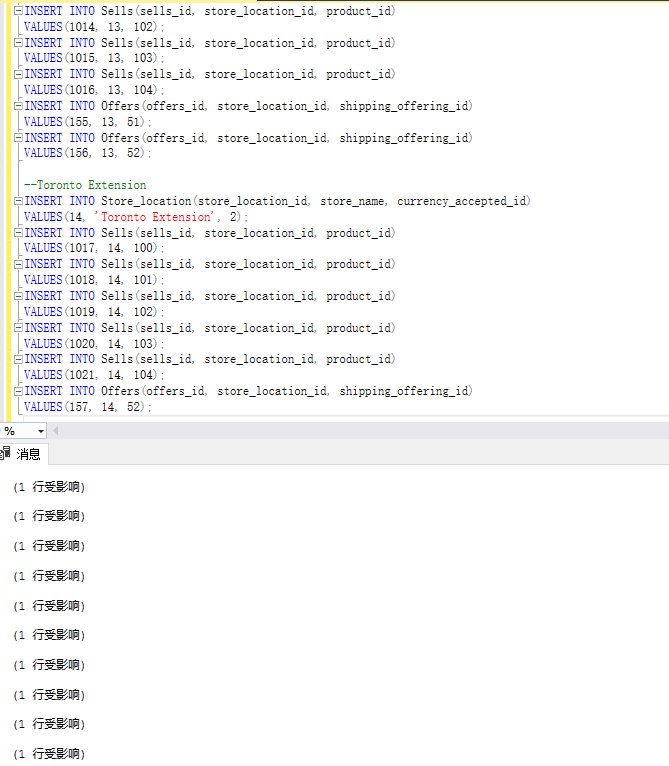
CS669

SiCheng Yi

LAB5

**Section One – Subqueries**

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).



1. 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.

SELECT us\_dollars\_to\_currency\_ratio

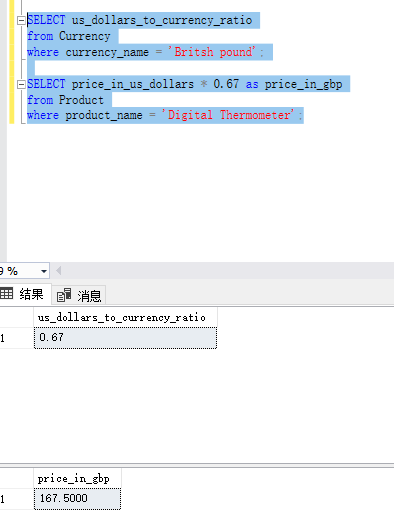
from Currency

where currency\_name = 'Britsh pound';

SELECT price\_in\_us\_dollars \* 0.67 as price\_in\_gbp

from Product

where product\_name = 'Digital Thermometer';



1. 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.

SELECT format(price\_in\_us\_dollars \*

(SELECT us\_dollars\_to\_currency\_ratio

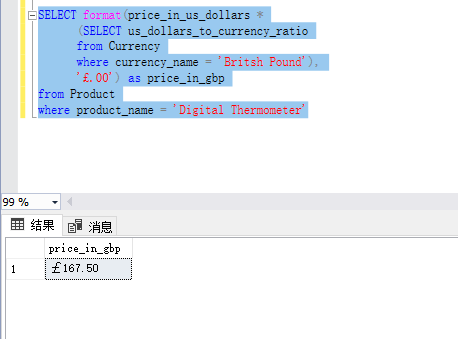
from Currency

where currency\_name = 'Britsh Pound'),

'£.00') as price\_in\_gbp

from Product

where product\_name = 'Digital Thermometer'



a. how your solution makes use of the uncorrelated subquery to help retrieve the result

The inner query can be run without the need for the outer query to be valid to provide viable results, we use the results of the inner query to complete the outer query.

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

The outer query does the inner query first, and then uses its results to query.

c. the advantages of this solution over your solution for step 2.

Inexpensive to run, requires no human intervention, and produces correct results even when values change.

1. 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 €.

SELECT product\_name,

format(price\_in\_us\_dollars \*

(SELECT us\_dollars\_to\_currency\_ratio

from Currency

where currency\_name = 'Euro'),

'€.00') as price\_in\_euro

from Product

where product.price\_in\_us\_dollars \*

(SELECT us\_dollars\_to\_currency\_ratio

From Currency

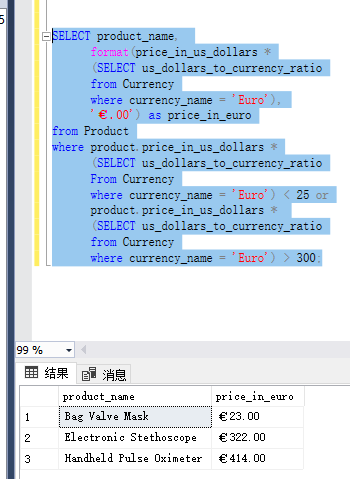
where currency\_name = 'Euro') < 25 or

product.price\_in\_us\_dollars \*

(SELECT us\_dollars\_to\_currency\_ratio

from Currency

where currency\_name = 'Euro') > 300;



First, a subquery is made to provide the US to EUR ratio. This multiplied with outer subquery yields a list of all products. Use €.

Then, make a subquery to filter prices to be less than 25 or greater than 300.

1. 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.
   1. 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).
   2. 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.

SELECT DISTINCT Product.product\_name,

Alternate\_name.name AS alternate\_name,

format(Product.price\_in\_us\_dollars, '$.00') as price

From Store\_location

join Sells ON sells.store\_location\_id = Store\_location.store\_location\_id

join Product ON Product.product\_id = Sells.product\_id

join Alternate\_name ON Alternate\_name.product\_id = Product.product\_id

where Product.product\_id IN

(SELECT product.product\_id

from Product

join Sells

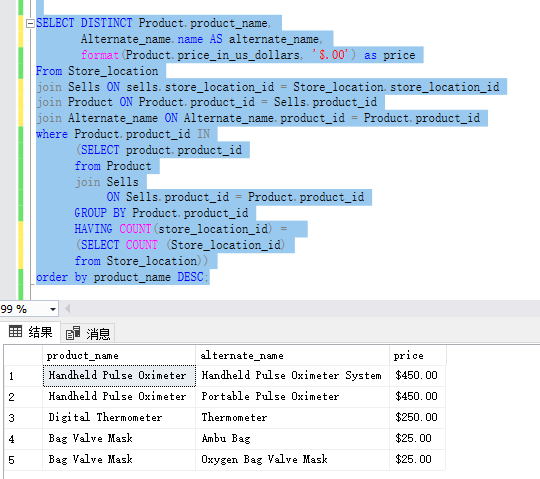
ON Sells.product\_id = Product.product\_id

GROUP BY Product.product\_id

HAVING COUNT(store\_location\_id) =

(SELECT COUNT (Store\_location\_id)

from Store\_location))

order by product\_name DESC;

The first subquery is the outer query itself, which produces a list of all product names, alternate names, and prices. The second subquery generates a list of all product IDs present in the sales table, and the third subquery generates all stores, and links the second and third subqueries with their results to filter the results of the inner query.

1. 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.

SELECT DISTINCT Product.product\_name,

Alternate\_name.name,

format(Product.price\_in\_us\_dollars, '$.00') as price

From (SELECT Product.product\_id

from Product

join Sells

ON Sells.product\_id = Product.product\_id

GROUP BY Product.product\_id

HAVING COUNT (Sells.product\_id) =

(SELECT COUNT (Store\_location\_id)

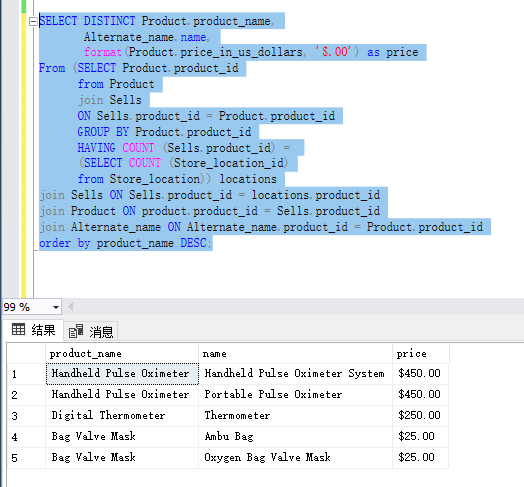
from Store\_location)) locations

join Sells ON Sells.product\_id = locations.product\_id

join Product ON product.product\_id = Sells.product\_id

join Alternate\_name ON Alternate\_name.product\_id = Product.product\_id

order by product\_name DESC;



1. 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.

SELECT DISTINCT Product.product\_name,

Alternate\_name.name AS alternate\_name,

format(Product.price\_in\_us\_dollars, '$.00') as price

From Store\_location

join Sells ON sells.store\_location\_id = Store\_location.store\_location\_id

join Product ON Product.product\_id = Sells.product\_id

join Alternate\_name ON Alternate\_name.product\_id = Product.product\_id

where EXISTS (SELECT All\_Stores.product\_id, All\_Stores.count\_ells

from (SELECT Sells.product\_id, COUNT(Sells.product\_id) AS count\_ells

from Sells

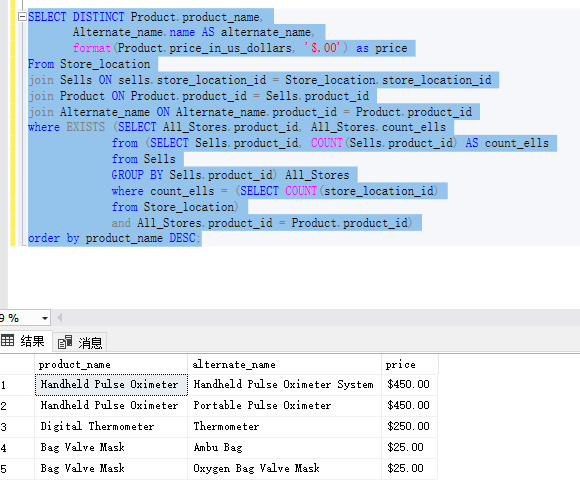
GROUP BY Sells.product\_id) All\_Stores

where count\_ells = (SELECT COUNT(store\_location\_id)

from Store\_location)

and All\_Stores.product\_id = Product.product\_id)

order by product\_name DESC;



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?

Where is the slowest way, but the easiest to build and inspect the understanding structure. From can create subqueries most flexibly without considering rows and columns, and can be filtered relatively easily. Exists only handles subqueries and is fast. But EXISTS usually demands a more complex method of combining the queries for each part. The subquery usually must be correlated with the outer query to get the results we want. A correlated subquery references at least one table from the outer query, which means that conceptually , the subquery is not an independent query.

I like the form of From the most, because he is the most flexible and can achieve the purpose I need more simply.