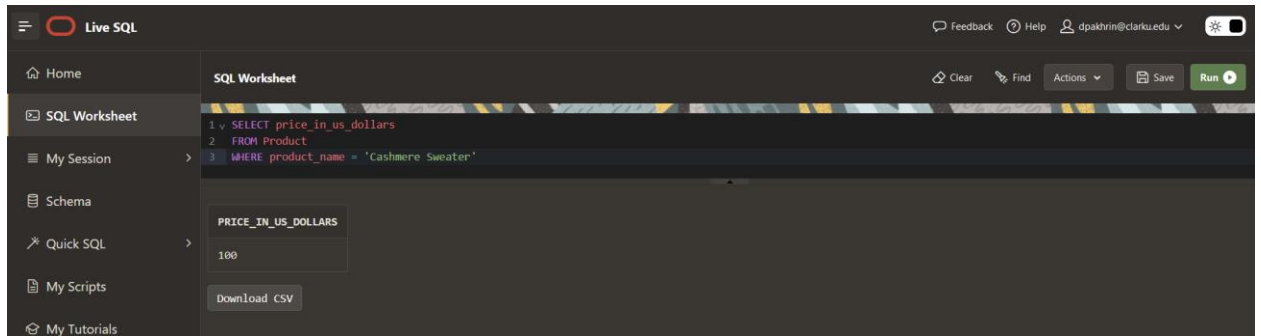


# Lab 5 – Submission Sheet

## Section One

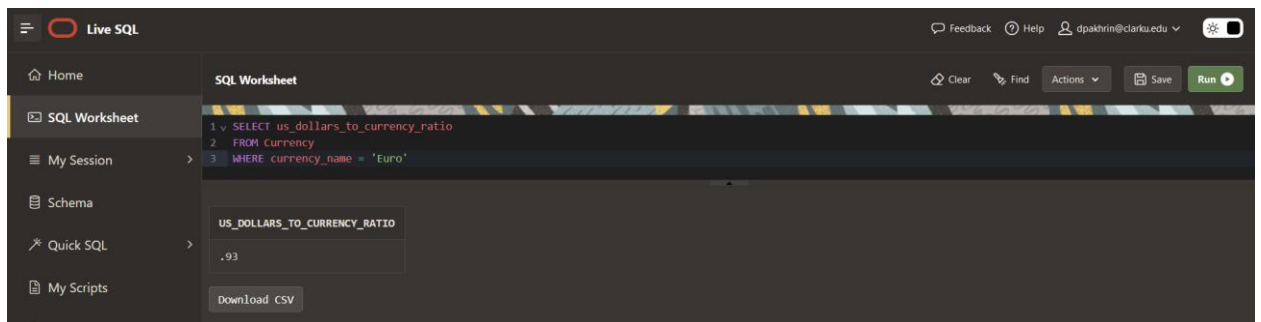
### 1. SELECT (Screen Shots)



The screenshot shows the 'Live SQL' interface. The left sidebar contains navigation links: Home, SQL Worksheet, My Session, Schema, Quick SQL, My Scripts, and My Tutorials. The main area is titled 'SQL Worksheet' and contains the following SQL query:

```
1 SELECT price_in_us_dollars
2 FROM Product
3 WHERE product_name = 'Cashmere Sweater'
```

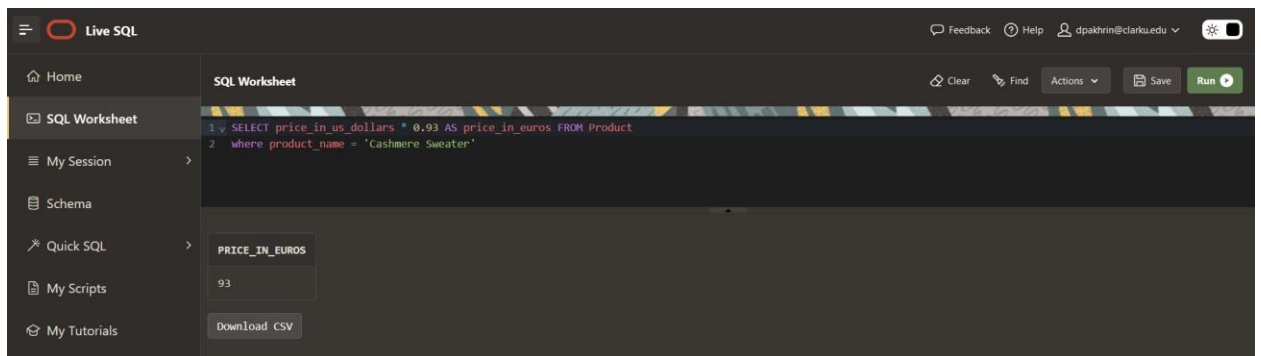
Below the query, the results are displayed in a table with the column header 'PRICE\_IN\_US\_DOLLARS' and a single value '100'. A 'Download CSV' button is located at the bottom of the results area.



The screenshot shows the 'Live SQL' interface. The left sidebar contains navigation links: Home, SQL Worksheet, My Session, Schema, Quick SQL, My Scripts, and My Tutorials. The main area is titled 'SQL Worksheet' and contains the following SQL query:

```
1 SELECT us_dollars_to_currency_ratio
2 FROM currency
3 WHERE currency_name = 'Euro'
```

Below the query, the results are displayed in a table with the column header 'US\_DOLLARS\_TO\_CURRENCY\_RATIO' and a single value '.93'. A 'Download CSV' button is located at the bottom of the results area.



The screenshot shows the 'Live SQL' interface. The left sidebar contains navigation links: Home, SQL Worksheet, My Session, Schema, Quick SQL, My Scripts, and My Tutorials. The main area is titled 'SQL Worksheet' and contains the following SQL query:

```
1 SELECT price_in_us_dollars * 0.93 AS price_in_euros FROM Product
2 where product_name = 'Cashmere Sweater'
```

Below the query, the results are displayed in a table with the column header 'PRICE\_IN\_EUROS' and a single value '93'. A 'Download CSV' button is located at the bottom of the results area.

## 2. SELECT (Screen Shots)



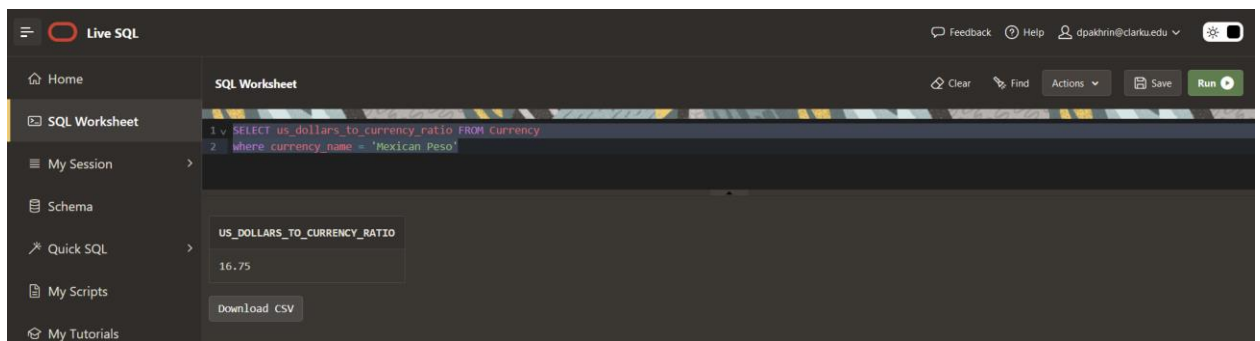
Live SQL interface showing a SQL Worksheet with the following query:

```
1 SELECT currency_name FROM Currency
2 WHERE currency_id IN (
3   SELECT currency_accepted_id FROM Store_location
4   WHERE store_name = 'Cancun Extension'
5 );
```

The result set displays the following data:

CURRENCY_NAME
Mexican Peso

A "Download CSV" button is visible below the result set.



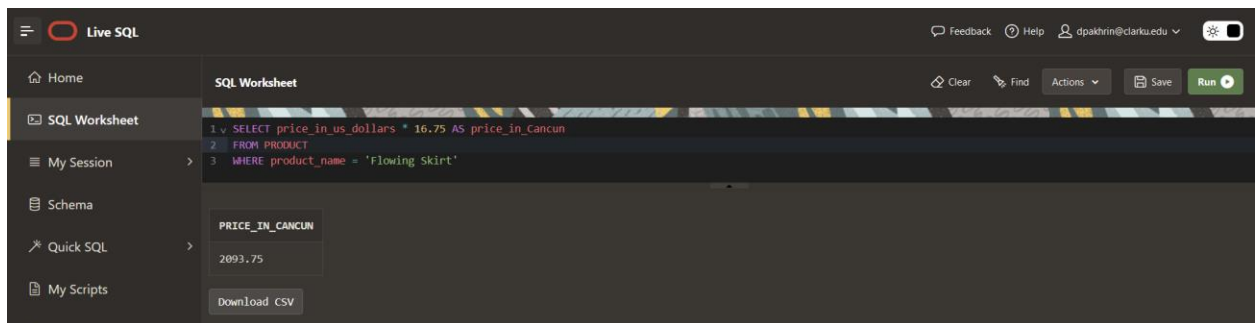
Live SQL interface showing a SQL Worksheet with the following query:

```
1 SELECT us_dollars_to_currency_ratio FROM Currency
2 WHERE currency_name = 'Mexican Peso';
```

The result set displays the following data:

US_DOLLARS_TO_CURRENCY_RATIO
16.75

A "Download CSV" button is visible below the result set.



Live SQL interface showing a SQL Worksheet with the following query:

```
1 SELECT price_in_us_dollars * 16.75 AS price_in_cancun
2 FROM PRODUCT
3 WHERE product_name = 'Flowing Skirt';
```

The result set displays the following data:

PRICE_IN_CANCUN
2093.75

A "Download CSV" button is visible below the result set.

## 3.

### a. EXPLANATION

Datatypes are the type of data to be recorded in the table's particular column. It ensures the validity of the values to be stored in that column.

For instance, the VARCHAR datatype allows only the string data to be stored in the column. To define the maximum size allowed for the column, we provide the size such that VARCHAR(255) that can store up to 255 characters.

If the value exceeds the defined size of the data, we receive value too large for column error.

## b. CREATE TABLE (Screen Shot)



The screenshot shows the Live SQL interface. The SQL Worksheet contains the following code:

```
1 CREATE TABLE Sports (
2   sports_name VARCHAR(6)
3 );
4
5 INSERT INTO Sports(sports_name)
6 VALUES('Badminton')
7
```

The output shows "Table created." followed by an error message:

```
ORA-12899: value too large for column "SQL_OQJHREKSCIPWQJ5VPC30PHLY"."SPORTS"."SPORTS_NAME" (actual: 9, maximum: 6) ORA-06512: at "SYS.DBMS_SQL", line 1721
More Details: https://docs.oracle.com/error-help/db/ora-12899
```

## c. INSERT (Screen Shot)



The screenshot shows the Live SQL interface. The SQL Worksheet contains the following code:

```
1 CREATE TABLE Sports (
2   sports_name VARCHAR(6)
3 );
4
5 INSERT INTO Sports(sports_name)
6 VALUES('Badminton')
7
```

The output shows "Table created." followed by an error message:

```
ORA-12899: value too large for column "SQL_OQJHREKSCIPWQJ5VPC30PHLY"."SPORTS"."SPORTS_NAME" (actual: 9, maximum: 6) ORA-06512: at "SYS.DBMS_SQL", line 1721
More Details: https://docs.oracle.com/error-help/db/ora-12899
```

## 4. EXPLANATION

Expression in a DBMS is a combination of operands and arithmetic signs resulting in an operation.

The DBMS strictly follows the precedence to determine the result. It follows the PEMDAS rule for the mathematical operation which means Parentheses, Exponents, Multiplication, Division, Addition, and Subtraction.

For example,

`SELECT (5 + 3) * 2`

Here the operation inside the parenthesis is calculated first and the result is multiplied by 2 which gives an output of 16.

## 5. EXPLANATION

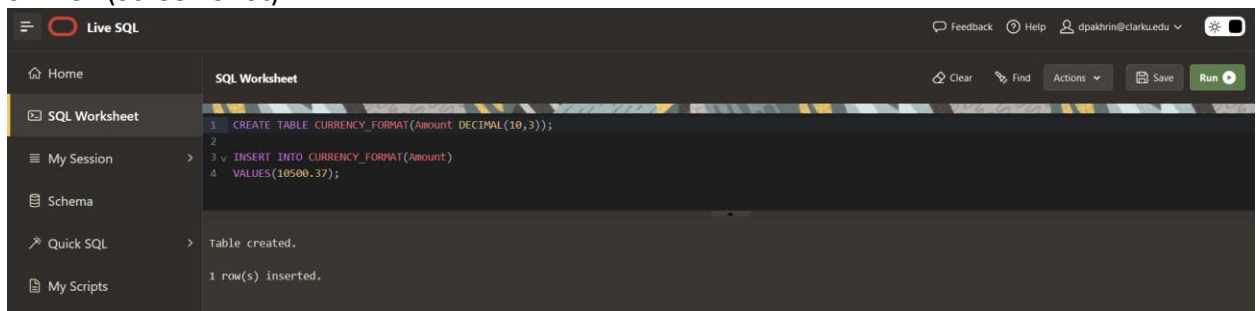
In DBMS, there is the level of precedence of datatype. There may be an operation between an unidentical datatype and the operation makes the datatype of the result to the higher precedence. In other words, the lower precedence datatype is implicitly converted into a higher precedence datatype.

For instance,

```
SELECT 'AGE' || 24;
```

Here we have two datatypes (AGE as VARCHAR and 24 as INT) which are concatenated by ||. Since VARCHAR has higher precedence, it implicitly converts INT into a String resulting an string output 'AGE24'.

## 6. SELECT (Screen Shot)

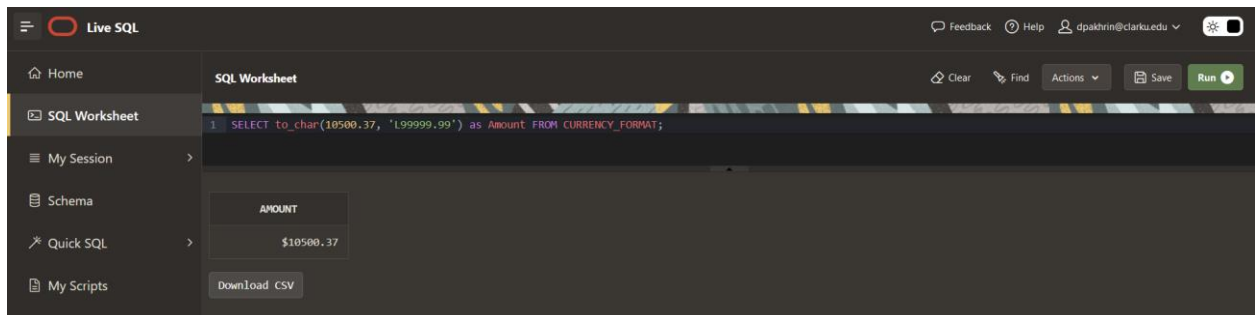


The screenshot shows the Live SQL interface. The SQL Worksheet contains the following code:

```
1 CREATE TABLE CURRENCY_FORMAT(Amount DECIMAL(10,3));
2
3 INSERT INTO CURRENCY_FORMAT(Amount)
4 VALUES(10500.37);
```

The output shows:

```
Table created.
1 row(s) inserted.
```



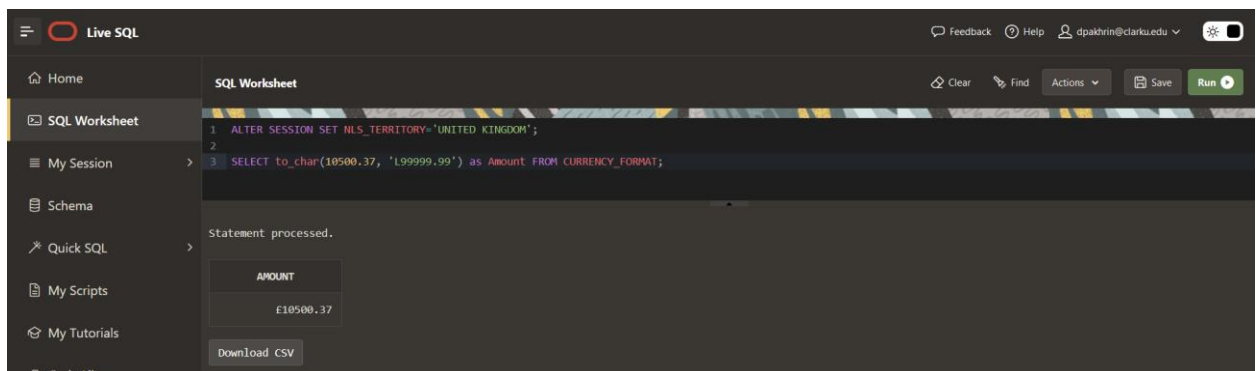
The screenshot shows the Live SQL interface with the following SQL statement:

```
1 SELECT to_char(10500.37, 'L99999.99') as Amount FROM CURRENCY_FORMAT;
```

The output shows a table with one row:

AMOUNT
\$10500.37

A "Download CSV" button is visible below the table.



The screenshot shows the Live SQL interface with the following SQL statements:

```
1 ALTER SESSION SET NLS_TERRITORY='UNITED KINGDOM';
2
3 SELECT to_char(10500.37, 'L99999.99') as Amount FROM CURRENCY_FORMAT;
```

The output shows "Statement processed." followed by a table with one row:

AMOUNT
£10500.37

A "Download CSV" button is visible below the table.

## Section Two

### 7. SELECT (Screen Shot)



The screenshot shows the Live SQL interface. The SQL Worksheet contains the following query:

```

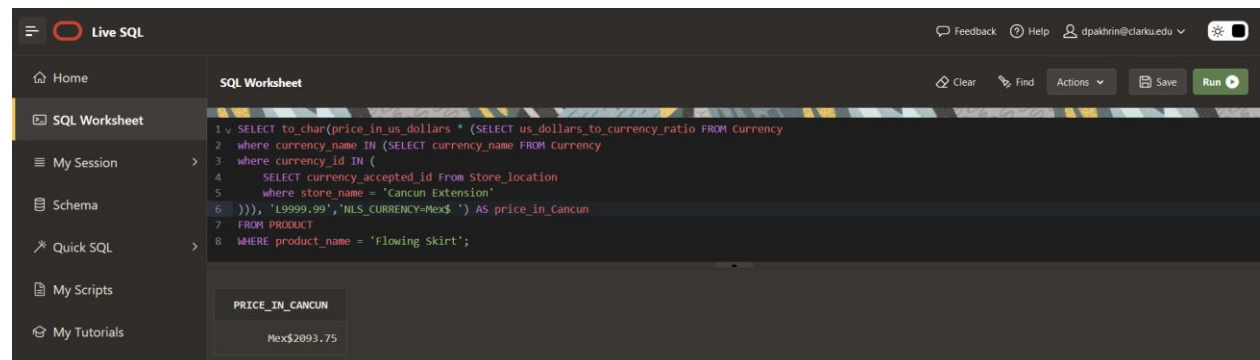
1 ALTER SESSION SET NLS_TERRITORY='FRANCE';
2
3 SELECT to_char(price_in_us_dollars *
4 (SELECT us_dollars_to_currency_ratio
5 FROM Currency
6 WHERE currency_name = 'Euro'), 'L99,99') AS price_in_euros
7 FROM Product
8 WHERE product_name = 'Cashmere Sweater'
  
```

Below the query, the output shows the statement was processed successfully, and the result is displayed in a table:

PRICE_IN_EUROS
€93.00

### 8.

#### a. SELECT (Screen Shot)



The screenshot shows the Live SQL interface. The SQL Worksheet contains the following query:

```

1 SELECT to_char(price_in_us_dollars * (SELECT us_dollars_to_currency_ratio FROM Currency
2 where currency_name IN (SELECT currency_name FROM Currency
3 where currency_id IN (
4 SELECT currency_accepted_id FROM store_location
5 where store_name = 'Cancun Extension'
6 )), 'L9999.99', 'NLS_CURRENCY=Mex$') AS price_in_cancun
7 FROM Product
8 WHERE product_name = 'Flowing Skirt';
  
```

Below the query, the output shows the statement was processed successfully, and the result is displayed in a table:

PRICE_IN_CANCUN
Mex\$2093.75

#### b. EXPLANATION

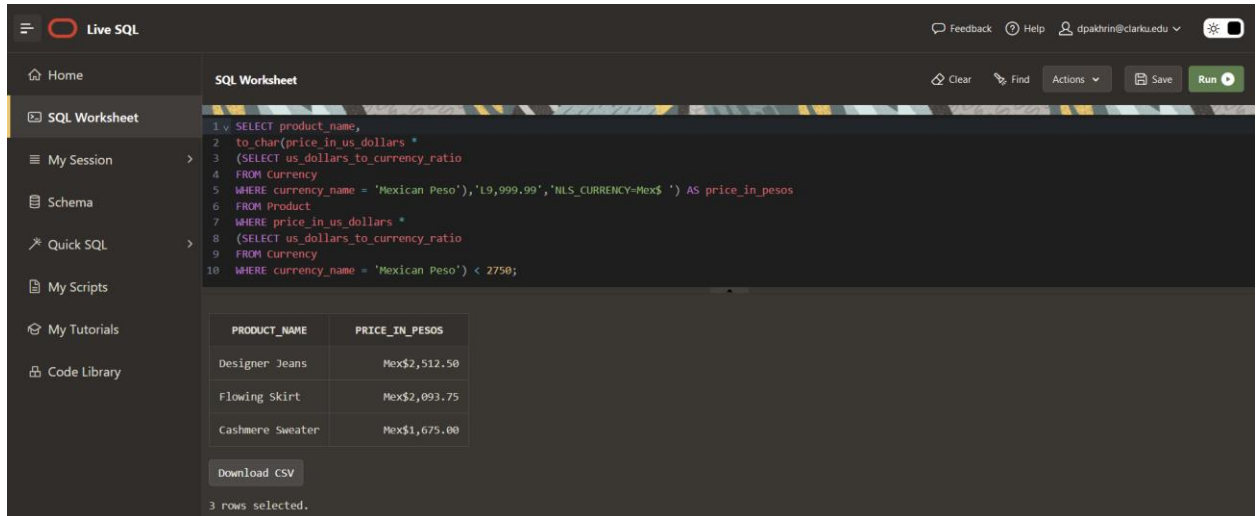
In the above query, we have three levels of subqueries. We want to display the price of the Flowing Skirt accepted in Cancun. For that we want the currency accepted in the Cancun which is extracted from the Store location table. This is the most inner sub query which returns the currency id. The next sub query above the previous sub query will use the currency id to return the name of the currency. Finally, this name is used by the the last sub query to retrieve the US dollar to currency ratio. It is then used to convert the US dollar into a currency of our desire which is in our case a Mexican Dollar(Mex\$). In this way, we can get our desired output in a single SQL query.

The advantage of using subquery:

- 1) It has made the query more simple and readable

- 2) It has reduced the execution time of our query
- 3) It enables the code reusability

## 9. SELECT (Screen Shot)



The screenshot shows the Live SQL interface with a sidebar on the left containing links to Home, SQL Worksheet, My Session, Schema, Quick SQL, My Scripts, My Tutorials, and Code Library. The main area is titled 'SQL Worksheet' and contains a SQL query:

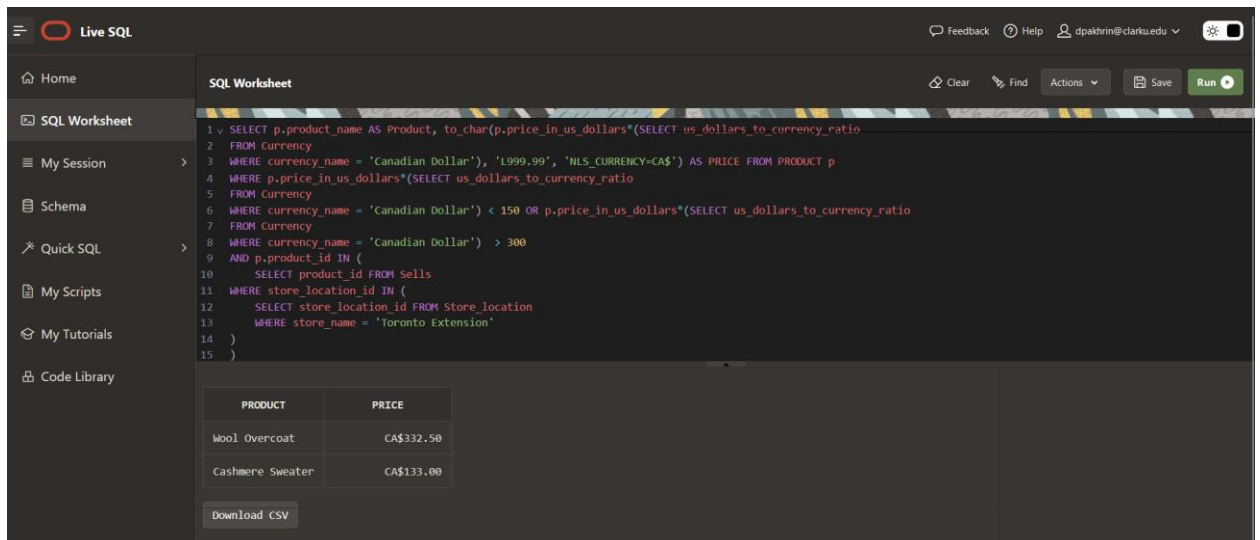
```
1 SELECT product_name,
2   to_char(price_in_us_dollars *
3   (SELECT us_dollars_to_currency_ratio
4   FROM Currency
5   WHERE currency_name = 'Mexican Peso'), 'L9,999.99', 'NLS_CURRENCY=Mex$ ') AS price_in_pesos
6 FROM Product
7 WHERE price_in_us_dollars *
8   (SELECT us_dollars_to_currency_ratio
9   FROM Currency
10  WHERE currency_name = 'Mexican Peso') < 2750;
```

Below the query, the results are displayed in a table:

PRODUCT_NAME	PRICE_IN_PESOS
Designer Jeans	Mex\$2,512.50
Flowing Skirt	Mex\$2,093.75
Cashmere Sweater	Mex\$1,675.00

A 'Download CSV' button is located below the table. At the bottom, it states '3 rows selected.'

## 10. SELECT (Screen Shot)



The screenshot shows the Live SQL interface with the same sidebar as the previous screenshot. The main area is titled 'SQL Worksheet' and contains a more complex SQL query:

```
1 SELECT p.product_name AS Product, to_char(p.price_in_us_dollars*(SELECT us_dollars_to_currency_ratio
2 FROM Currency
3 WHERE currency_name = 'Canadian Dollar'), 'L999.99', 'NLS_CURRENCY=CA$') AS PRICE FROM PRODUCT p
4 WHERE p.price_in_us_dollars*(SELECT us_dollars_to_currency_ratio
5 FROM Currency
6 WHERE currency_name = 'Canadian Dollar') < 150 OR p.price_in_us_dollars*(SELECT us_dollars_to_currency_ratio
7 FROM Currency
8 WHERE currency_name = 'Canadian Dollar') > 300
9 AND p.product_id IN (
10  SELECT product_id FROM Sells
11 WHERE store_location_id IN (
12  SELECT store_location_id FROM Store_location
13  WHERE store_name = 'Toronto Extension'
14 )
15 )
```

Below the query, the results are displayed in a table:

PRODUCT	PRICE
Wool Overcoat	CA\$332.50
Cashmere Sweater	CA\$133.00

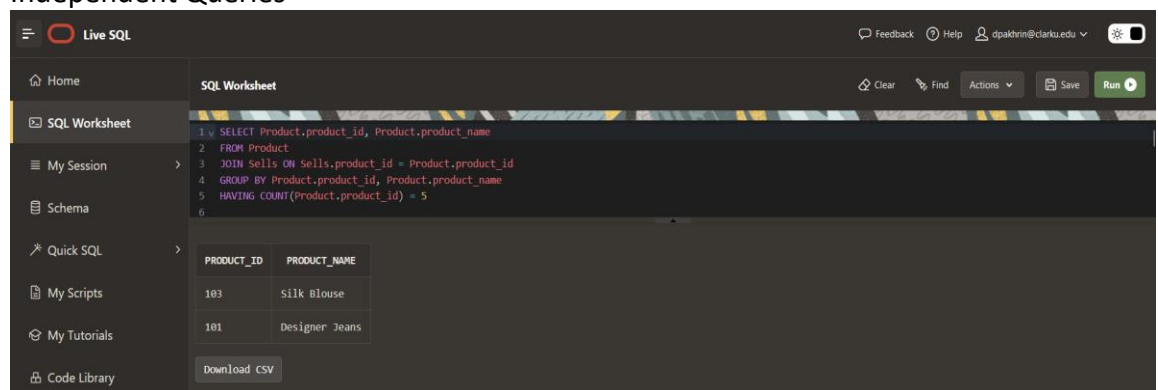
A 'Download CSV' button is located below the table.

11.

a. EXPLANATION

For the independent query, we can consider accessing the products id and product name that are available in every store location. For that we inner join the product table with the Sells table and group the rows by product id and product name which is available in every store(HAVING COUNT(Product.product\_id) = 5). Next independent query can be accessing the product name and the size of the products that are available in every store. Since the relation between the Product table and Size table is shown by Available In table, we join this on the Product table.

b. Independent Queries



The screenshot shows a SQL Worksheet interface with a query that filters products based on their availability across all five stores. The query is as follows:

```

1 SELECT Product.product_id, Product.product_name
2 FROM Product
3 JOIN Sells ON Sells.product_id = Product.product_id
4 GROUP BY Product.product_id, Product.product_name
5 HAVING COUNT(Product.product_id) = 5
6

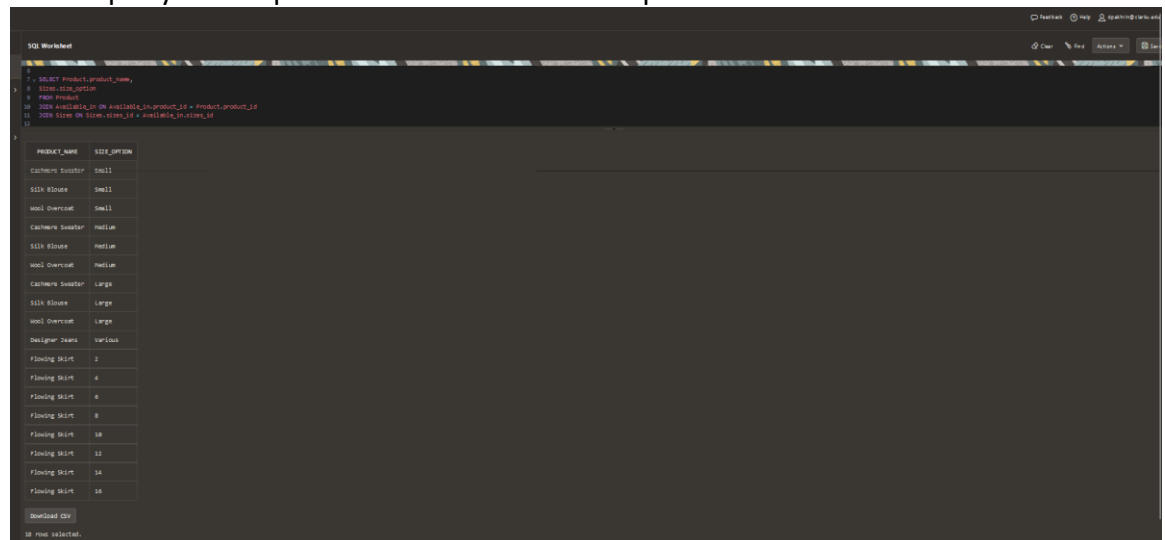
```

The result of the query is displayed in a table with two columns: PRODUCT\_ID and PRODUCT\_NAME.

PRODUCT_ID	PRODUCT_NAME
103	Silk Blouse
101	Designer Jeans

A "Download CSV" button is visible below the table.

This query fetches all the products that are sold by every store. Later, the result of this query will help to access the size of those products.



The screenshot shows a SQL Worksheet interface with a query that joins the Product table with the Size table to find the sizes of products available in all five stores. The query is as follows:

```

1 SELECT Product.product_name,
2       Size.size_option
3 FROM Product
4 JOIN Available_In ON Available_In.product_id = Product.product_id
5 JOIN Size ON Size.product_id = Available_In.size_id
6

```

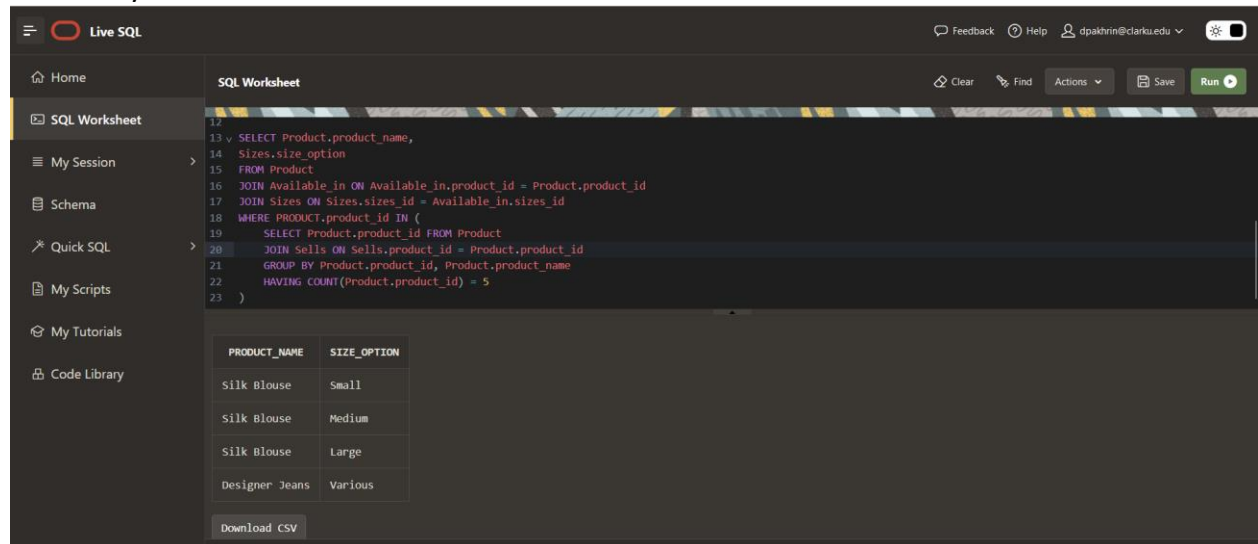
The result of the query is displayed in a table with two columns: PRODUCT\_NAME and SIZE\_OPTION.

PRODUCT_NAME	SIZE_OPTION
Cashmere Sweater	Small
Silk Blouse	Small
Wool Overcoat	Small
Cashmere Sweater	Medium
Silk Blouse	Medium
Wool Overcoat	Medium
Cashmere Sweater	Large
Silk Blouse	Large
Wool Overcoat	Large
Designer Jeans	Various
Flaming Skirt	2
Flaming Skirt	4
Flaming Skirt	6
Flaming Skirt	8
Flaming Skirt	10
Flaming Skirt	12
Flaming Skirt	14
Flaming Skirt	16

A "Download CSV" button is visible below the table, and a status message at the bottom indicates "18 rows selected".

This query fetches the sizes of all the available products. Later, this query is combined with another independent query to access our end result

### c. Full Query



The screenshot shows the Live SQL interface with the following SQL query:

```

12
13 v SELECT Product.product_name,
14     Sizes.size_option
15 FROM Product
16 JOIN Available_in ON Available_in.product_id = Product.product_id
17 JOIN Sizes ON Sizes.size_id = Available_in.size_id
18 WHERE Product.product_id IN (
19     SELECT Product.product_id FROM Product
20     JOIN Sells ON Sells.product_id = Product.product_id
21     GROUP BY Product.product_id, Product.product_name
22     HAVING COUNT(Product.product_id) = 5
23 )
  
```

The results table is as follows:

PRODUCT_NAME	SIZE_OPTION
Silk Blouse	Small
Silk Blouse	Medium
Silk Blouse	Large
Designer Jeans	Various

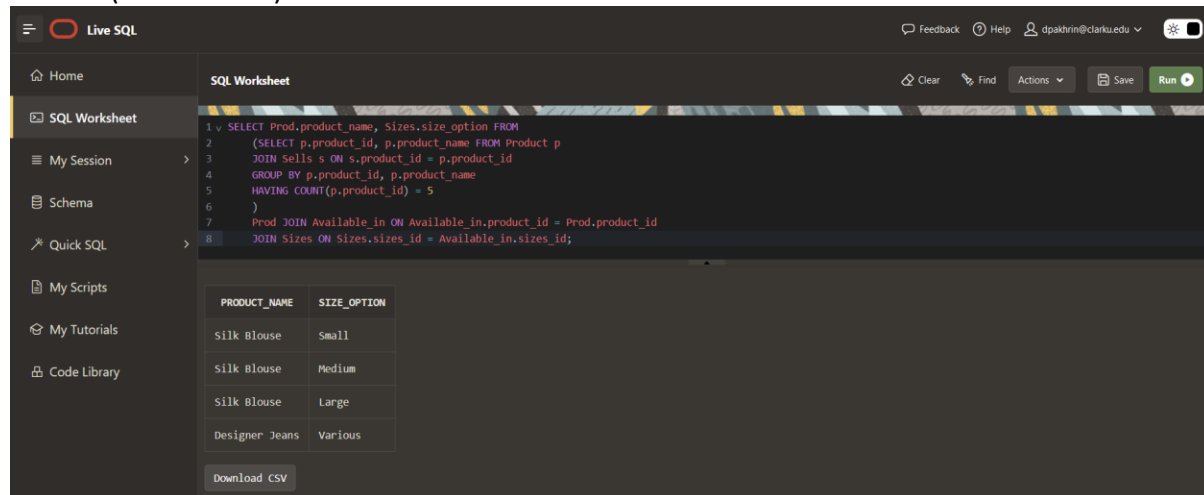
Download CSV

### d. EXPLANATION

Finally, we have combined both independent queries in a single stand-alone query. In the first part, we have our second independent query which fetches the product name and the size of the query. Here, we have added a clause which is where clause. It is because we just want the sizes and the name of the product which is available in every store location. This is returned by our first independent clause. It is done by comparing with the product id which is sold by all the store locations.

12.

### a. SELECT (Screen Shot)



The screenshot shows the Live SQL interface with the following SQL query:

```

1 v SELECT Prod.product_name, Sizes.size_option FROM
2     (SELECT p.product_id, p.product_name FROM Product p
3     JOIN Sells s ON s.product_id = p.product_id
4     GROUP BY p.product_id, p.product_name
5     HAVING COUNT(p.product_id) = 5
6     )
7 Prod JOIN Available_in ON Available_in.product_id = Prod.product_id
8 JOIN Sizes ON Sizes.size_id = Available_in.size_id;
  
```

The results table is as follows:

PRODUCT_NAME	SIZE_OPTION
Silk Blouse	Small
Silk Blouse	Medium
Silk Blouse	Large
Designer Jeans	Various

Download CSV



b. EXPLANATION

The subquery in the query mentioned above obtains the product name and product ID for each item that is offered at every store location. An alias that gives the result of the subquery a name is "Prod." This is the part of the join condition that merges the subquery result into the table that is already available. The FORM clause, instead of the WHERE clause, is where filtering is accomplished, as the subquery only returns products that meet Marcus's requirements.

13. Correlated Subquery vs Uncorrelated Subquery:

a. Correlated Subquery

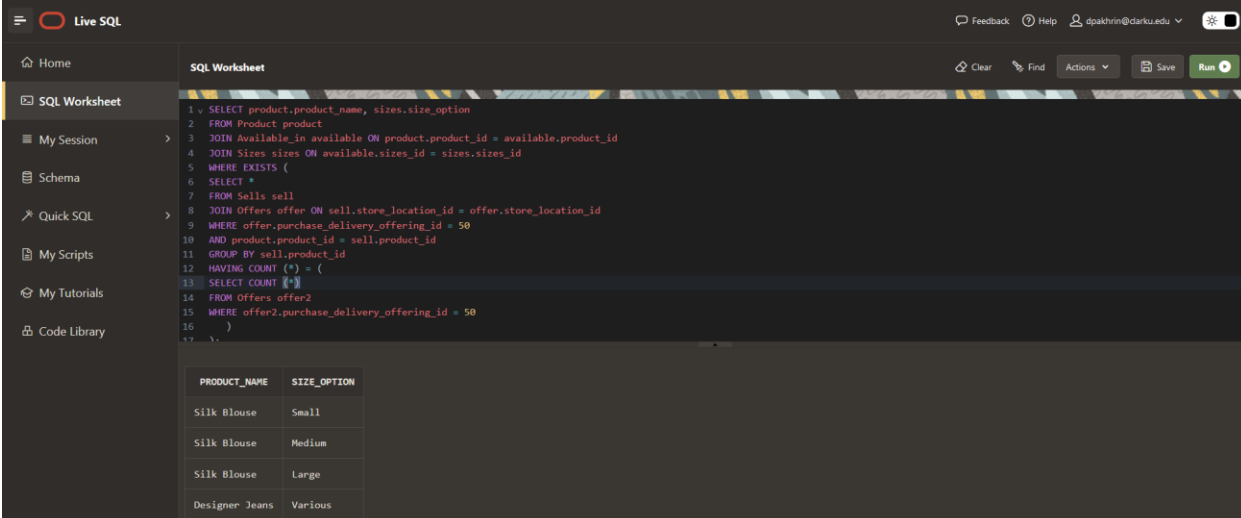
- This subquery relies on the outer/main query so that the subquery gets executed at least once for every row in the main query.
- It is joined with the Exists keyword.

b. Uncorrelated Subquery

This subquery is independent of the main query that can be executed without relying on the main query. It is only executed once and the output is passed to the main query.

14.

a. SELECT (Screen Shot)



The screenshot shows a web-based SQL editor interface titled "Live SQL". The interface includes a sidebar on the left with navigation options: Home, SQL Worksheet (selected), My Session, Schema, Quick SQL, My Scripts, My Tutorials, and Code Library. The main area displays a SQL query with line numbers 1 through 17. The query is a complex SELECT statement involving multiple joins and a subquery. Below the query editor, the results of the query are displayed in a table with two columns: PRODUCT\_NAME and SIZE\_OPTION. The results show five rows of data.

```
1 SELECT product.product_name, sizes.size_option
2 FROM Product product
3 JOIN Available_in available ON product.product_id = available.product_id
4 JOIN Sizes sizes ON available.sizes_id = sizes.sizes_id
5 WHERE EXISTS (
6   SELECT *
7   FROM Sells sell
8   JOIN Offers offer ON sell.store_location_id = offer.store_location_id
9   WHERE offer.purchase_delivery_offering_id = 50
10  AND product.product_id = sell.product_id
11  GROUP BY sell.product_id
12  HAVING COUNT (*) = (
13   SELECT COUNT (*)
14   FROM Offers offer2
15   WHERE offer2.purchase_delivery_offering_id = 50
16  )
17 )
```

PRODUCT_NAME	SIZE_OPTION
Silk Blouse	Small
Silk Blouse	Medium
Silk Blouse	Large
Designer Jeans	Various

b. EXPLANATION

The above query is the another approach of obtaining the data that is similar to the problem 11. Here we used the EXISTS keyword where we define our sub query. It gets executed at least once for every row in the main query. Also we matched our the product\_id of the outer query with the subquery (i.e) product.product\_id = sell.product\_id.

c. EXPLANATION

In my opinion, the query in problem 11 is more straightforward. The result obtained from the subquery is directly used by the outer query. In contrast, the query in this problem is complex and time-consuming as it gets executed at least once for every row in the main query.