|  |
| --- |
| **Overview of the Lab** |
| In this lab we learn to work with subqueries, which significantly extend the expressional power of queries. Through the use of subqueries, a single query can extract result sets that could not be extracted without subqueries. Subqueries enable the query creator to ask the database for many complex structures in a single query. This lab teaches you the mechanics crafting SQL queries that harness the power of subqueries to handle more complex use cases.  From a technical perspective, together, we will learn:   * what correlated and uncorrelated subqueries are and the theory supporting both. * to use subqueries that return a single value, a list of values, and a table of values. * to use subqueries that use aggregation. * to address use cases by using uncorrelated subqueries in the column select list, the where clause, and the from clause. * to address use cases by using correlated subqueries and an EXIST clause in the WHERE clause. * how transaction schedules, locks, and multiversioning works with transaction concurrency. |

|  |
| --- |
| **Lab 5 Explanations Reminder** |
| As a reminder, it is important to read through the Lab 5 Explanation document to successfully complete this lab, available in the assignment inbox alongside this lab. The explanation document illustrates how to correctly execute each SQL construct step-by-step, and explains important theoretical and practical details. |

|  |
| --- |
| **Other Reminders** |
| * The examples in this lab will execute in modern versions of Oracle, Microsoft SQL Server, and PostgreSQL as is. * The screenshots in this lab display execution of SQL in the default SQL clients supported in the course – Oracle SQL Developer, SQL Server Management Studio, and pgAdmin – but your screenshots may vary somewhat as different version of these clients are released. * Don’t forget to commit your changes if you work on the lab in different sittings, using the “COMMIT” command, so that you do not lose your work. |

**Section One – Subqueries**

**Section Background**

In this section, you will practice crafting subqueries for the schema illustrated below.



This schema’s structure supports basic medical product and currency information for an international medical supplier, including store locations, the products they sell, shipping offerings, the currency each location accepts, as well as conversion factors for converting from U.S. dollars into the accepted currency. Due to the specific and technical nature of the names of medical products, the supplier also keeps a list of alternative names for each product that may help customers identify them. This schema models prices and exchange rates at a specific point in time. While a real-world schema would make provision for changes to prices and exchange rates over time, the tables needed to support this have been intentionally excluded from our schema, because their addition would add unneeded complexity on your journey of learning subqueries, expressions, and value manipulation. The schema has just the right amount of complexity for your learning.

The data for the tables is listed below.

Currencies

|  |  |
| --- | --- |
| **Name** | **Ratio** |
| British Pound | 0.67 |
| Canadian Dollar | 1.34 |
| US Dollar | 1.00 |
| Euro | 0.92 |
| Mexican Peso | 16.76 |

Store Locations

|  |  |
| --- | --- |
| **Name** | **Currency** |
| Berlin Extension | Euro |
| Cancun Extension | Mexican Peso |
| London Extension | British Pound |
| New York Extension | US Dollar |
| Toronto Extension | Canadian Dollar |

Product

|  |  |
| --- | --- |
| **Name** | **US Dollar Price** |
| Glucometer | $50 |
| Bag Valve Mask | $25 |
| Digital Thermometer | $250 |
| Electronic Stethoscope | $350 |
| Handheld Pulse Oximeter | $450 |

Sells

|  |  |
| --- | --- |
| **Store Location** | **Product** |
| Berlin Extension | Glucometer |
| Berlin Extension | Bag Valve Mask |
| Berlin Extension | Digital Thermometer |
| Berlin Extension | Handheld Pulse Oximeter |
| Cancun Extension | Bag Valve Mask |
| Cancun Extension | Digital Thermometer |
| Cancun Extension | Handheld Pulse Oximeter |
| London Extension | Glucometer |
| London Extension | Bag Valve Mask |
| London Extension | Digital Thermometer |
| London Extension | Electronic Stethoscope |
| London Extension | Handheld Pulse Oximeter |
| New York Extension | Glucometer |
| New York Extension | Bag Valve Mask |
| New York Extension | Digital Thermometer |
| New York Extension | Electronic Stethoscope |
| New York Extension | Handheld Pulse Oximeter |
| Toronto Extension | Glucometer |
| Toronto Extension | Bag Valve Mask |
| Toronto Extension | Digital Thermometer |
| Toronto Extension | Electronic Stethoscope |
| Toronto Extension | Handheld Pulse Oximeter |

Shipping\_offering

|  |
| --- |
| **Offering** |
| Same Day |
| Overnight |
| Two Day |

Offers

|  |  |
| --- | --- |
| **Store Location** | **Shipping Offering** |
| Berlin Extension | Two Day |
| Cancun Extension | Two Day |
| London Extension | Same Day |
| London Extension | Overnight |
| London Extension | Two Day |
| New York Extension | Overnight |
| New York Extension | Two Day |
| Toronto Extension | Two Day |

Alternate Names

|  |  |
| --- | --- |
| **Name** | **Product** |
| Glucose Meter | Glucometer |
| Blood Glucose Meter | Glucometer |
| Glucose Monitoring System | Glucometer |
| Thermometer | Digital Thermometer |
| Ambu Bag | Bag Valve Mask |
| Oxygen Bag Valve Mask | Oxygen Bag Valve Mask |
| Cardiology Stethoscope | Electronic Stethoscope |
| Portable Pulse Oximeter | Handheld Pulse Oximeter |
| Handheld Pulse Oximeter System | Handheld Pulse Oximeter |

The DDL and DML to create and populate the tables in the schema are listed below. You can copy and paste this into your SQL client to create and populate the tables.

DROP TABLE Sells;

DROP TABLE Offers;

DROP TABLE Store\_location;

DROP TABLE Alternate\_name;

DROP TABLE Product;

DROP TABLE Currency;

DROP TABLE Shipping\_offering;

CREATE TABLE Currency (

currency\_id DECIMAL(12) NOT NULL PRIMARY KEY,

currency\_name VARCHAR(255) NOT NULL,

us\_dollars\_to\_currency\_ratio DECIMAL(12,2) NOT NULL);

CREATE TABLE Store\_location (

store\_location\_id DECIMAL(12) NOT NULL PRIMARY KEY,

store\_name VARCHAR(255) NOT NULL,

currency\_accepted\_id DECIMAL(12) NOT NULL);

CREATE TABLE Product (

product\_id DECIMAL(12) NOT NULL PRIMARY KEY,

product\_name VARCHAR(255) NOT NULL,

price\_in\_us\_dollars DECIMAL(12,2) NOT NULL);

CREATE TABLE Sells (

sells\_id DECIMAL(12) NOT NULL PRIMARY KEY,

product\_id DECIMAL(12) NOT NULL,

store\_location\_id DECIMAL(12) NOT NULL);

CREATE TABLE Shipping\_offering (

shipping\_offering\_id DECIMAL(12) NOT NULL PRIMARY KEY,

offering VARCHAR(255) NOT NULL);

CREATE TABLE Offers (

offers\_id DECIMAL(12) NOT NULL PRIMARY KEY,

store\_location\_id DECIMAL(12) NOT NULL,

shipping\_offering\_id DECIMAL(12) NOT NULL);

CREATE TABLE Alternate\_name (

alternate\_name\_id DECIMAL(12) NOT NULL PRIMARY KEY,

name VARCHAR(255) NOT NULL,

product\_id DECIMAL(12) NOT NULL);

ALTER TABLE Store\_location

ADD CONSTRAINT fk\_location\_to\_currency FOREIGN KEY(currency\_accepted\_id) REFERENCES Currency(currency\_id);

ALTER TABLE Sells

ADD CONSTRAINT fk\_sells\_to\_product FOREIGN KEY(product\_id) REFERENCES Product(product\_id);

ALTER TABLE Sells

ADD CONSTRAINT fk\_sells\_to\_location FOREIGN KEY(store\_location\_id) REFERENCES Store\_location(store\_location\_id);

ALTER TABLE Offers

ADD CONSTRAINT fk\_offers\_to\_location FOREIGN KEY(store\_location\_id) REFERENCES Store\_location(store\_location\_id);

ALTER TABLE Offers

ADD CONSTRAINT fk\_offers\_to\_offering FOREIGN KEY(shipping\_offering\_id)

REFERENCES Shipping\_offering(shipping\_offering\_id);

ALTER TABLE Alternate\_name

ADD CONSTRAINT fk\_name\_to\_product FOREIGN KEY(product\_id)

REFERENCES Product(product\_id);

INSERT INTO Currency(currency\_id, currency\_name, us\_dollars\_to\_currency\_ratio)

VALUES(1, 'Britsh Pound', 0.67);

INSERT INTO Currency(currency\_id, currency\_name, us\_dollars\_to\_currency\_ratio)

VALUES(2, 'Canadian Dollar', 1.34);

INSERT INTO Currency(currency\_id, currency\_name, us\_dollars\_to\_currency\_ratio)

VALUES(3, 'US Dollar', 1.00);

INSERT INTO Currency(currency\_id, currency\_name, us\_dollars\_to\_currency\_ratio)

VALUES(4, 'Euro', 0.92);

INSERT INTO Currency(currency\_id, currency\_name, us\_dollars\_to\_currency\_ratio)

VALUES(5, 'Mexican Peso', 16.76);

INSERT INTO Shipping\_offering(shipping\_offering\_id, offering)

VALUES (50, 'Same Day');

INSERT INTO Shipping\_offering(shipping\_offering\_id, offering)

VALUES (51, 'Overnight');

INSERT INTO Shipping\_offering(shipping\_offering\_id, offering)

VALUES (52, 'Two Day');

--Glucometer

INSERT INTO Product(product\_id, product\_name, price\_in\_us\_dollars)

VALUES(100, 'Glucometer', 50);

INSERT INTO Alternate\_name(alternate\_name\_id, name, product\_id)

VALUES(10000, 'Glucose Meter', 100);

INSERT INTO Alternate\_name(alternate\_name\_id, name, product\_id)

VALUES(10001, 'Blood Glucose Meter', 100);

INSERT INTO Alternate\_name(alternate\_name\_id, name, product\_id)

VALUES(10002, 'Glucose Monitoring System', 100);

--Bag Valve Mask

INSERT INTO Product(product\_id, product\_name, price\_in\_us\_dollars)

VALUES(101, 'Bag Valve Mask', 25);

INSERT INTO Alternate\_name(alternate\_name\_id, name, product\_id)

VALUES(10003, 'Ambu Bag', 101);

INSERT INTO Alternate\_name(alternate\_name\_id, name, product\_id)

VALUES(10004, 'Oxygen Bag Valve Mask', 101);

--Digital Thermometer

INSERT INTO Product(product\_id, product\_name, price\_in\_us\_dollars)

VALUES(102, 'Digital Thermometer', 250);

INSERT INTO Alternate\_name(alternate\_name\_id, name, product\_id)

VALUES(10005, 'Thermometer', 102);

--Electronic Stethoscope

INSERT INTO Product(product\_id, product\_name, price\_in\_us\_dollars)

VALUES(103, 'Electronic Stethoscope', 350);

INSERT INTO Alternate\_name(alternate\_name\_id, name, product\_id)

VALUES(10006, 'Cardiology Stethoscope', 103);

--Handheld Pulse Oximeter

INSERT INTO Product(product\_id, product\_name, price\_in\_us\_dollars)

VALUES(104, 'Handheld Pulse Oximeter', 450);

INSERT INTO Alternate\_name(alternate\_name\_id, name, product\_id)

VALUES(10007, 'Portable Pulse Oximeter', 104);

INSERT INTO Alternate\_name(alternate\_name\_id, name, product\_id)

VALUES(10008, 'Handheld Pulse Oximeter System', 104);

--Berlin Extension

INSERT INTO Store\_location(store\_location\_id, store\_name, currency\_accepted\_id)

VALUES(10, 'Berlin Extension', 4);

INSERT INTO Sells(sells\_id, store\_location\_id, product\_id)

VALUES(1000, 10, 100);

INSERT INTO Sells(sells\_id, store\_location\_id, product\_id)

VALUES(1001, 10, 101);

INSERT INTO Sells(sells\_id, store\_location\_id, product\_id)

VALUES(1002, 10, 102);

INSERT INTO Sells(sells\_id, store\_location\_id, product\_id)

VALUES(1003, 10, 104);

INSERT INTO Offers(offers\_id, store\_location\_id, shipping\_offering\_id)

VALUES(150, 10, 52);

--Cancun Extension

INSERT INTO Store\_location(store\_location\_id, store\_name, currency\_accepted\_id)

VALUES(11, 'Cancun Extension', 5);

INSERT INTO Sells(sells\_id, store\_location\_id, product\_id)

VALUES(1004, 11, 101);

INSERT INTO Sells(sells\_id, store\_location\_id, product\_id)

VALUES(1005, 11, 102);

INSERT INTO Sells(sells\_id, store\_location\_id, product\_id)

VALUES(1006, 11, 104);

INSERT INTO Offers(offers\_id, store\_location\_id, shipping\_offering\_id)

VALUES(151, 11, 52);

--London Extension

INSERT INTO Store\_location(store\_location\_id, store\_name, currency\_accepted\_id)

VALUES(12, 'London Extension', 1);

INSERT INTO Sells(sells\_id, store\_location\_id, product\_id)

VALUES(1007, 12, 100);

INSERT INTO Sells(sells\_id, store\_location\_id, product\_id)

VALUES(1008, 12, 101);

INSERT INTO Sells(sells\_id, store\_location\_id, product\_id)

VALUES(1009, 12, 102);

INSERT INTO Sells(sells\_id, store\_location\_id, product\_id)

VALUES(1010, 12, 103);

INSERT INTO Sells(sells\_id, store\_location\_id, product\_id)

VALUES(1011, 12, 104);

INSERT INTO Offers(offers\_id, store\_location\_id, shipping\_offering\_id)

VALUES(152, 12, 50);

INSERT INTO Offers(offers\_id, store\_location\_id, shipping\_offering\_id)

VALUES(153, 12, 51);

INSERT INTO Offers(offers\_id, store\_location\_id, shipping\_offering\_id)

VALUES(154, 12, 52);

--New York Extension

INSERT INTO Store\_location(store\_location\_id, store\_name, currency\_accepted\_id)

VALUES(13, 'New York Extension', 3);

INSERT INTO Sells(sells\_id, store\_location\_id, product\_id)

VALUES(1012, 13, 100);

INSERT INTO Sells(sells\_id, store\_location\_id, product\_id)

VALUES(1013, 13, 101);

INSERT INTO Sells(sells\_id, store\_location\_id, product\_id)

VALUES(1014, 13, 102);

INSERT INTO Sells(sells\_id, store\_location\_id, product\_id)

VALUES(1015, 13, 103);

INSERT INTO Sells(sells\_id, store\_location\_id, product\_id)

VALUES(1016, 13, 104);

INSERT INTO Offers(offers\_id, store\_location\_id, shipping\_offering\_id)

VALUES(155, 13, 51);

INSERT INTO Offers(offers\_id, store\_location\_id, shipping\_offering\_id)

VALUES(156, 13, 52);

--Toronto Extension

INSERT INTO Store\_location(store\_location\_id, store\_name, currency\_accepted\_id)

VALUES(14, 'Toronto Extension', 2);

INSERT INTO Sells(sells\_id, store\_location\_id, product\_id)

VALUES(1017, 14, 100);

INSERT INTO Sells(sells\_id, store\_location\_id, product\_id)

VALUES(1018, 14, 101);

INSERT INTO Sells(sells\_id, store\_location\_id, product\_id)

VALUES(1019, 14, 102);

INSERT INTO Sells(sells\_id, store\_location\_id, product\_id)

VALUES(1020, 14, 103);

INSERT INTO Sells(sells\_id, store\_location\_id, product\_id)

VALUES(1021, 14, 104);

INSERT INTO Offers(offers\_id, store\_location\_id, shipping\_offering\_id)

VALUES(157, 14, 52);

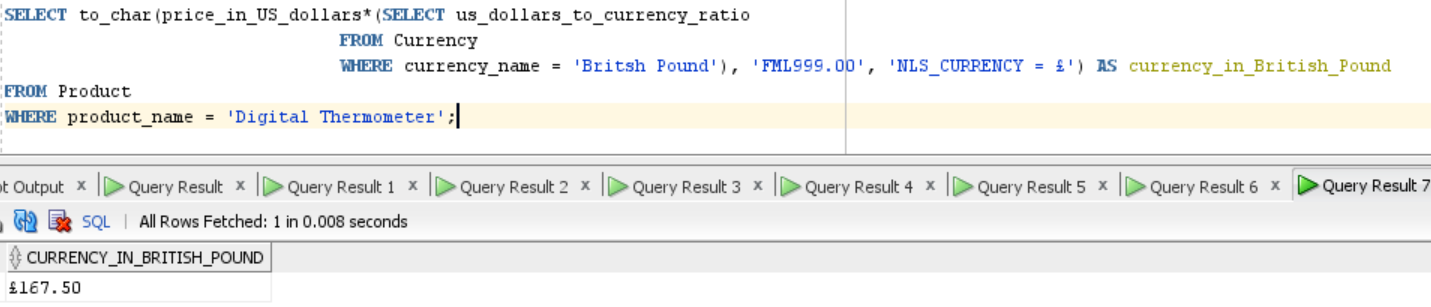
As a reminder, for each step that requires SQL, make sure to capture a screenshot of the command and the results of its execution. *Further, make sure to eliminate unneeded columns from the result set, to name your columns something user-friendly and human readable, and to format any prices as currencies.*

**Section Steps**

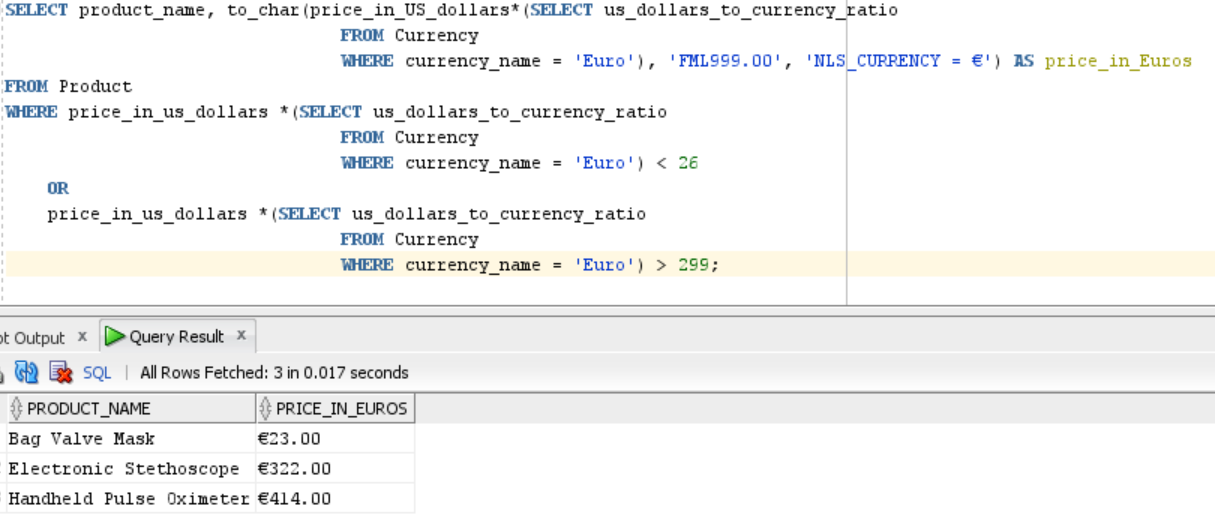
1. *Create Table Structure –* Create the tables in the schema, including all of their columns, datatypes, and constraints, and populate the tables with data. You can do so by executing the DDL and DML above in your SQL client. You only need to capture one or two demonstrative screenshots for this step. 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.



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.

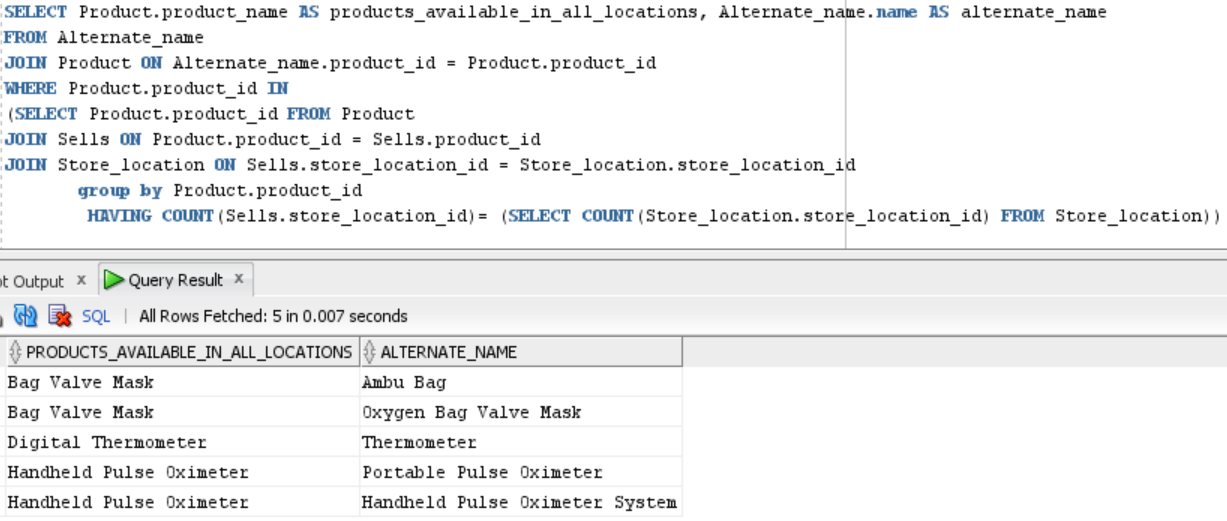


b. Explain how what each subquery does, its role in the overall query, and how the subqueries were integrated to give the correct results.  
Note that the Euro monetary prefix is €.

The query in the SELECT clause retrieves the currency ratio of Euro from the Currency table and the outer query takes this ratio and multiplies it with the price in the US dollars from the Product table, which provides the converted currency price, and formats it with Euro currency symbol. ‘FML999.00’ tells the SQL engine to display the necessary digits, and there may be up to 3 digits to the left of the decimal point and 2 digits after the decimal point.

The query in the WHERE clause only performs the currency conversion (USD to Euro) and retrieves rows from the Currency table where the converted price is less than 26 Euros and the rows where the converted price is more than 299 Euros. The other rows that do not match with these conditions are excluded. The subquery in this clause is executed first and its results are tied to the subquery in the SELECT clause and then to the SELECT statement.

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).

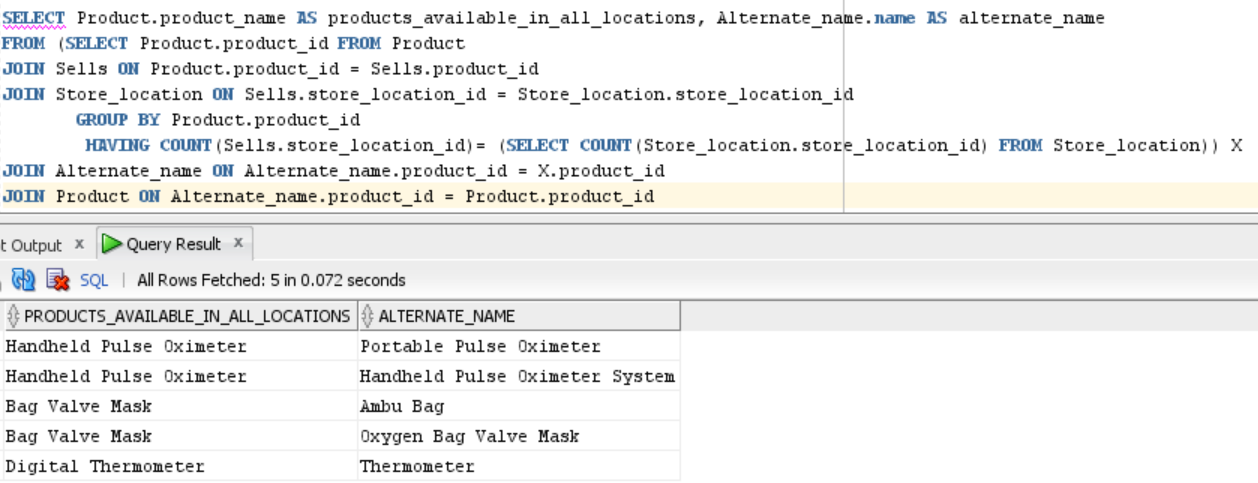


b. Explain how what each subquery does, its role in the overall query, and how the subqueries were integrated to give the correct results.  
The innermost query returns the number of stores in the Store table, which is 5. Since the number of stores may increase or decrease over time, this query will return the accurate result compared to hardcoding the number of stores. The second innermost query joins the Sells table with the Product and Store\_location tables, groups by product\_id to receive line by line result for the products sold at all locations. The HAVING clause is used to determine the products sold at all locations. Finally, the outer query performs a join operation on Alternate\_name and Product table and in the WHERE clause uses IN to search for product\_id present in the result returned by inner queries. In this case, product\_id IN (101, 102, 104) and then

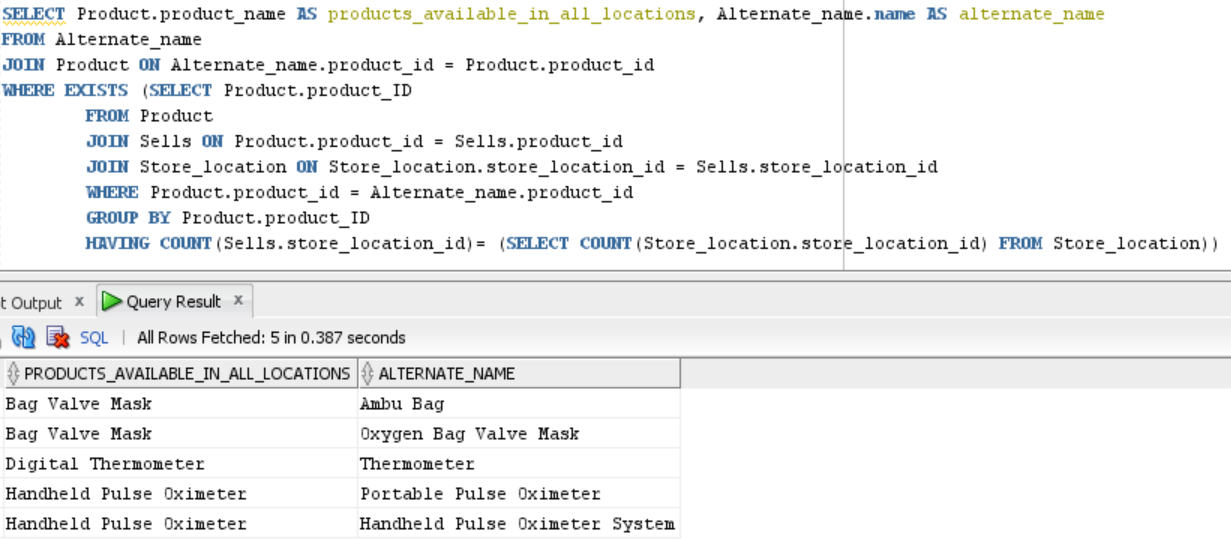
the outer query gathers all the information from the inner queries and executes the SELECT clause to display the alternate names of products with IDs 101, 102, and 104.

In your thinking about how to address this use case, one item should be brought to your attention – the phrase “all store locations”. By eyeballing the data, you can determine the number of locations and hardcode that number, which will satisfy Esther’s request at this present time; however, as the number of locations change over time (with stores opening or closing), such hardcoding would fail. It’s better to dynamically determine the total number of locations in the query itself so that the results are correct over time.

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.



The subquery that was in the WHERE clause has been moved into the FROM clause. The letter “X” after the subquery is the alias for the subquery results. The letter “X” is joins the Alternate\_name table to join the results of the subquery with the Alternate\_name table, which in turn joins the Product table. The query in general provides the same result as in the previous step. The subqueries in the FROM clause are executed first.

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 correlated subqueries are utilized to process each, and each subquery is executed once for each row of the outer query. The EXISTS operator checks if the rows exist in the subquery’s results set. If yes, the condition evaluates to *true* and SQL halts executing the inner query. If not, the condition evaluates to *false* and SQL continues executing the inner query. EXISTS is typically coupled with a correlated subquery to achieve meaningful results.

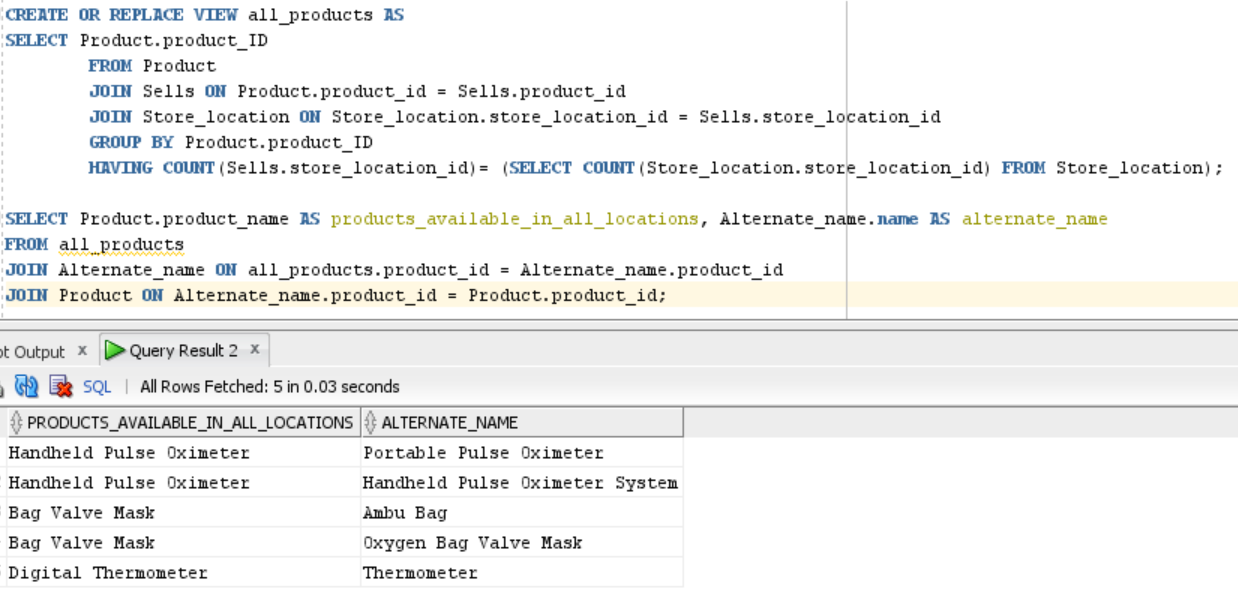
b. how and when the correlated subquery is executed in the context of the outer query.

A correlated subquery references a column name of a table that is different from a table mentioned it its FROM clause. Hence, the subquery is not an independent query as it refers to at least one table in the outer query. The column name of the referenced table is generally provided in the WHERE clause instead of performing a JOIN operation.

Correlated queries are different from the uncorrelated subqueries as they cannot execute on their own and must be tied to an outer query for execution. correlated subqueries are executed once for each row in the outer query and therefore retrieve one result set for each row in the outer query whereas, uncorrelated subqueries are executed once and retrieve results once. Unlike uncorrelated subquery that has fixed results, the results of a correlated subquery are relative to each row in the outer

query.

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.



**Section Two – Concurrency**

**Section Background**

Modern information systems run transactions in parallel. Running hundreds or even thousands of transactions at the same time is commonplace for information systems today. Transactions running at the same run into many issues, including lost updates, uncommitted dependencies, inconsistent analysis, and others. To eliminate and manage these issues, modern relational databases use a scheduler which controls the schedule and timing of transaction execution, in addition to other mechanisms.

You have a chance to demonstrate understanding of concurrency control in this section.

In this section, the questions refer to the following data table, as well the following transactions and steps.

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

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

|  |
| --- |
| **Transaction 2** |
| Read the value from row 2. |
| Write that value to row 4. |
| Write the literal value “15” to row 3. |
| Commit. |

**Section Steps**

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 or Word.

Transactions can run into lost updates, uncommitted dependencies, and inconsistent analysis, and other issues when there is no concurrency control.

Lost update:

A lost update occurs when two transactions running at the same time without concurrency control try to update the same item. If two transactions run serially, the output will be as desired. However, if the transaction steps are interleaved, there are many execution schedules that result in a lost update. Whichever transaction updates last wins; the first update has no effect. The following table demonstrates one such instance.

|  |  |
| --- | --- |
| **Initial Person Table Row** | |
| **Step** | **Explanation** |
| Transaction 1: Select Row. | Transaction 1 reads value 4 |
| Transaction 2: Select Row. | Transaction 2 reads value 2 |
| Transaction 1: Change the value to 12 | Transaction 1 multiplies three times 4 |
| Transaction 1: Write back result to row 3 | The result written back to database is 12 |
| Transaction 2: Change value to 2 | Transaction 2 writes 2 in row 4 |
| Transaction 2: Write back result to row 4 | The result written back to database is 2 |
| Transaction 1: Change the value in row 2 to literal value “8” | Transaction 1 copies literal value “8” in place of number 2 |
| Transaction 1: Write the literal value “8” to row 2. | The result written back to database is literal value “8” |
| Transaction 1: Change the value in row 5 to literal value “20” | Transaction 1 copies literal value “20” in row 5 |
| Transaction 1: Write the literal value “20” to row 5. | The result written back to database is literal value “20” |
| Transaction 2: Change the value in row 3 to literal value “15” | Transaction 2 copies literal value “15” in row 3 |
| Transaction 2: Write the literal value “15” to row 3. | The result written back to database is literal value “15” |

It can be observed that Transaction 1 first writes the value 12 to row 3 and Transaction 2 writes the literal value “15” to row 3. Since Transaction 2 performs the last update, it wins; the first update made by Transaction 1 has no effect.

The contents of the table look like this after the transactions complete.

|  |
| --- |
| Data table |
| 1 |
| ‘8’ |
| ‘15’ |
| 2 |
| ‘20’ |

Uncommitted Dependency

An uncommitted dependency arises when two transactions run at the same time, and one transaction reads and uses a value from the other transaction before it has committed. If the other transaction aborts, all its steps are supposed to be rolled back; however, the first transaction is still working with the modified value and may commit it, violating the ACIDS properties of transactions.

Inconsistent Analysis

Inconsistent Analysis occurs when one transaction updates multiple values, and a second concurrent transaction reads in some of those values before they are changed, and others after they are changed, resulting in an inconsistent view of the data. (CS669\_Lab5Explanation)

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 step-by-step how the use of locking and multiversioning modifies the schedule, and also list out the contents of the table after the transactions complete using the new schedule.

The following table shows how the use of locking modifies the schedule.

|  |  |
| --- | --- |
| **Locking** | |
| **Step** | **Explanation** |
| Transaction 1: Read Row 4. | Transaction 1 now has a shared lock on row 4. |
| Transaction 2: Read Row 2. | Transaction 2 now has a shared lock on row 2. |
| Transaction 1: Update row 3: Change the value to 12 | Transaction 1 now has an exclusive lock on row 3. |
| Transaction 2: Update row 4: Change value to 2 (attempt) | Transaction 2 must wait at this step as it is unable receive an exclusive lock, because Transaction 1 has a shared lock on row 4. |
| Transaction 1: Update Row 2: Change the value in row 2 to literal value “8” (attempt) | Transaction 1 must wait at this step as it is unable receive an exclusive lock, because Transaction 2 has a shared lock on row 2. |
| Transaction 1: Update Row 5: Change the value in row 5 to literal value “20” | Transaction 1 now has an exclusive lock on row 5. |
| Transaction 2: Update Row 3: Change the value in row 3 to literal value “15” | Transaction 2 must wait as Transaction 1 has an exclusive lock on row 3. |

The contents of the table look like this at this stage. However, neither transaction complete with this execution schedule.

|  |
| --- |
| **Data table** |
| 1 |
| 2 |
| 12 |
| 4 |
| ‘20’ |

The two transactions lock the same rows in a different order, so the execution schedule results in a deadlock. Transaction 1 is waiting on Transaction 2 to release its locks, and Transaction 2 is waiting on Transaction 1 to release its locks. Neither can continue, resulting in a deadlock.

|  |  |
| --- | --- |
| **Multiversioning** | |
| **Step** | **Explanation** |
| Transaction 1: Read Row 4. | Transaction 1 reads row 4. |
| Transaction 2: Read Row 2. | Transaction 2 reads row 2. |
| Transaction 1: Update row 3: Change the value to 12. | Transaction 1 now has an exclusive lock on row 3. |
| Transaction 2: Update row 4: Change value to 2. | Transaction 2 now has an exclusive lock on row 4. |
| Transaction 1: Update Row 2: Change the value in row 2 to literal value “8”. | Transaction 1 now has an exclusive lock on row |
| Transaction 1: Update Row 5: Change the value in row 5 to literal value “20”. | Transaction 1 now has an exclusive lock on row 5. |
| Transaction 2: Update Row 3: Change the value in row 3 to literal value “15”. | Transaction 2 must wait as Transaction 1 has an exclusive lock on row 3. |
| Transaction 1: Commit. | Transaction 1 commits and releases all of its locks. |
| Transaction 2: Update Row 3: Change the value in row 3 to literal value “15”. | Transaction 2 now has an exclusive lock on row 3. |
| Transaction 2: Commit. | Transaction 2 commits and releases all of its locks. |

Multiversioning only uses exclusive locks; there is no concept of shared locks for reading a row. In the schedule table above, the two transactions are attempting to update the same row (row 3) in a different order. In this case, Transaction 1 commits after updating row 3, so Transaction 2 gains access to row 3.

The history data table will contain all the original updated values of each row:

|  |  |  |
| --- | --- | --- |
| **Data table** | | |
| 1 |  |  |
| 2 | “8” |  |
| 3 | 12 | “15” |
| 4 |  |  |
| 5 | “20” |  |

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.

The transactions result in a deadlock when locking technique is used, but the transactions do not encounter in a deadlock when multiversioning is used. Unlike locking, multiversioning does not use shared locks for reading a row, multiversioning uses exclusive locks for updating or deleting a row. It is unlikely to result in a deadlock when multiversioning is used.

# Evaluation

Your lab will be reviewed by your facilitator or instructor with the criteria outlined in the table below. Note that the grading process:

* involves the grader assigning an appropriate letter grade to each criterion.
* uses the following letter-to-number grade mapping – A+=100,A=96,A-=92,B+=88,B=85,B-=82,C+=88,C=85,C-=82,D=67,F=0.
* provides an overall grade for the submission based upon the grade and weight assigned to each criterion.
* allows the grader to apply additional deductions or adjustments as appropriate for the submission.
* applies equally to every student in the course.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Criterion** | **A** | **B** | **C** | **D** | **F** |
| **Section 1: Results (20%)** | The results for all steps in Section 1 are complete and accurate. No extra rows or columns are present, and no rows or columns are missing. | The results for most steps in Section 1 are complete and accurate. Few extra rows or columns are present, and few rows or columns are missing. | The results for some steps in Section 1 are accurate. Some extra rows or columns may be present, and some rows or columns may be missing. | The results for most steps in Section 1 are incomplete and inaccurate. Extra rows or columns may be present, and extra rows or columns may be missing. | The results for all of section 1 may be missing. All steps' results are incomplete and inaccurate. |
| **Section 1: Construction (30%)** | Appropriate subqueries have been defined and integrated well for all steps. For all steps possible, dynamic lookups have been used in lieu of hardcoded values. | Appropriate subqueries have been defined and integrated well for most steps. For most steps possible, dynamic lookups have been used in lieu of hardcoded values. | Appropriate subqueries have been defined and integrated well for some steps. For some steps possible, dynamic lookups have been used in lieu of hardcoded values. | Inappropriate subqueries have been defined or integrated poorly for most steps. Hardcoded values have been used for most steps, instead of dynamic lookups. | Answers for all of section 1 may be missing. Inappropriate subqueries have been defined or integrated poorly for all steps. Hardcoded values have been used for all steps, instead of dynamic lookups. |
| **Section 2: #8 Quality (15%)** | The explanation, schedule, and resulting table contents for #8 are entirely accurate, thorough, and clear. | The explanation, schedule, and resulting table contents for #8 are mostly accurate, thorough, and clear. | The explanation, schedule, and resulting table contents for 8 are somewhat accurate and clear. | The explanation, schedule, and resulting table contents for #8 are mostly inaccurate and unclear. | The answers for #8 could be missing. The explanation, schedule, and resulting table contents for #8 are entirely inaccurate and unclear. |
| **Section 2: #9 Quality (15%)** | The explanations, schedules, and resulting table contents for 9a and 9b are entirely accurate, thorough, and clear. | The explanations, schedules, and resulting table contents for 9a and 9b are mostly accurate, thorough, and clear. | The explanations, schedules, and resulting table contents for 9a and 9b are somewhat accurate and clear. | The explanations, schedules, and resulting table contents for 9a and 9b are mostly inaccurate and unclear. | The answer for #9 could be missing. The explanations, schedules, and resulting table contents for 9a and 9b are entirely inaccurate and unclear. |
| **Overall Presentation (20%)** | The explanations supporting all answers are excellent. Queries or other SQL commands have been executed to demonstrate correctness for all answers that need them. | The explanations supporting the answers are good. Queries or other SQL commands have been executed to demonstrate correctness for most answers that need them. | The explanations supporting the answers are satisfactory. Queries or other SQL commands have been executed to demonstrate correctness for some answers that need them. | The explanations supporting the answers are minimal. Queries or other SQL commands have not been executed to demonstrate correctness for most answers that need them. | Explanations for all answers are missing. Queries or other SQL commands have not been executed to demonstrate correctness. |

Use the **Ask the Teaching Team Discussion Forum** if you have any questions regarding how to approach this lab. Make sure to include your name in the filename and submit it in the *Assignments* section of the course.