

Section One – Subqueries

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 DROP TABLE IF EXISTS Sells;
2 DROP TABLE IF EXISTS Offers;
3 DROP TABLE IF EXISTS Store_location;
4 DROP TABLE IF EXISTS Alternate_name;
5 DROP TABLE IF EXISTS Product;
6 DROP TABLE IF EXISTS Currency;
7 DROP TABLE IF EXISTS Shipping_offering;
8 CREATE TABLE Currency (
9   currency_id DECIMAL(12) NOT NULL PRIMARY KEY,
10  currency_name VARCHAR(255) NOT NULL,
11  us_dollars_to_currency_ratio DECIMAL(12,2) NOT NULL);
12 CREATE TABLE Store_location (
13   store_location_id DECIMAL(12) NOT NULL PRIMARY KEY,
14   store_name VARCHAR(255) NOT NULL,
15   currency_accepted_id DECIMAL(12) NOT NULL);
16 CREATE TABLE Product (
17   product_id DECIMAL(12) NOT NULL PRIMARY KEY,
18   product_name VARCHAR(255) NOT NULL,
19   price_in_us_dollars DECIMAL(12,2) NOT NULL);
20 CREATE TABLE Sells (
21   sells_id DECIMAL(12) NOT NULL PRIMARY KEY,
22   product_id DECIMAL(12) NOT NULL,
23   store_location_id DECIMAL(12) NOT NULL);
24 CREATE TABLE Shipping_offering (
25   shipping_offering_id DECIMAL(12) NOT NULL PRIMARY KEY,
26   offering VARCHAR(255) NOT NULL);
27 CREATE TABLE Offers (
28   offers_id DECIMAL(12) NOT NULL PRIMARY KEY,
29   store_location_id DECIMAL(12) NOT NULL,
30   shipping_offering_id DECIMAL(12) NOT NULL);
31
```

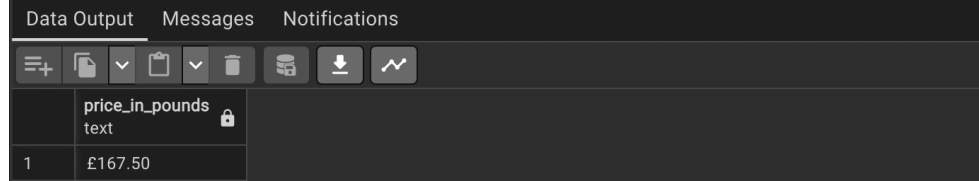
Data Output Messages Notifications

NOTICE: table "product" does not exist, skipping
NOTICE: table "currency" does not exist, skipping
NOTICE: table "shipping_offering" does not exist, skipping
INSERT 0 1

Query returned successfully in 101 msec.

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.

```
177 SELECT to_char(price_in_us_dollars *
178     (SELECT us_dollars_to_currency_ratio
179     FROM Currency
180     WHERE currency_name = 'British Pound'), 'FM£999D00') AS price_in_pounds
181 FROM Product
182 WHERE product_name = 'Digital Thermometer';
183
```



	price_in_pounds text
1	£167.50

The solution makes use of the uncorrelated subquery by retrieving the value of the ratio of the British Pound, which helps us get the converted price of the thermometer from Dollars to Pounds by multiplying the ratio by the price in Dollars, which is retrieved by the outer query. This subquery does not reference a table or value in the outer query and is therefore uncorrelated, and can be executed on its own without needing the outer query.

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.

```

184 SELECT product_name,
185        to_char(price_in_us_dollars *
186        (SELECT us_dollars_to_currency_ratio
187        FROM Currency
188        WHERE currency_name = 'Euro'), 'FM€999D00') AS price_in_euros
189 FROM Product
190 WHERE price_in_us_dollars *
191        (SELECT us_dollars_to_currency_ratio
192        FROM Currency
193        WHERE currency_name = 'Euro') < 26 OR
194        price_in_us_dollars *
195        (SELECT us_dollars_to_currency_ratio
196        FROM Currency
197        WHERE currency_name = 'Euro') > 299;
198

```

Data Output Messages Notifications

	product_name character varying (255)	price_in_euros text
1	Bag Valve Mask	€23.00
2	Electronic Stethoscope	€322.00
3	Handheld Pulse Oximeter	€414.00

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 €.

Each subquery does the same thing by retrieving the ratio of USD to Euros, and these subqueries contribute to the overall query by retrieving information to be multiplied with the prices in USD that are retrieved by the outer query. Integrating the subquery in the first SELECT clause returns all the product prices in Euros, and integrating the subqueries in the WHERE clause allows for the products in the specified price range to be returned with their prices converted to Euros. Not using these subqueries would give the incorrect answers as the prices would simply be returned in USD.

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).

```

199 SELECT Store_location.store_name,
200        Product.product_name,
201        Alternate_name.name
202 FROM Product
203 JOIN Alternate_name ON Alternate_name.product_id = Product.product_id
204 JOIN Sells ON Sells.product_id = Alternate_name.product_id
205 JOIN Store_location ON Store_location.store_location_id = Sells.store_location_id
206 WHERE Product.product_id IN
207        (SELECT Product.product_id
208         FROM Product
209         JOIN Sells ON Sells.product_id = Product.product_id
210         GROUP BY Product.product_id
211         HAVING COUNT(DISTINCT Sells.store_location_id) = (
212                  SELECT COUNT(*) FROM Store_location));

```

	store_name character varying (255)	product_name character varying (255)	name character varying (255)
1	Berlin Extension	Bag Valve Mask	Oxygen Bag Valve Mask
2	Berlin Extension	Bag Valve Mask	Ambu Bag
3	Berlin Extension	Digital Thermometer	Thermometer
4	Berlin Extension	Handheld Pulse Oximeter	Handheld Pulse Oximeter System
5	Berlin Extension	Handheld Pulse Oximeter	Portable Pulse Oximeter
6	Cancun Extension	Bag Valve Mask	Oxygen Bag Valve Mask
7	Cancun Extension	Bag Valve Mask	Ambu Bag
8	Cancun Extension	Digital Thermometer	Thermometer
9	Cancun Extension	Handheld Pulse Oximeter	Handheld Pulse Oximeter System
10	Cancun Extension	Handheld Pulse Oximeter	Portable Pulse Oximeter
11	London Extension	Bag Valve Mask	Oxygen Bag Valve Mask
12	London Extension	Bag Valve Mask	Ambu Bag
13	London Extension	Digital Thermometer	Thermometer
14	London Extension	Handheld Pulse Oximeter	Handheld Pulse Oximeter System
15	London Extension	Handheld Pulse Oximeter	Portable Pulse Oximeter
16	New York Extension	Bag Valve Mask	Oxygen Bag Valve Mask

17	New York Extension	Bag Valve Mask	Ambu Bag
18	New York Extension	Digital Thermometer	Thermometer
19	New York Extension	Handheld Pulse Oximeter	Handheld Pulse Oximeter System
20	New York Extension	Handheld Pulse Oximeter	Portable Pulse Oximeter
21	Toronto Extension	Bag Valve Mask	Oxygen Bag Valve Mask
22	Toronto Extension	Bag Valve Mask	Ambu Bag
23	Toronto Extension	Digital Thermometer	Thermometer
24	Toronto Extension	Handheld Pulse Oximeter	Handheld Pulse Oximeter System
25	Toronto Extension	Handheld Pulse Oximeter	Portable Pulse Oximeter
Total rows: 25 of 25		Query complete 00:00:00.256	

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.

The subquery was integrated into the WHERE clause to return only the products that were relevant to Esther by being sold in all of the stores. Without the subquery, the outer query simply returns all products and their alternate names at all stores, without specifying if the product is only sold at every store and not just a few. The WHERE clause takes the criteria of the subquery and applies it to each row being selected by the outer query, and restricting/excluding rows based on if they match the subquery criteria. The subquery returns products that are only present IN all store locations, with the HAVING COUNT portion of the subquery assisting with aggregating the results and making sure the locations do not have to be hard coded by checking that the products are sold in each distinct location and not just in the existing 5.

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.

```

214 SELECT Store_location.store_name,
215        products.product_name,
216        Alternate_name.name
217 FROM (SELECT Product.product_id,
218        Product.product_name
219        FROM Product
220        JOIN Sells ON Sells.product_id = Product.product_id
221        GROUP BY Product.product_id, Product.product_name
222        HAVING COUNT(DISTINCT Sells.store_location_id) =
223        (SELECT COUNT(*) FROM Store_location)) products
224 JOIN Alternate_name ON Alternate_name.product_id = products.product_id
225 JOIN Sells ON Sells.product_id = Alternate_name.product_id
226 JOIN Store_location ON Store_location.store_location_id = Sells.store_location_id;

```

Data Output Messages Notifications

	store_name character varying (255)	product_name character varying (255)	name character varying (255)
1	Berlin Extension	Bag Valve Mask	Oxygen Bag Valve Mask
2	Berlin Extension	Bag Valve Mask	Ambu Bag
3	Berlin Extension	Digital Thermometer	Thermometer
4	Berlin Extension	Handheld Pulse Oximeter	Handheld Pulse Oximeter System
5	Berlin Extension	Handheld Pulse Oximeter	Portable Pulse Oximeter
6	Cancun Extension	Bag Valve Mask	Oxygen Bag Valve Mask
7	Cancun Extension	Bag Valve Mask	Ambu Bag
8	Cancun Extension	Digital Thermometer	Thermometer
9	Cancun Extension	Handheld Pulse Oximeter	Handheld Pulse Oximeter System
10	Cancun Extension	Handheld Pulse Oximeter	Portable Pulse Oximeter
11	London Extension	Bag Valve Mask	Oxygen Bag Valve Mask
12	London Extension	Bag Valve Mask	Ambu Bag
13	London Extension	Digital Thermometer	Thermometer
14	London Extension	Handheld Pulse Oximeter	Handheld Pulse Oximeter System
15	London Extension	Handheld Pulse Oximeter	Portable Pulse Oximeter
16	New York Extension	Bag Valve Mask	Oxygen Bag Valve Mask
17	New York Extension	Bag Valve Mask	Ambu Bag

18	New York Extension	Digital Thermometer	Thermometer
19	New York Extension	Handheld Pulse Oximeter	Handheld Pulse Oximeter System
20	New York Extension	Handheld Pulse Oximeter	Portable Pulse Oximeter
21	Toronto Extension	Bag Valve Mask	Oxygen Bag Valve Mask
22	Toronto Extension	Bag Valve Mask	Ambu Bag
23	Toronto Extension	Digital Thermometer	Thermometer
24	Toronto Extension	Handheld Pulse Oximeter	Handheld Pulse Oximeter System
25	Toronto Extension	Handheld Pulse Oximeter	Portable Pulse Oximeter
Total rows: 25 of 25		Query complete 00:00:00.266	

Integrating the subquery into the FROM clause allows for the same filtering and results as putting it in the WHERE clause, but integrating it into the FROM clause allows for more flexibility because a subquery can be used in the FROM clause without regard to the volume of tabular data it retrieves, because the FROM clause expects tabular data. The subquery was implemented in the FROM clause with the use of the “products” alias as a name for the subquery results, and the alias is used like a table for the results which can be used to join with other tables. The alias allows for columns retrieved by the subquery in the FROM clause to be used outside of the FROM clause in the outer query. As shown in the second line of the outer query, the alias takes the place of the table and there is no need to join the Product table.

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

```

229 SELECT Store_location.store_name,
230        Product.product_name,
231        Alternate_name.name
232 FROM Product
233 JOIN Alternate_name ON Alternate_name.product_id = Product.product_id
234 JOIN Sells ON Sells.product_id = Alternate_name.product_id
235 JOIN Store_location ON Store_location.store_location_id = Sells.store_location_id
236 WHERE EXISTS (
237     SELECT Sells.product_id
238     FROM Sells
239     WHERE Sells.product_id = Product.product_id
240     GROUP BY Sells.product_id
241     HAVING COUNT(DISTINCT Sells.store_location_id) = (SELECT COUNT(*) FROM Store_location)
242 );

```

Data Output Messages Notifications



	store_name character varying (255)	product_name character varying (255)	name character varying (255)
1	Toronto Extension	Bag Valve Mask	Ambu Bag
2	New York Extension	Bag Valve Mask	Ambu Bag
3	London Extension	Bag Valve Mask	Ambu Bag
4	Cancun Extension	Bag Valve Mask	Ambu Bag
5	Berlin Extension	Bag Valve Mask	Ambu Bag
6	Toronto Extension	Bag Valve Mask	Oxygen Bag Valve Mask
7	New York Extension	Bag Valve Mask	Oxygen Bag Valve Mask
8	London Extension	Bag Valve Mask	Oxygen Bag Valve Mask
9	Cancun Extension	Bag Valve Mask	Oxygen Bag Valve Mask
10	Berlin Extension	Bag Valve Mask	Oxygen Bag Valve Mask
11	Toronto Extension	Digital Thermometer	Thermometer
12	New York Extension	Digital Thermometer	Thermometer
13	London Extension	Digital Thermometer	Thermometer
14	Cancun Extension	Digital Thermometer	Thermometer
15	Berlin Extension	Digital Thermometer	Thermometer
16	Toronto Extension	Handheld Pulse Oximeter	Portable Pulse Oximeter

17	New York Extension	Handheld Pulse Oximeter	Portable Pulse Oximeter
18	London Extension	Handheld Pulse Oximeter	Portable Pulse Oximeter
19	Cancun Extension	Handheld Pulse Oximeter	Portable Pulse Oximeter
20	Berlin Extension	Handheld Pulse Oximeter	Portable Pulse Oximeter
21	Toronto Extension	Handheld Pulse Oximeter	Handheld Pulse Oximeter System
22	New York Extension	Handheld Pulse Oximeter	Handheld Pulse Oximeter System
23	London Extension	Handheld Pulse Oximeter	Handheld Pulse Oximeter System
24	Cancun Extension	Handheld Pulse Oximeter	Handheld Pulse Oximeter System
25	Berlin Extension	Handheld Pulse Oximeter	Handheld Pulse Oximeter System
Total rows: 25 of 25		Query complete 00:00:00.099	

The EXISTS clause returns true if the subquery within it returns any rows at all, is good for checking the existence of certain items, and is more effectively used with a correlated subquery. The 11th line of the query above makes the subquery a correlated subquery, as it references the Product table from the outer query. By using EXISTS, the subquery makes use of the boolean by returning each product ID from the Product table only if the product EXISTS/is being sold in each distinct store.

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

The correlated subquery retrieves results based on the current row of the outer query because it references at least one table from the outer query. If the current row in the outer query does not match the criteria within the correlated subquery, the row is excluded from the result set. Correlated subqueries only make sense within the context of the outer query and they cannot be executed on their own like uncorrelated queries. The correlated subquery is executed for each row of the outer query and retrieves one result set per row of the outer query, as opposed to uncorrelated subqueries which are executed once and retrieve results once.

7. *Correlated Subquery 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.

```

243 CREATE OR REPLACE VIEW Products_available AS
244 SELECT Product.product_id,
245        Product.product_name
246 FROM Product
247 JOIN Sells ON Sells.product_id = Product.product_id
248 GROUP BY Product.product_id, Product.product_name
249 HAVING COUNT(DISTINCT Sells.store_location_id) =
250        (SELECT COUNT(*) FROM Store_location);
251
252 SELECT products_available.product_name,
253        Store_location.store_name,
254        Alternate_name.name
255 FROM Products_available
256 JOIN Alternate_name ON Alternate_name.product_id = products_available.product_id
257 JOIN Sells ON Sells.product_id = Alternate_name.product_id
258 JOIN Store_location ON Store_location.store_location_id = Sells.store_location_id;

```

	product_name character varying (255)	store_name character varying (255)	name character varying (255)
1	Bag Valve Mask	Berlin Extension	Oxygen Bag Valve Mask
2	Bag Valve Mask	Berlin Extension	Ambu Bag
3	Digital Thermometer	Berlin Extension	Thermometer
4	Handheld Pulse Oximeter	Berlin Extension	Handheld Pulse Oximeter System
5	Handheld Pulse Oximeter	Berlin Extension	Portable Pulse Oximeter
6	Bag Valve Mask	Cancun Extension	Oxygen Bag Valve Mask
7	Bag Valve Mask	Cancun Extension	Ambu Bag
8	Digital Thermometer	Cancun Extension	Thermometer
9	Handheld Pulse Oximeter	Cancun Extension	Handheld Pulse Oximeter System
10	Handheld Pulse Oximeter	Cancun Extension	Portable Pulse Oximeter
11	Bag Valve Mask	London Extension	Oxygen Bag Valve Mask
12	Bag Valve Mask	London Extension	Ambu Bag
13	Digital Thermometer	London Extension	Thermometer
14	Handheld Pulse Oximeter	London Extension	Handheld Pulse Oximeter System

15	Handheld Pulse Oximeter	London Extension	Portable Pulse Oximeter
16	Bag Valve Mask	New York Extension	Oxygen Bag Valve Mask
17	Bag Valve Mask	New York Extension	Ambu Bag
18	Digital Thermometer	New York Extension	Thermometer
19	Handheld Pulse Oximeter	New York Extension	Handheld Pulse Oximeter System
20	Handheld Pulse Oximeter	New York Extension	Portable Pulse Oximeter
21	Bag Valve Mask	Toronto Extension	Oxygen Bag Valve Mask
22	Bag Valve Mask	Toronto Extension	Ambu Bag
23	Digital Thermometer	Toronto Extension	Thermometer
24	Handheld Pulse Oximeter	Toronto Extension	Handheld Pulse Oximeter System
25	Handheld Pulse Oximeter	Toronto Extension	Portable Pulse Oximeter
Total rows: 25 of 25		Query complete 00:00:00.517	

Section Two – Concurrency

Use the tables and transactions provided in the lab; do not create your own.

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, Word, or another comparable application.

Lost Update Schedule	
Step	Explanation
Transaction 1: Read the value from row 4	The value from row 4 is read as 4
Transaction 1: Multiply that value times 3	The read value of 4 is now 12 after being multiplied by 3
Transaction 2: Read the value from row 2	The value from row 2 is read as 2
Transaction 2: Write that value to row 4	The value of row 4 is now 2
Transaction 1: Write that result to row 3	The value of row 3 is now 12
Transaction 2: Write the literal value “15” to row 3	The value of row 3 is now 15
Transaction 2: Commit	The changes from Transaction 2 are made permanent in the database
Transaction 1: Write the literal value “8” to row 2	The value of row 2 is now 8

Transaction 1: Write the literal value “20” to row 5	The value of row 5 is now 20
Transaction 1: Commit	The changes from Transaction 1 are made permanent in the database

This is an example of a lost update created by interleaved transactions because Transaction 1 and Transaction 2 both update the same row concurrently. Despite both transactions attempting to write a new value to row 3, Transaction 2 is committed first and the value of row 3 from Transaction 1 is lost. The final data table is a mix of the incomplete interleaved transactions and looks like the following:

Data Table
1
8
15
2
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 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.

Lost Update Schedule	
Step	Explanation
Transaction 1: Read the value from row 4	The value from row 4 is read as 4 (Multiversioning makes it so that this value will remain as 4 for Transaction 1 because this is the value of this row when the transaction started)
Transaction 1: Multiply that value times 3	The read value of 4 is now 12 after being multiplied by 3
Transaction 2: Read the value from row 2	The value from row 2 is read as 2 (Multiversioning makes it so that this value will remain as 2 for Transaction 2 because this is the value of this row when the transaction started)
Transaction 2: Write that value to row 4	The value of row 4 is now 2 (Multiversioning allows this to happen as shared locks aren't necessary as updating the row does not affect the reading transaction) (Transaction 2 has an exclusive lock on row 4)
Transaction 1: Write that result to row 3	The value of row 3 is now 12 (Transaction 1 has an exclusive lock on row 3)
Transaction 2: Write the literal value "15" to row 3 (attempt)	The value of row 3 is now 15 (This transaction must wait because Transaction 1 has an exclusive lock on row 3)
Transaction 1: Write the literal value "8" to row 2	The value of row 2 is now 8 (Transaction 1 has an exclusive lock on row 2)
Transaction 1: Write the literal value "20" to row 5	The value of row 5 is now 20 (Transaction 2 has an exclusive

	lock on row 5)
Transaction 1: Commit	The changes from Transaction 1 are made permanent in the database (All exclusive locks are released now)
Transaction 2: Write the literal value "15" to row 3	The value of row 3 is now 15 (This transaction stops waiting and now has an exclusive lock on row 3 now that Transaction 1 is committed and has released its locks)
Transaction 2: Commit	The changes from Transaction 2 are made permanent in the database (Transaction 2 releases its locks)

The resulting data table looks similar to the previous step but the transactions both completed successfully thanks to locking and multiversioning preventing concurrency conflicts, and the database would save a history of the values before the transactions updated them.

Data Table
1
8
15
2
20

- 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.

A schedule of these transactions could not result in deadlock thanks to multiversioning, which makes it such that deadlocks are much more uncommon due to not needing shared locks when a transaction is updating a row that is currently being read. None of the reads take out locks. When both transactions write to row 3, the timing can cause one transaction to have to wait because of the exclusive lock preventing both from updating row 3 at the same time, but this does not cause a deadlock because the second transaction will execute normally after the first commits, and nothing is stopping the progression of the first transaction.