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

|  |
| --- |
| **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 lab, 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 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).
2. Write two queries which together retrieve the price of a digital thermometer in London. The first query will retrieve the currency ratio for the currency accepted in London. Your second query will hardcode the currency ratio retrieved in the first query, in order to determine the price of the thermometer in London. The first query should be dynamic in that the needed currency should be looked up rather than hardcoded. That is, the currency should be obtained by looking up the currency the store location accepts, not hardcoded by manually eyeballing the tables yourself.
3. In step 2, you determined the price of a digital thermometer in London by writing two queries. For this step, determine the same by writing a single query that contains an uncorrelated subquery. Explain:  
     
   a. how your solution makes use of the uncorrelated subquery to help retrieve the result  
   b. how and when the uncorrelated subquery is executed in the context of the outer query, and  
   c. the advantages of this solution over your solution for step 2.
4. 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 25 Euros, and for the wealthier group, they would like to know what products cost more than 300 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 €.
5. Imagine that Denisha 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 Denisha 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 Denisha’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.
6. For this problem you will write a single query to address the same use case as in step 5, 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 5, 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 5.
7. For this problem you will write a single query to address the same use case as in step 5, but change your query to use a *correlated* query combined with an EXISTS clause. The results should be the same as in step 5, 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

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

1. Compare and contrast the construction of the three different queries you developed in steps 5-7, which all address the same use case. What advantages and disadvantages do each construction have over the others? Which do you prefer and why?

# Evaluation

Your lab will be reviewed by your facilitator or instructor with the following criteria and grade breakdown.



Use the **Ask the Facilitators 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.