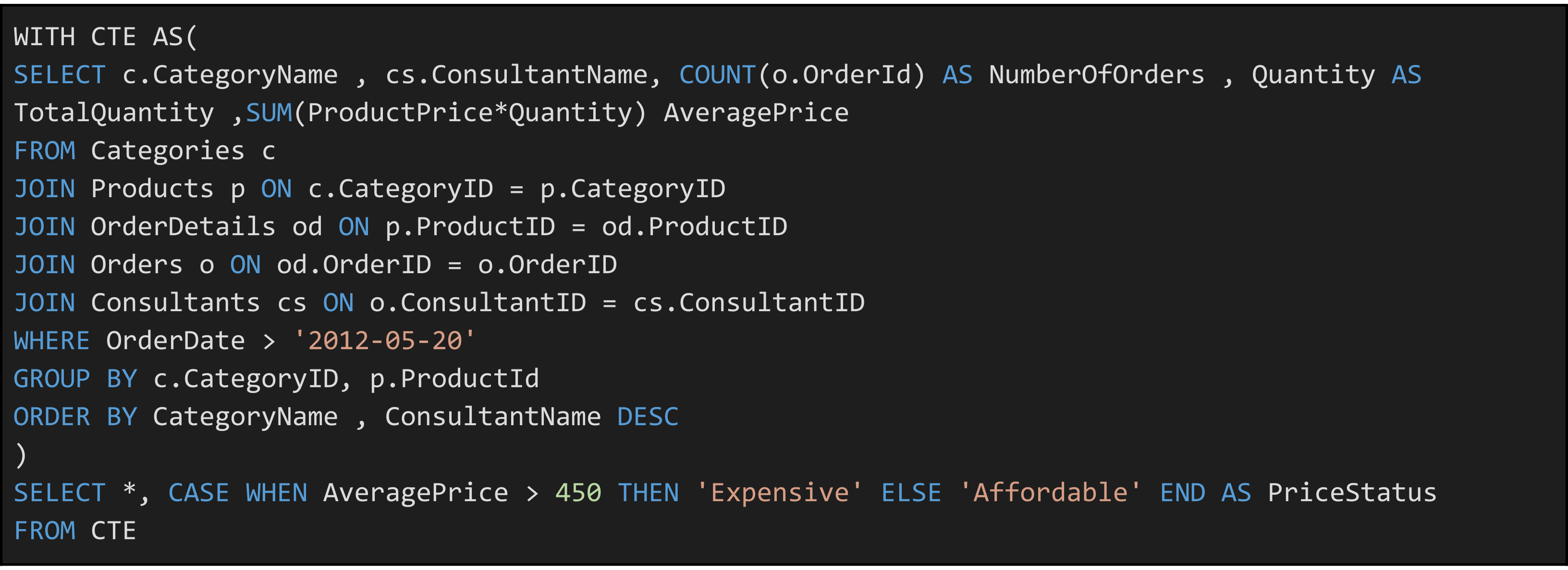
Result:

Test	Expected	Got
--first test case	CategoryName AverageSalesAmount ------------ ------------------ Perfumes 2320.25	CategoryName AverageSalesAmount ------------ ------------------ Perfumes 2320.25

[10] Create a CTE named "CTE."
Join `Products`, `Categories`, `OrderDetails`, `Orders`.
Filter orders shipped between May 15, 2012, and May 31, 2012.
Calculate average sales amount per category.
Select categories with average sales > 1800.
Order results by category name in descending order.

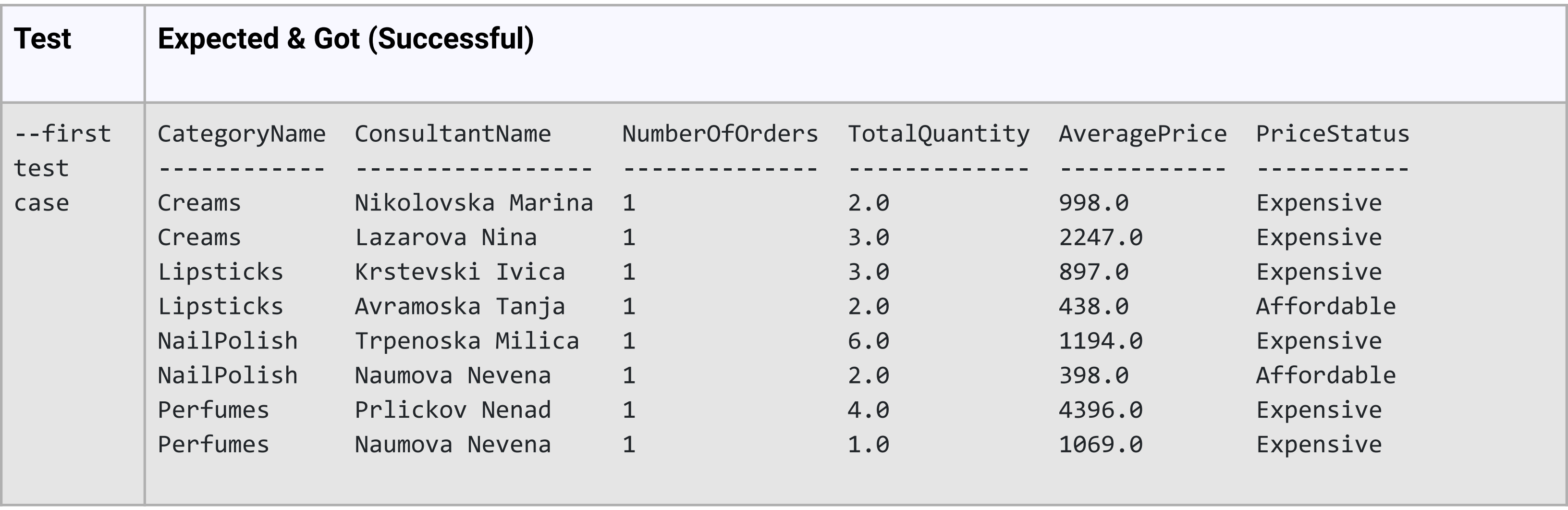
11. Write a SQL query to evaluate the sales of products in each category and the performance of consultants since May 20, 2012. The query should calculate the number of orders, the total quantity of products sold, and the average price per category.

It should label each category as 'Expensive' or 'Affordable' based on whether the average price is above or below $450. The query also needs to list the consultants' names who made sales in each category, sorted by the category name and consultant name in descending order.[11]



```
WITH CTE AS(
SELECT c.CategoryName , cs.ConsultantName, COUNT(o.OrderId) AS NumberOfOrders , Quantity AS
TotalQuantity ,SUM(ProductPrice*Quantity) AveragePrice
FROM Categories c
JOIN Products p ON c.CategoryID = p.CategoryID
JOIN OrderDetails od ON p.ProductID = od.ProductID
JOIN Orders o ON od.OrderID = o.OrderID
JOIN Consultants cs ON o.ConsultantID = cs.ConsultantID
WHERE OrderDate > '2012-05-20'
GROUP BY c.CategoryID, p.ProductId
ORDER BY CategoryName , ConsultantName DESC
)
SELECT *, CASE WHEN AveragePrice > 450 THEN 'Expensive' ELSE 'Affordable' END AS PriceStatus
FROM CTE
```

Result:



Test	Expected & Got (Successful)
--first test case	see output below

```
CategoryName  ConsultantName     NumberOfOrders  TotalQuantity  AveragePrice  PriceStatus
------------  -----------------  --------------  -------------  ------------  -----------
Creams        Nikolovska Marina  1               2.0            998.0         Expensive
Creams        Lazarova Nina      1               3.0            2247.0        Expensive
Lipsticks     Krstevski Ivica    1               3.0            897.0         Expensive
Lipsticks     Avramoska Tanja    1               2.0            438.0         Affordable
NailPolish    Trpenoska Milica   1               6.0            1194.0        Expensive
NailPolish    Naumova Nevena     1               2.0            398.0         Affordable
Perfumes      Prlickov Nenad     1               4.0            4396.0        Expensive
Perfumes      Naumova Nevena     1               1.0            1069.0        Expensive
```

[11] AveragePrice is calculated by using AVG() but it seems like assistant forgot or messed up, so i had to change it like the code you see to get the test passed, be aware of such stuff in exams, analyze the tests