|  |
| --- |
| **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).

Answer:

1.after copying and pasting and executing the provided SQL scripts, I have taken screen shots for couple of tables which are given below.

a. creating table screenshot:

Graphical user interface, text, application, email

Description automatically generated

b . adding foreign keys to the above created tables:

Graphical user interface, text, application

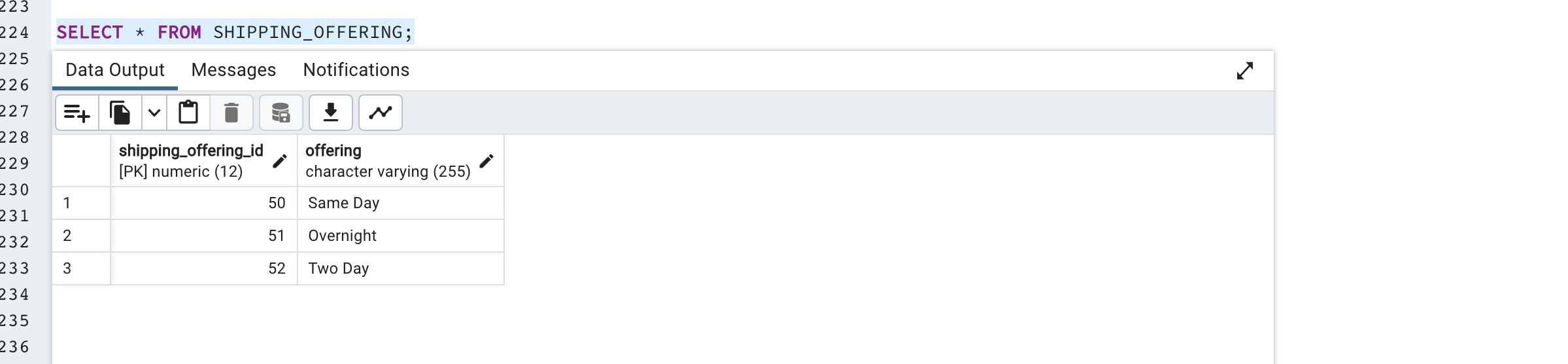
Description automatically generated

c. Populating tables:

Graphical user interface, text, application

Description automatically generated

d. one example of table: Shipping offering table



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.

**Answer:** Here are two queries which are nested together to get the result we wanted. First query gets the dollars to currency ratio value of British pound from currency table and this value is used to convert the us-dollars of values of all products to Pound and then where clause is use to filter out price of product in pound for thermometer only. Since Thermometer is alternative name for Digital Thermometer, I had to join product table to Alternate name table to get thermometer. My SQL query and its result is given below in the snapshot.

Graphical user interface, application

Description automatically generated

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 results, 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 €.
2. Answer

This is the query which gives answer to the question in a.

Graphical user interface, text, application

Description automatically generated

1. **Answer:** So, for this question, First I get the currency ratio for Euro currency from currency table which then multiply to all product prices from product table. Based on this product prices, I select the product which is less than 26 Euros and greater than 299 Euros. Outer query is for to get currency ratio value and then multiply with product prices while subquery in where clause is to filter out the product prices which are less than 26 euros and more than 299 euros.
2. *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.  
     
   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.

**Answer: a**

Snapshot of the query that list the product names with their alternative name which are available in all the locations.

Graphical user interface, application

Description automatically generated

**Answer: b**

So, query in **where clause** finds the product\_id which are found in all the locations. Those products are “Bag Valve Mask”,” Digital Thermometer” and “Handheld Pulse Oximeter”. These products are found all the listed locations. Here, First need to join the product table with store\_location table which can be join by joining another table called Sells. As I must find the product in all the location, having clause filters out those product\_id that are not available in all locations which means total number of locations must equal to the total number of locations in which a specific product available. Outer query is to get product names and their alternative names that are found in all locations by joining with alternative name table.

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.

**Answer:** In this exercise, what I did is, rather than using in where clause to filter the desired rows and columns of product table, I use same subquery in **FROM clause**. Here, new filtered temporary product table named products with product name and product\_id is used to join with alternate\_name tables to get desired product names with alternative names.

Graphical user interface, application, Teams

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.

**Answer:** In this correlated subquery, the subquery (lines 338-346), it is basically giving the table of store\_location\_ids in which all products are available. First two join clauses join all three tables from which all store\_locations are retrieved where products available in some or all locations are retrieved which are then filtered by using group by and having clause. Just filtering is not enough in this case as we also must correlate the subquery to outer query. Otherwise list of products found in some or all location will be retrieved. Here I use where clause in which product\_id and store\_location\_id is used to refer the subquery to outer query (corelated subquery). Exists clause is a Boolean expression that returns only true or false. Only combining two queries does not work with EXISTS clause, it needs more complex construct. Outer query is just regular query like we did in previous exercise. where expected tables are joins. But when we use EXISTS clause with subquery with proper references, it retrieves the product found in the current row of the outer query only if that product is available in all the locations.

Graphical user interface, text, application

Description automatically generated

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

**Answer:** I already wrote little bit about how it is executed in 6a. Correlated subquery unlike other uncorrelated subquery can not be executed by itself it is dependent on outer query to provide overall result. In our case, where clause of subquery corelates the subquery with outer query using product\_id and store\_location\_id. If we remove where clause, it retrieves products that are available in some or all locations, but we wanted only those products that are available in all the locations. Unlike some SQL constructs that work with a single value, a list of values, or a table of values, the EXISTS clause only works with a subquery. EXISTS does not consider the number of columns or the column’s data types retrieved by the subquery, and even does not consider whether any values are null. EXISTS only test the existence of at least one row. It basically tests” is any row retrieved from this subquery?” During execution of an overall query, the result of a correlated subquery is relative to each row in the outer query. subquery basically filter the product that are available in all location and EXISTS check each row of outer query with subquery, if it is true, it retrieves the row, if not it won’t include in the result.

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.

**Answer:** Here, I used similar script but in a different way. Create or replace view make a temporary table which is created, from Product, Sells and store\_location tables with group by and having clause, that has product\_id and product name found in all the locations. This table is used to join the alternate\_name table to find the alternate name for the filtered products. SQL scripts and its result looks like this in snapshot below.

Graphical user interface, application, Teams

Description automatically generated

**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, Word, or another comparable application.

**Answer:** Concurrency control is important DBMS in which several transactions take place. Transactions without concurrency control runs into several serious issues that could be during writing or reading the data in the database. It is like not having traffic lights and stop signs in the busy street. Without the concurrency control transactions may fall into issues like lost update, inconsistent analysis, or uncommitted dependency. Here I will be writing on lost update issue.

**Lost update:** This issue occurs when two or more transactions occurs at the same time without concurrency control. While updating both transactions read the value at a same time, first transaction update to a value but second transaction updates again same value to different value than first transaction. Altogether, final value is from transaction two and update from the first transaction is lost. That is why this issue is call lost update. There are many execution schedules that result in lost update. Below is one example.

Also, let say row 4 has initial value of 7 and row 2 has initial value of 10 for example to demonstrate the concept of lost update.

**Table -1 without concurrency control**

|  |  |
| --- | --- |
| Steps | Explanation |
| Transaction 1: Select Row | Transaction 1 has read the value 7 from row 4 |
| Transaction 2: Select Row | Transaction 2 has read the value 10 from row 2 |
| Transaction 1: update value | Transaction 1 multiply with 3 to value in row 4 and write 21 to row 3 |
| Transaction 1: write value | Transaction 1 write literal value”8” to row 2 |
| Transaction 2: write value | Transaction 2 write value 10 to row 4 |
| Transaction 1: write value | Transaction 1 write literal value “20” to row 5 |
| Transaction 2: write value | Transaction w write literal value “15” to row 3 |

Here in above example, as transaction 2 read value 10 already before transaction 1 write literal value”8 in” row 2. Consequently row 4 has updated with value 10 instead of literal value “8”. Hence there is lost update occurred in row 4 during update without concurrency control. With the concurrency control table should look something like this.

**Table:2 with concurrency control**

|  |  |
| --- | --- |
| Steps | Explanation |
| Transaction 1: Select Row | Transaction 1 has read the value 7 from row 4 |
| Transaction 1: update value | Transaction 1 multiply with 3 and write 21 to row 3 |
| Transaction 1: write value | Transaction 1 write literal value”8” to row 2 |
| Transaction 1: write value | Transaction 1 write literal value “20” to row 5 |
| Transaction 2: Select Row | Transaction 2 read literal value “8” from row 2 |
| Transaction 2: write value | Transaction 2 write literal value “8” to row 4 |
| Transaction 2: write value | Transaction 2 write literal value “15” to row 3 |

If there was concurrency control, result after commit looks something like this in second table but if there is no concurrency control, there are multiple ways an inconsistency may occur and one of them is shown in first table.

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.

**Answer:** Locking and multiversioning is the concurrency control mechanisms that are used by database management system. In **locking system**, if same value is trying to access and modify by two transactions, for example, second transaction must wait for the first transaction to complete. There are shared and exclusive lock system, in shared lock system, two transactions can read the value in same row or same table but cannot write. However, if one transaction needs to modify the value, it must get exclusive lock. And once one transaction has exclusive lock no other transaction can read or write the value in database. **Multiversioning** is advanced concurrency control mechanism where it replaces the shared lock mechanism in which a transaction does not have to wait to read a row value even if another transaction is updating same row. Instead, it uses the history notes of changes in a row. i.e., it will look for a version of update that make sense, which is the time a transaction first started.One example of use of locking and multiversioning is given in a table which is table 2 from above.

**Table:3 -> similar as table 2 but with explanation**

|  |  |
| --- | --- |
| Steps | Explanation |
| Transaction 1: read row 4 | Transaction 1 will have shared lock and no other transaction can write on row 4 |
| Transaction 1: update row 3 | Transaction 1 will have exclusive lock on multiply value in row 4 with 3 and write product value to row 3 |
| Transaction 1: lock release | Transaction 1 once done with row 4 or 3 and releases lock |
| Transaction 1: read row | Transaction 1 get exclusive lock write literal value”8” to row 2. When commit it release lock |
| Transaction 2: read row 2 | Transaction 2 get shared lock to read literal value “8” from row 2 |
| Transaction 2: write row 4 | As lock on row 4 is already release, Transaction 2 get exclusive lock write literal value “8” to row 4 |
| Transaction 1: write row 5 | Transaction 1 get exclusive lock on row 5 write literal value “20” to row 5 |
|  |  |
| Transaction 2: write row 3 | As transaction 1 already released the exclusive lock on row 3, Transaction2 get exclusive lock and write literal value “15” to row 3 |
|  |  |

Here, focus of the concurrency control mechanism is not to allow multiple transactions to write the value in same row which runs into inconsistency issues. Here if there is locking mechanism, the result for row 4 would be literal”8” instead of value 10 as in table number 1. In the above table, multiversioning can be used in a row where transaction get shared lock in which, based on time of update of specific row, another transaction can retrieve the information.

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.

Answer: Yes, this schedule of transaction could result in a deadlock if there is no multiversioning mechanism. Let me take out the rows and transactions that show how could these transaction result into the dead lock.

|  |
| --- |
| Transaction 1 |
| Read the value from row 4. |
| Write the literal value “8” to row 2. |
|  |
| Transaction 2 |
| Read the value from row 2. |
| Write that value to row 4. |

In the above list of rows, when transaction1 get the shared lock and read row 4 and transaction 2 get the shared lock on row 2. Transaction 1 tries to write the value in row 2 but cannot because transaction 2 has shared lock on row 2. Transaction 1 can only write on row 2 if transaction 2 releases shared lock and transaction 1 get exclusive lock. Similarly, transaction 2 cannot write value on row 4 as it is locked by transaction 1. Here transaction 1 is waiting for transaction 2 to release lock on row 2 and transaction 2 waiting for transaction 1 to release lock in row 4. which results in deadlock. However, with mutiversioning, deadlock will be removed as there is no shared lock and just reading a row won’t prevent other transaction to retrieve the data.

Example deadlock schedule in table:

|  |  |
| --- | --- |
| Transaction 1: Read the value from row 4. | Transaction 1 now has shared lock on row 4 |
| Transaction 2: Read the value from row 2. | Transaction 2 now has a shared lock on row 2 |
| Transaction 1 : Write the literal value “8” to row 2(attempt) | Transaction 1 must wait at this step, because it is unable to obtain an exclusive lock on row 2, since Transaction 2 already has a shared lock on row 2 |
| Transaction 2: Write the value to row 4.(attempt) | Transaction 2 must wait at this step, because it is unable to obtain an exclusive lock on row 4, since Transaction 1 already has a shared lock on row 4 |

# Evaluation

Your lab will be reviewed by your facilitator or instructor with the aspects and measures 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.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Aspect** | **What is Measured** | **A+ Excellent** | **B Good** | **C Fair/Satisfactory** | **D Insufficient** | **F Failure** |
| **Section 1: Results (20%)** | This measures the correctness of the results for the steps in Section 1. Excellent results are complete and accurate, with no extra rows or columns present, and no rows or columns missing. | Entirely correct | Mostly correct | Somewhat correct | Mostly incorrect | Results missing *or* Entirely incorrect. |
| **Section 1: Construction (30%)** | This measures the quality of the construction of the SQL queries and subqueries. In excellent solutions, appropriate subqueries are defined and integrated well, and dynamic lookups are used in lieu of hardcoded values, for all steps. | Excellent construction | Good construction | Satisfactory construction | Insufficient construction | No queries provided *or* Unacceptable construction |
| **Section 2: #8 (15%)** | This measures the accuracy, clarity, and thoroughness of the explanation, schedule, and resulting table contents for #8. | Entirely accurate Entirely clear Very Thorough | Mostly accurate Mostly clear Thorough | Somewhat accurate Somewhat clear | Mostly inaccurate Mostly unclear Incomplete | #8 answer missing *or* Entirely inaccurate Entirely unclear Incomplete |
| **Section 2: #9 Quality (15%)** | This measures the accuracy, clarity, and thoroughness of the explanation, schedule, and resulting table contents for #9. | Entirely accurate Entirely clear Very Thorough | Mostly accurate Mostly clear Thorough | Somewhat accurate Somewhat clear | Mostly inaccurate Mostly unclear Incomplete | #9 answer missing *or* Entirely inaccurate Entirely unclear Incomplete |
| **Overall Presentation (20%)** | This measures how well your choices are supported with explanations, and how well the document is organized and presented. | Excellent support Well organized and presented | Good support Organized and presentable | Partial support Somewhat organized and presented | Mostly unsupported Mostly disorganized presentation | No explanations Entirely disorganized presentation |
|  |  | **Preliminary Grade:** |  | **Lateness Deduction:** 5 points per day4 days maximumContact your facilitator for any exceptions |  | **Lab Grade:** |

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.