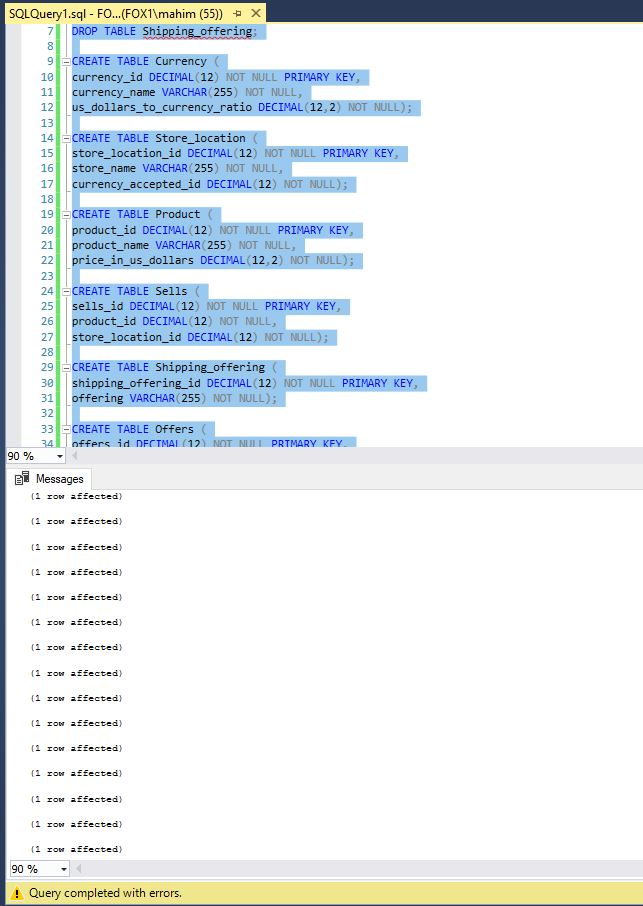
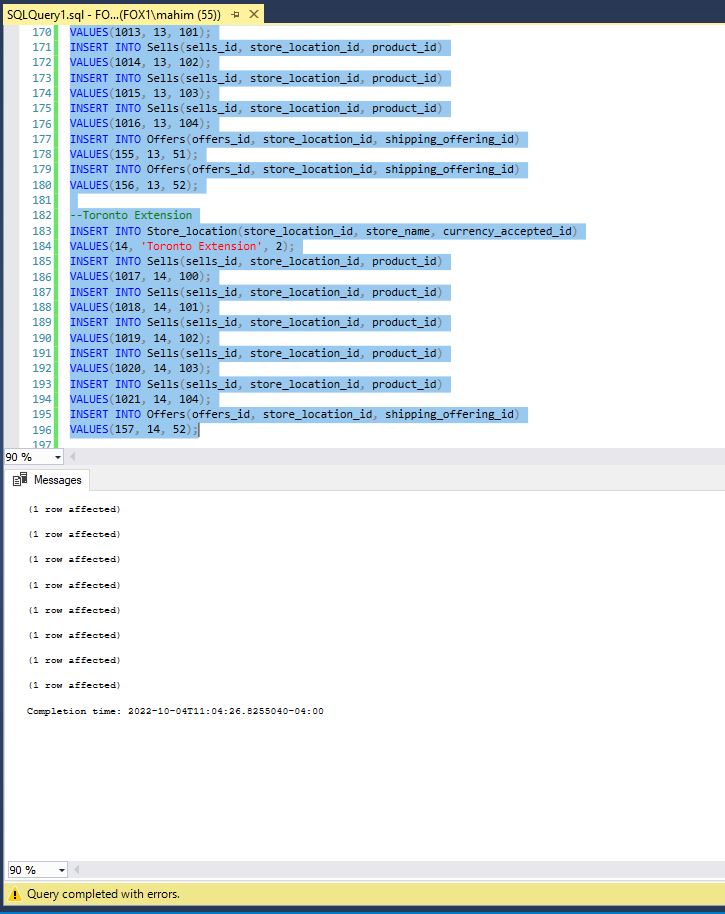
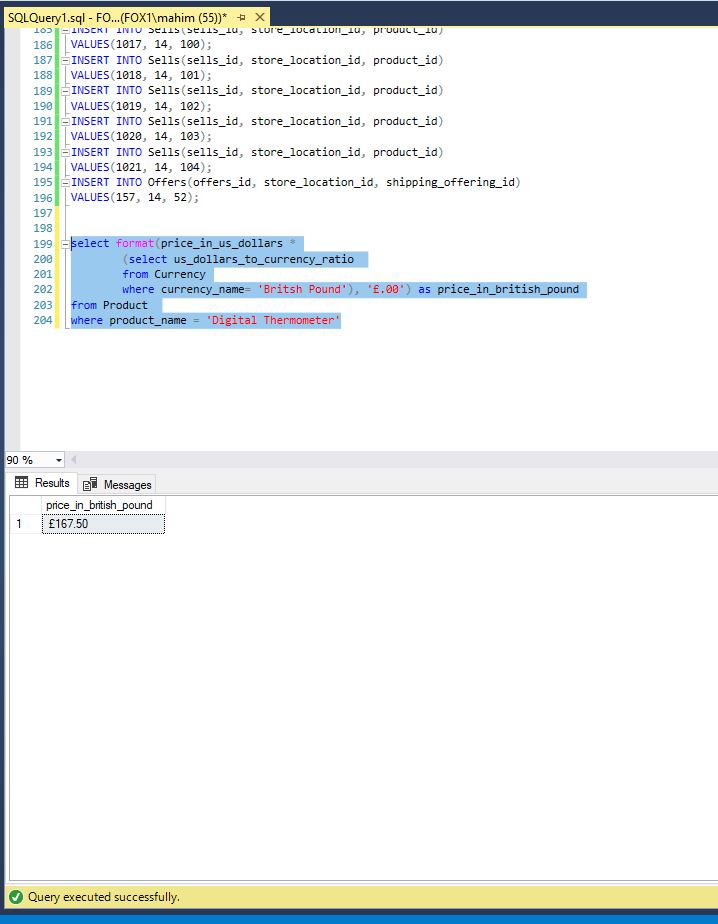
1. *Create Table Structure –* Create the tables in the schema, including all of their columns, datatypes, and constraints, and populate the tables with data. You can do so by executing the DDL and DML above in your SQL client. You only need to capture one or two demonstrative screenshots for this step. No need to screenshot execution of every line of code (that could require dozens of screenshots).

**

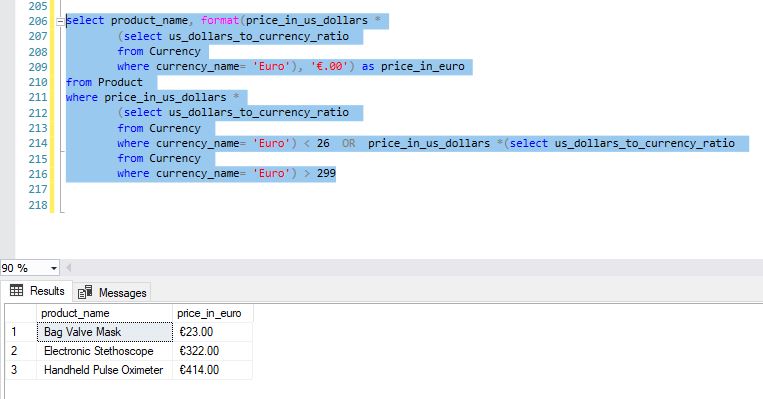


1. *Subquery in Column List –* Write a query that retrieves the price of a digital thermometer in London. A subquery will retrieve the currency ratio for the currency accepted in London. The outer query will use the results of the subquery (the currency ratio) in order to determine the price of the thermometer. The subquery should retrieve dynamic results by looking up the currency the store location accepts, not by hardcoding a specific value. Briefly explain how your solution makes use of the uncorrelated subquery to help retrieve the result.



The usage of subquery reduced the number of separate queries to perform the same task and mitigated the mistakes while performing the desired task. Because the Inner query doesn’t depend on the Product we are interested in, even if the price changes, this subquery query will work because it only produces the ratio result. Here, even if we don’t execute the outer query, the subquery will execute independently.

1. *Subquery in WHERE Clause –* Imagine a charity in London is hosting a fundraiser to purchase medical supplies for organizations that provide care to people in impoverished areas. The charity is targeting both people with average income as well a few wealthier people, and to this end asks for a selection of products both groups can contribute to purchase. Specifically, for the average income group, they would like to know what products cost less than 26 Euros, and for the wealthier group, they would like to know what products cost more than 299 Euros.  
     
   a. Develop a single query to provide them this result, which should contain uncorrelated subqueries and should list the names of the products as well as their prices in Euros.

**  
b. Explain how what each subquery does, its role in the overall query, and how the subqueries were integrated to give the correct results.  
Here, we used one subquery to produce the ratio of price in euros and then used the SQL filter capability to look into the Product table and find the products that are less than 26 euros or more than 299 euros. To find these, we used two subqueries that had the appropriate condition for the results. We then used those subqueries results to list the product names and prices in a table which can serve the purpose of this step.

Note that the Euro monetary prefix is €.

1. *Using the IN Clause with a Subquery* – Imagine that Esther is a traveling doctor who works for an agency that sends her to various locations throughout the world with very little notice. As a result, she needs to know about medical supplies *that are available in all store locations (not just some locations)*. This way, regardless of where she is sent, she knows she can purchase those products. She is also interested in viewing the alternate names for these products, so she is absolutely certain what each product is.  
     
   Note: It is important to Esther that she can purchase the product in any location; only products sold in all stores should be listed, that is, if a product is sold in some stores, but not all stores, it should not be listed.   
     
   a. Develop a single query to list out these results, making sure to use uncorrelated subqueries where needed (one subquery will be put into the WHERE clause of the outer query).

Graphical user interface, text, application

Description automatically generated  
b. Explain how what each subquery does, its role in the overall query, and how the subqueries were integrated to give the correct results.  
This could be done multiple ways but first I selected the product name, alternate\_name, and store location, and then I used the IN clause to filter out the stores which sell all 4 possible medical products. I used the subquery to select the store locations through its ID and using the aggregation of product id which made sure only to show the results when the count is above 3 since all the other locations sell 3 or more products.

1. *Subquery in FROM Clause –* For this problem you will write a single query to address the same use case as in step 4, but change your query so that the main uncorrelated subquery is in the FROM clause rather than in the WHERE clause. The results should be the same as in step 4, except of course possibly row ordering which can vary. Explain how you integrated the subquery into the FROM clause to derive the same results as step 4.

Explanation: Replacing the WHERE clause in the outer query with FROM gave me the same result as step 4(screenshot below). I just moved the Join clauses at the end and replaced the IN clause with FROM. I created an alias called ‘locations’ and used it as the result of this subquery. Notice the store name had to be included inside the subquery as part of the change.

Graphical user interface, text, application

Description automatically generated

1. *Correlated Subquery –* For this problem you will write a single query to address the same use case as in step 4, but change your query to use a *correlated* query combined with an EXISTS clause. The results should be the same as in step 4, except of course possibly row ordering which can vary. Explain:  
     
   a. how your solution makes use of the correlated subquery and EXISTS clause to help retrieve the result

The use of a correlated subquery means the inner subquery is dependent on a table from the outer query. An EXISTS clause will show everything in the table if it sees the user condition met once, meaning it sees a row exists off the condition, it will display everything in that table that was selected. I used an alias named ‘Desired\_store\_location’ that distinguishes the inner subquery from the outer one which was set to equal towards the end of the query. Within the EXISTS clause, I used that alias location to filter out the locations with more than three products and avoided the EXISTS clause problem of providing unnecessary result. I then set the alias to the actual store location id for SQL to understand it was an alias of Store\_location\_id.

Graphical user interface, application

Description automatically generated

b. how and when the correlated subquery is executed in the context of the outer query.

From the lab explanation document, I have gathered the knowledge,

A correlated subquery executes once for each row at a time and thus it shows result sets per row. Since correlated subqueries are not independent unlike uncorrelated subqueries, they depend on the execution of each row of the outer query.

*Using View in Query –* For this problem you will write a query to address the same use case as in step 4, except you will create and use a *view* in the FROM clause in place of the subquery. The results should be the same as in step 4, except of course possibly row ordering which can vary.

1. *Using View in Query –* For this problem you will write a query to address the same use case as in step 4, except you will create and use a *view* in the FROM clause in place of the subquery. The results should be the same as in step 4, except of course possibly row ordering which can vary.

Graphical user interface, application

Description automatically generated

1. *Issues with No Concurrency Control –* Imagine the transactions for this section are presented to a modern relational database at the same time, and the database does *not* have concurrency control mechanisms in place. Show a step-by-step schedule that results in a lost update, inconsistent analysis, or uncommitted dependency. Also list out the contents of the table after the transactions complete using the schedule. You only need to show a schedule for one of the issues, not all three. You are not creating this table in SQL, so it is fine to show the table in Excel, Word, or another comparable application.

|  |
| --- |
| 1. **Data Table** |
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |

|  |  |
| --- | --- |
| **Transaction 1** | **Transaction 2** |
| Read the value from row 4. | Read the value from row 2. |
| Multiply that value times 3. | Write that value to row 4. |
| Write the result to row 3. | Write the literal value “15” to row 3. |
| Write the literal value “8” to row 2. |  |
| Write the literal value “20” to row 5. |  |
| Commit. |  |

There can be many variations of scheduling, but I have chosen the following,

|  |  |  |
| --- | --- | --- |
| Time | Schedule1 | Schedule2 |
| 1 | T1: Read the value from row 4. | T1: Read the value from row 4. |
| 2 | T2: Read the value from row 2. | T1: Multiply that value times 3. |
| 3 | T1: Multiply that value times 3. | T2: Read the value from row 2. |
| 4 | T2: Write that value to row 4. | T2: Write that value to row 4. |
| 5 | T1: Write the result to row 3. | T1: Write the result to row 3. |
| 6 | T2: Write the literal value “15” to row 3. | T1: Write the literal value “8” to row 2. |
| 7 | T1: Write the literal value “8” to row 2. | T1: Commit |
| 8 | T1: Write the literal value “20” to row 5. | T1: Write the literal value “20” to row 5. |
| 9 | T1: Commit | T2: Write the literal value “15” to row 3. |

From Schedule 1, the lost update happened because Transaction 2 did not know that transaction 1 has already updated the values of table 1. When T1 wrote (3\*4) =12 in row 4, T2 wrote 2 in row 4 immediately. Therefore, the update for row 4 did not take place as expected. Also, T1 wrote 12 in row 3, then T2 wrote 15 in row 3 immediately and so the update was lost.

|  |  |  |  |
| --- | --- | --- | --- |
| Time | Transaction | Step | Result |
| 1 | T1 | Read the value from row 4. | Row 4 = 4 |
| 2 | T2 | Read the value from row 2. | Row 2 = 2 |
| 3 | T1 | Multiply that value times 3. | Row 4 =12 (3\*4) |
| 4 | T2 | Write that value to row 4. | Row 4 = 2 |
| 5 | T1 | Write the result to row 3. | Row 3 = 12 |
| 6 | T2 | Write the literal value “15” to row 3. | Row 3 = 15 |
| 7 | T1 | Write the literal value “8” to row 2. | Row 2 = 8 |
| 8 | T1 | Write the literal value “20” to row 5. | Row 5 = 20 |

|  |
| --- |
| **Data Table** |
| 1 |
| 2 -> 8 |
| 3 -> 15 |
| 4 -> 2 |
| 5 - > 20 |

1. *Issues with Locking and Multiversioning –* Imagine the database has both locking and multiversioning in place for concurrency control.
2. Starting with the same schedule in the prior step, show and explain step-by-step how the use of locking and multiversioning modifies the schedule. Also list out the contents of the table after the transactions complete using the new schedule. Make sure to explain specifically whether and how locking and multiversioning modifies the schedule and affects the final resulting table.

Schedule,

|  |  |  |
| --- | --- | --- |
| Time | Schedule1 | Schedule2 |
| 1 | T1: Read the value from row 4. | T1: Read the value from row 4. |
| 2 | T2: Read the value from row 2. | T1: Multiply that value times 3. |
| 3 | T1: Multiply that value times 3. | T1: Write the literal value “20” to row 5. |
| 4 | T2: Write that value to row 4. | T2: Write that value to row 4. |
| 5 | T1: Write the result to row 3. | T1: Write the result to row 3. |
| 6 | T2: Write the literal value “15” to row 3. | T1: Write the literal value “8” to row 2. |
| 7 | T1: Write the literal value “8” to row 2. | T1: Commit |
| 8 | T1: Write the literal value “20” to row 5. | T2: Read the value from row 2. |
| 9 | T1: Commit | T2: Write the literal value “15” to row 3. |

Let’s analyze the 2nd schedule from the scheduling table for this task. Shared locking can first allow the two transactions to read the values of row 4 and row 2, then, when T1 writes 3\*4 = 12 in row 4, exclusive locking will lock it and only allow T2 to modify this row after T1 has committed. Same for other rows as well.

.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Time | Transaction | Step | Result | Explanation |
| 1 | T1 | Read the value from row 4. | Row 4 = 4 | (Shared locking T1) |
| 2 | T1 | Multiply that value times 3. | Row 4 = 12 | Exclusive lock for T1 |
| 3 | T1 | Write the literal value “20” to row 5. | Row 5 = 20 | Exclusive lock for T1 |
| 4 | T2 | Write that value to row 4. | ~~Row 4 = 20~~ | An exclusive lock of T1 will prevent this execution until T1 commits |
| 5 | T1 | Write the result to row 3. | Row 3 = 20 | Exclusive lock for T1 (Result from Step 3) |
| 6 | T1 | Write the literal value “8” to row 2. | Row 2 = 8 | Exclusive lock for T1 |
| 7 | T1 | Commit |  | T1 locks is now unlocked |
| 8 | T2 | Read the value from row 2. | Row 2 = 8 | Shared locking T2. Now reads 8 because T1 has commited |
| 9 | T2 | Write the literal value “15” to row 3. | Row 3 = 15 | Exclusive lock for T2. But T2 has not committed though. |

After T1 Commits, the table should look like this,

|  |
| --- |
| **Data Table** |
| 1 |
| 8 |
| 20 |
| 12 |
| 20 |

The results clearly indicated the effect of locking in one of the scheduling.

Now for Multiversioning lets analyze the same schedule, even though the scheduling doesn’t have many useful examples for this purpose. It basically makes sure if one row has a value and two transactions change the value, it makes sure they get to work with the original value of that row instead of the changed value by a transaction.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Time | Transaction | Step | Result | Explanation |
| 1 | T1 | Read the value from row 4. | Row 4 = 4 | Original value of row 4 is 4 |
| 2 | T1 | Multiply that value times 3. | Row 4 = 12 | Now it’s 12 for T1 |
| 3 | T1 | Write the literal value “20” to row 5. | Row 5 = 20 | Row 5 is 20 for T1 |
| 4 | T2 | Write that value to row 4. | Row 4 = 20 | Sets the row 4 to 20 but remembers the original value as well |
| 5 | T1 | Write the result to row 3. | Row 3 = 20 | Sets 20 for row 3 |
| 6 | T1 | Write the literal value “8” to row 2. | Row 2 = 8 | Sets row 2 value to be 8 |
| 7 | T1 | Commit |  | T1 locks is now unlocked |
| 8 | T2 | Read the value from row 2. | Row 2 = 8 | Reads the original value 8 |
| 9 | T2 | Write the literal value “15” to row 3. | Row 3 = 15 | Sets row 3 to 15 |

1. Could a schedule of these transactions result in a deadlock? If not, explain why. If so, show a step-by-step schedule that results in a deadlock.

I don’t think there is a deadlock situation here because both transactions are locking the same row same time and they are not dependent on each other’s lock because one must finish before the other.