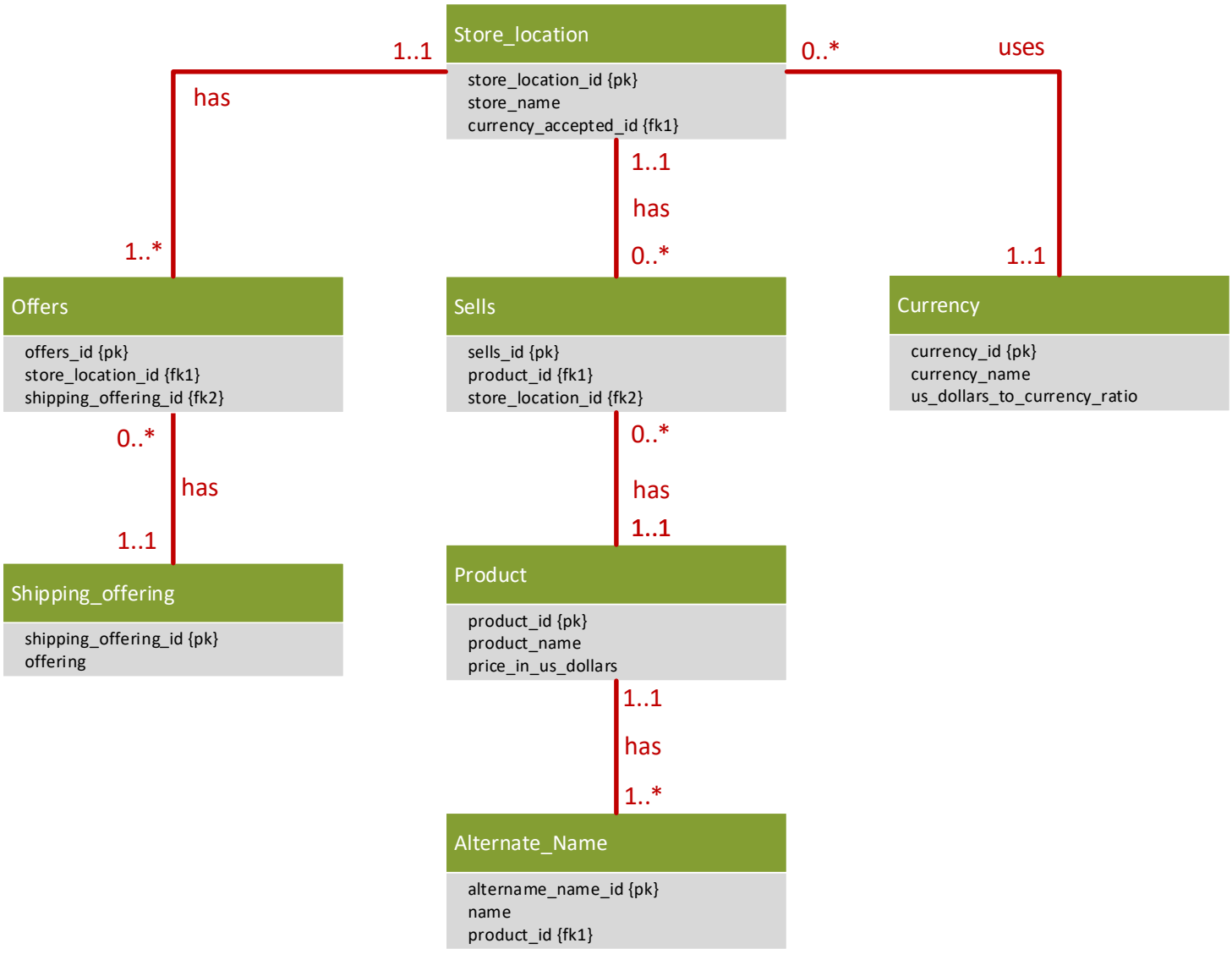
Section One – Subqueries

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).

```
180  VALUES(155, 13, 51);
181  INSERT INTO Offers(offers_id, store_location_id, shipping_offering_id)
182  VALUES(156, 13, 52);
183
184  --Toronto Extension
185  INSERT INTO Store_location(store_location_id, store_name, currency_accepted_id)
186  VALUES(14, 'Toronto Extension', 2);
187  INSERT INTO Sells(sells_id, store_location_id, product_id)
188  VALUES(1017, 14, 100);
189  INSERT INTO Sells(sells_id, store_location_id, product_id)
190  VALUES(1018, 14, 101);
191  INSERT INTO Sells(sells_id, store_location_id, product_id)
192  VALUES(1019, 14, 102);
193  INSERT INTO Sells(sells_id, store_location_id, product_id)
194  VALUES(1020, 14, 103);
195  INSERT INTO Sells(sells_id, store_location_id, product_id)
196  VALUES(1021, 14, 104);
197  INSERT INTO Offers(offers_id, store_location_id, shipping_offering_id)
198  VALUES(157, 14, 52);
199
```

121 %

Messages

```
(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)
```

All the queries to create all the tables with constraints and insert the values has been executed.

2. *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.

```
SELECT format(price_in_us_dollars *
    (SELECT us_dollars_to_currency_ratio FROM Currency WHERE currency_name = 'Britsh Pound')
    , '£.00')
    AS 'Price Digital Thermometer in London'
FROM Store_location
JOIN Sells ON Sells.store_location_id = Store_location.store_location_id
JOIN Product ON Product.product_id = Sells.product_id
WHERE store_name = 'London Extension' AND product_name = 'Digital Thermometer';
```

	Price Digital Thermometer in London
1	£167.50

The subquery returns the us_dollars_to_currency_ratio value of US dollars to British pounds (0.67). This value is used by the outer query, which retrieves the price in US dollars of a digital thermometer in London, to multiply this price (price_in_us_dollars) by 0.67, with the whole query returning the converted price in pounds of £167.50.

3. *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.

```
SELECT DISTINCT product_name AS 'Product name', format(price_in_us_dollars *
    (SELECT us_dollars_to_currency_ratio FROM Currency WHERE currency_name = 'Euro')
    , '.00 €')
    AS 'Product price in euros'
FROM Store_location
JOIN Sells ON Sells.store_location_id = Store_location.store_location_id
JOIN Product ON Product.product_id = Sells.product_id
WHERE price_in_us_dollars *
    (SELECT us_dollars_to_currency_ratio FROM Currency WHERE currency_name = 'Euro') < 26 OR
    price_in_us_dollars *
    (SELECT us_dollars_to_currency_ratio FROM Currency WHERE currency_name = 'Euro') > 299;
```

	Product name	Product price in euros
1	Bag Valve Mask	23.00 €
2	Electronic Stethoscope	322.00 €
3	Handheld Pulse Oximeter	414.00 €

Note, there are two attributes in the schema that serve as product name: product_name in table Product and name in table Alternate_Name. Since the task explicitly asked for ‘names of the products’, the attribute product_name was used.

b. Explain how what each subquery does, its role in the overall query, and how the subqueries were integrated to give the correct results.
Exactly the same subquery was used three times. The subquery retrieves the

exchange rate dollars to euros. In lines 211 to 212, the product price in dollars determined from the outer query is multiplied by the exchange rate from the subquery to get the product prices in euros, so that the product prices are displayed in euros in the output table. And in the WHERE clause in lines 218 to 221, the product prices in dollars are converted into euros before a comparison is made as to whether the prices are less than 26 euros or greater than 299 euros.
The subqueries were inserted into the query in such a way that the result of the subquery can be used by the outer query. The subquery

```
(SELECT us_dollars_to_currency_ratio FROM Currency WHERE currency_name = 'Euro')
```

returns 0.92, so the outer query can be read as follows after the subquery has been executed (without the formatting); marked in yellow is result of the subquery:

```
SELECT DISTINCT product_name, price_in_us_dollars * 0.92
FROM Store_location
JOIN Sells ON Sells.store_location_id = Store_location.store_location_id
JOIN Product ON Product.product_id = Sells.product_id
WHERE (price_in_us_dollars * 0.92) < 26 OR
      (price_in_us_dollars * 0.92) > 299;
```

It is only necessary to ensure that the correct subquery is placed in the correct place.

4. *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).

```
224 SELECT DISTINCT product_name AS 'Product name', name AS 'Alternate name'
225 FROM Store_location
226 JOIN Sells ON Sells.store_location_id = Store_location.store_location_id
227 JOIN Product ON Product.product_id = Sells.product_id
228 JOIN Alternate_name ON Alternate_name.product_id = Product.product_id
229 WHERE product_name IN
230     -- Determines unique products (product names) sold in all locations
231     -- ('Bag Valve Mask', 'Digital Thermometer', 'Handheld Pulse Oximeter').
232     (SELECT product_name FROM Product JOIN Sells ON Product.product_id = Sells.product_id
233     GROUP BY product_name
234     HAVING COUNT(Sells.product_id) =
235         -- Gets count of store locations (5).
236         (SELECT COUNT(store_location_id) FROM Store_location));
```

121 %

Results | Messages

	Product name	Alternate name
1	Bag Valve Mask	Ambu Bag
2	Bag Valve Mask	Oxygen Bag Valve Mask
3	Digital Thermometer	Thermometer
4	Handheld Pulse Oximeter	Handheld Pulse Oximeter System
5	Handheld Pulse Oximeter	Portable Pulse Oximeter

Note, the task requires that all products be listed that are available in each store location, but it was not stated that the store locations were unknown to the buyer or that the price in USD is of interest, so the store locations and prices were not listed.

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 subquery was put in the outer query after the IN keyword (line 229), goes from lines 232 to 236, and contains a subquery itself and returns the product names (product_name) of only products sold in all 5 store locations. For this, via GROUP BY in line 233, product names are grouped whose product_id occurs 5 times, lines 234 to 236, in the Sells table; if a product needs to be sold in each of the 5 locations, the product_id needs to be listed 5 times in the Sells table (Sells table contains which product (product_id) is sold in which store location (store_location_id)). In order not to have to hard code 5, another subquery is used in line 236 within the subquery in lines 232 to 236. The subquery in line 236 determines the number of store locations.

5. *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.

```
239 SELECT product_name AS 'Product name', name AS 'Alternate name'
240 FROM (SELECT Product.product_id FROM Product JOIN Sells ON Product.product_id = Sells.product_id
241     GROUP BY Product.product_id
242     HAVING COUNT(Sells.product_id) = (SELECT COUNT(store_location_id) FROM Store_location)) results_table_subquery
243 -- results_table_subquery includes results table of subquery.
244 JOIN Product ON Product.product_id = results_table_subquery.product_id
245 JOIN Alternate_name ON Alternate_name.product_id = Product.product_id;
```

121 %

Results | Messages

	Product name	Alternate name
1	Bag Valve Mask	Ambu Bag
2	Bag Valve Mask	Oxygen Bag Valve Mask
3	Digital Thermometer	Thermometer
4	Handheld Pulse Oximeter	Portable Pulse Oximeter
5	Handheld Pulse Oximeter	Handheld Pulse Oximeter System

The Store_location table in 'FROM Store_location' in the outer query has been replaced by the whole subquery, making the WHERE block unnecessary; so the WHERE block was removed. To use the results table of the subquery in the outer query, results_table_subquery was defined at the end of line 242. And this table results_table_subquery was then used to join the tables results_table_subquery, Product, and Alternate_name in the outer query. However, in the subquery from task 4, product_name had to be replaced with product_id so that the results table of the subquery results_table_subquery could be joined with the tables in the outer query. But task 4 could have been implemented with product_id instead of product_name in the subquery, which would not have made this change necessary in task 5.

6. *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.

```
248 SELECT DISTINCT product_name AS 'Product name', name AS 'Alternate name'
249 FROM Store_location
250 JOIN Sells ON Sells.store_location_id = Store_location.store_location_id
251 JOIN Product ON Product.product_id = Sells.product_id
252 JOIN Alternate_name ON Alternate_name.product_id = Product.product_id
253 WHERE EXISTS (
254     -- Gets product_id 101 (Bag Valve Mask), 102 (Digital Thermometer), and
255     -- 104 (Handheld Pulse Oximeter).
256     SELECT results_table_subquery.product_id FROM
257         -- Determines how many stores each product
258         -- (based on product_id) is sold in.
259         (SELECT product_id, COUNT(product_id) AS count_stores_sold FROM Sells
260         GROUP BY Sells.product_id) results_table_subquery
261         WHERE count_stores_sold =
262             -- Gets count of store locations (5).
263             (SELECT COUNT(store_location_id) FROM Store_location)
264             -- This correlates subquery with outer query.
265             AND Product.product_id = results_table_subquery.product_id);
```

121 %

Results | Messages

	Product name	Alternate name
1	Bag Valve Mask	Ambu Bag
2	Bag Valve Mask	Oxygen Bag Valve Mask
3	Digital Thermometer	Thermometer
4	Handheld Pulse Oximeter	Handheld Pulse Oximeter System
5	Handheld Pulse Oximeter	Portable Pulse Oximeter

Explain:

a. how your solution makes use of the correlated subquery and EXISTS clause to help retrieve the result

The query in lines 248 to 253 is the outer query. The query in lines 256 to 265 is the correlated inner query, which itself contains an inner query in lines 259 and 263. However, when the inner query is mentioned below, the correlated inner query in lines 256 to 265 is meant. Since the inner query accesses the product_id from the outer query in line 265, the inner and outer queries are correlated. The inner query performs the same logic as the WHERE part in task 4, except that it returns the product_id instead of the product name of the products sold in all 5 store locations. If the results set of the inner query contains a row, then 'WHERE EXISTS ...' is true and the outer query contains the rows for the corresponding product_id; here the rows for the product_id values 101 (Bag Valve Mask), 102 (Digital Thermometer), and 104 (Handheld Pulse Oximeter) are included in the final results table.

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

If no correlated inner query were used, then the outer query would be executed once. However, since the inner query accesses the product_id from the outer query at line 265, the inner query is executed for each iteration of the outer query. The inner query must therefore be executed for every possible row in the database that would result if only lines 248 to 252 were executed (outer query).

7. *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.

```
268  -- Defines view.
269  CREATE OR ALTER VIEW View_table AS
270  SELECT Product.product_id FROM Product JOIN Sells ON Product.product_id = Sells.product_id
271      GROUP BY Product.product_id
272      HAVING COUNT(Sells.product_id) = (SELECT COUNT(store_location_id) FROM Store_location);
```

121 %

Messages

```
Commands completed successfully.
```

```
274  SELECT product_name AS 'Product name', name AS 'Alternate name'
275  FROM View_table -- Uses view.
276  JOIN Product ON Product.product_id = View_table.product_id
277  JOIN Alternate_name ON Alternate_name.product_id = Product.product_id;
```

121 %

Results | Messages

	Product name	Alternate name
1	Bag Valve Mask	Ambu Bag
2	Bag Valve Mask	Oxygen Bag Valve Mask
3	Digital Thermometer	Thermometer
4	Handheld Pulse Oximeter	Portable Pulse Oximeter
5	Handheld Pulse Oximeter	Handheld Pulse Oximeter System

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

8. *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.

   There are many possible execution schedules, the lost update problem is shown below:

Time	Transaction	Explanation of step	Uncommitted data	Committed data
1	T1	Read the value from row 4 >> read value is 4		
2	T2	Read the value from row 2 >> read value is 2		
3	T1	Multiply that value times 3 >> 4 * 3 gives 12		
4	T2	Write that value to row 4 >> write 2 to row 4	1, 2, 3, 2, 5	
5	T1	Write the result to row 3 >> write 12 to row 3	1, 2, 12, 4, 5	
6	T2	Write the literal value "15" to row 3	1, 2, 15, 2, 5	
7	T1	Write the literal value "8" to row 2	1, 8, 12, 4, 5	
8	T2	Commit		1, 2, 15, 2, 5
9	T1	Write the literal value "20" to row 5	1, 8, 12, 4, 20	
10	T1	Commit		1, 8, 12, 4, 20

   The two orange-colored cells show where the lost update happens – the lost update is on the value of row 3. Lost update means that these two transactions are executed concurrently and both change the same value. Transaction 1 changed the value of row 3 to 12

5	T1	Write the result to row 3 >> write 12 to row 3	1, 2, 12, 4, 5	

   and transaction 2 changed the value to 15

6	T2	Write the literal value "15" to row 3	1, 2, 15, 2, 5	

   and because transaction 2 was committed and then transaction 1, the update of row 3 to 15 was lost.

Data Table (final)
1
8
12
4
20

9. *Issues with Locking and Multiversioning* – Imagine the database has both locking and multiversioning in place for concurrency control.
    a. 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.

Time	Transaction	Explanation of step	Uncommitted data	Committed data	Locks
1	T1	Read the value from row 4 >> read value is 4			No (shared) lock
2	T2	Read the value from row 2 >> read value is 2			No (shared) lock
3	T1	Multiply that value times 3 >> 4 * 3 gives 12			No (shared) lock
4	T2	Write that value to row 4 >> write 2 to row 4	1, 2, 3, 2, 5		Exclusive lock held by T2 on row 4
5	T1	Write the result to row 3 >> write 12 to row 3	1, 2, 12, 4, 5		Exclusive lock held by T1 on row 3
6	T1	Write the literal value "8" to row 2	1, 8, 12, 4, 5		Exclusive lock held by T1 on row 2
7	T1	Write the literal value "20" to row 5	1, 8, 12, 4, 20		Exclusive lock held by T1 on row 5
8	T1	Commit		1, 8, 12, 4, 20	All locks held by T1 are released
9	T2	Write the literal value "15" to row 3	1, 2, 15, 2, 5		Exclusive lock held by T2 on row 3
10	T2	Commit		1, 2, 15, 2, 5	All locks held by T2 are released

Note: each transaction only releases held locks when the transaction is completed – due to a commit or abort of the transaction.

Data Table (final)
1
2
15
2
5

    b. 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.
    Since no shared locks are used in multiversioning (read operation), no deadlocks can occur as a result of the read operations. But exclusive locks (write or delete) are used, which could result in a deadlock. However, since in this task row 3 is the only row that both transactions update, no deadlock can occur, because if T2 holds the exclusive lock on row 3, then T1 has to wait, but T2 can still finish (and then T1 can finish), and vice versa. It would be different if T1 and T2 both updated two of the same rows, then a deadlock would be possible. However, that is not the case here.