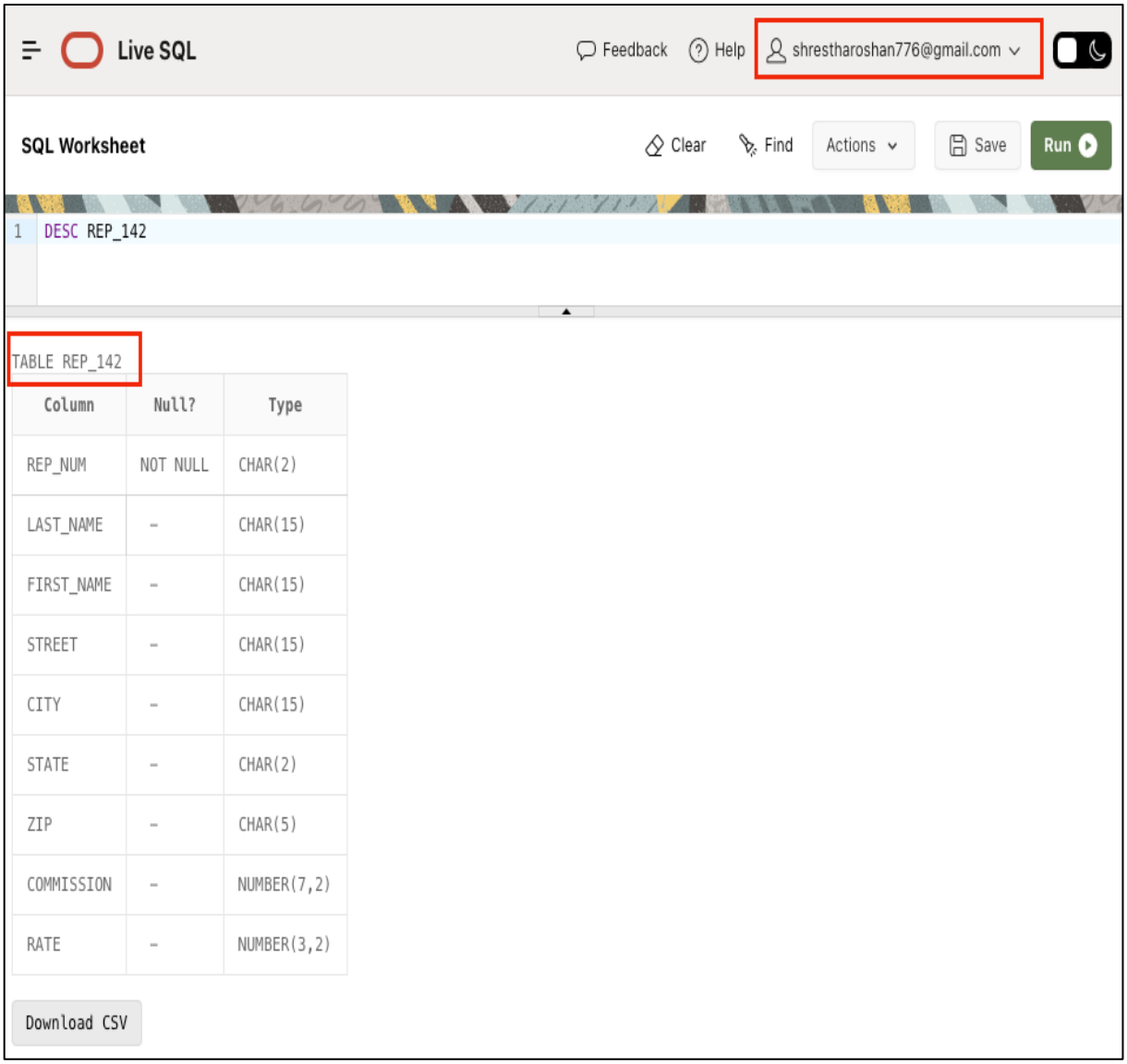
Database Design – 2023S
Student ID: 901142
Student Name: Roshan Shrestha
Practical Activity#8 – Sub Queries / Multiple Table Queries using Where.

The list of all the created tables after modifying and executing the SQL scripts provided are listed below as following:

Table: REP_142



Column	Null?	Type
REP_NUM	NOT NULL	CHAR(2)
LAST_NAME	-	CHAR(15)
FIRST_NAME	-	CHAR(15)
STREET	-	CHAR(15)
CITY	-	CHAR(15)
STATE	-	CHAR(2)
ZIP	-	CHAR(5)
COMMISSION	-	NUMBER(7,2)
RATE	-	NUMBER(3,2)

Table: CUSTOMER_142

Live SQL | Feedback | Help | shrestharoshan776@gmail.com

SQL Worksheet | Clear | Find | Actions | Save | Run

```
1 DESC CUSTOMER_142
```

TABLE CUSTOMER_142

Column	Null?	Type
CUSTOMER_NUM	NOT NULL	CHAR(3)
CUSTOMER_NAME	NOT NULL	CHAR(35)
STREET	-	CHAR(15)
CITY	-	CHAR(15)
STATE	-	CHAR(2)
ZIP	-	CHAR(5)
BALANCE	-	NUMBER(8,2)
CREDIT_LIMIT	-	NUMBER(8,2)
REP_NUM	-	CHAR(2)

Table: ORDERS_142

Live SQL | Feedback | Help | shrestharoshan776@gmail.com

SQL Worksheet | Clear | Find | Actions | Save | Run

```
1 DESC ORDERS_142
```

TABLE ORDERS_142

Column	Null?	Type
ORDER_NUM	NOT NULL	CHAR(5)
ORDER_DATE	-	DATE
CUSTOMER_NUM	-	CHAR(3)

Table: PART_142

Live SQL Feedback Help shrestharoshan776@gmail.com

SQL Worksheet Clear Find Actions Save Run

```
1  DESC PART_142
```

TABLE PART_142

Column	Null?	Type
PART_NUM	NOT NULL	CHAR(4)
DESCRIPTION	-	CHAR(15)
ON_HAND	-	NUMBER(4,0)
CLASS	-	CHAR(2)
WAREHOUSE	-	CHAR(1)
PRICE	-	NUMBER(6,2)

Table: ORDER_LINE_142

Live SQL Feedback Help shrestharoshan776@gmail.com

SQL Worksheet Clear Find Actions Save Run

```
1  DESC ORDER_LINE_142
```

TABLE ORDER_LINE_142

Column	Null?	Type
ORDER_NUM	NOT NULL	CHAR(5)
PART_NUM	NOT NULL	CHAR(4)
NUM_ORDERED	-	NUMBER(3,0)
QUOTED_PRICE	-	NUMBER(6,2)

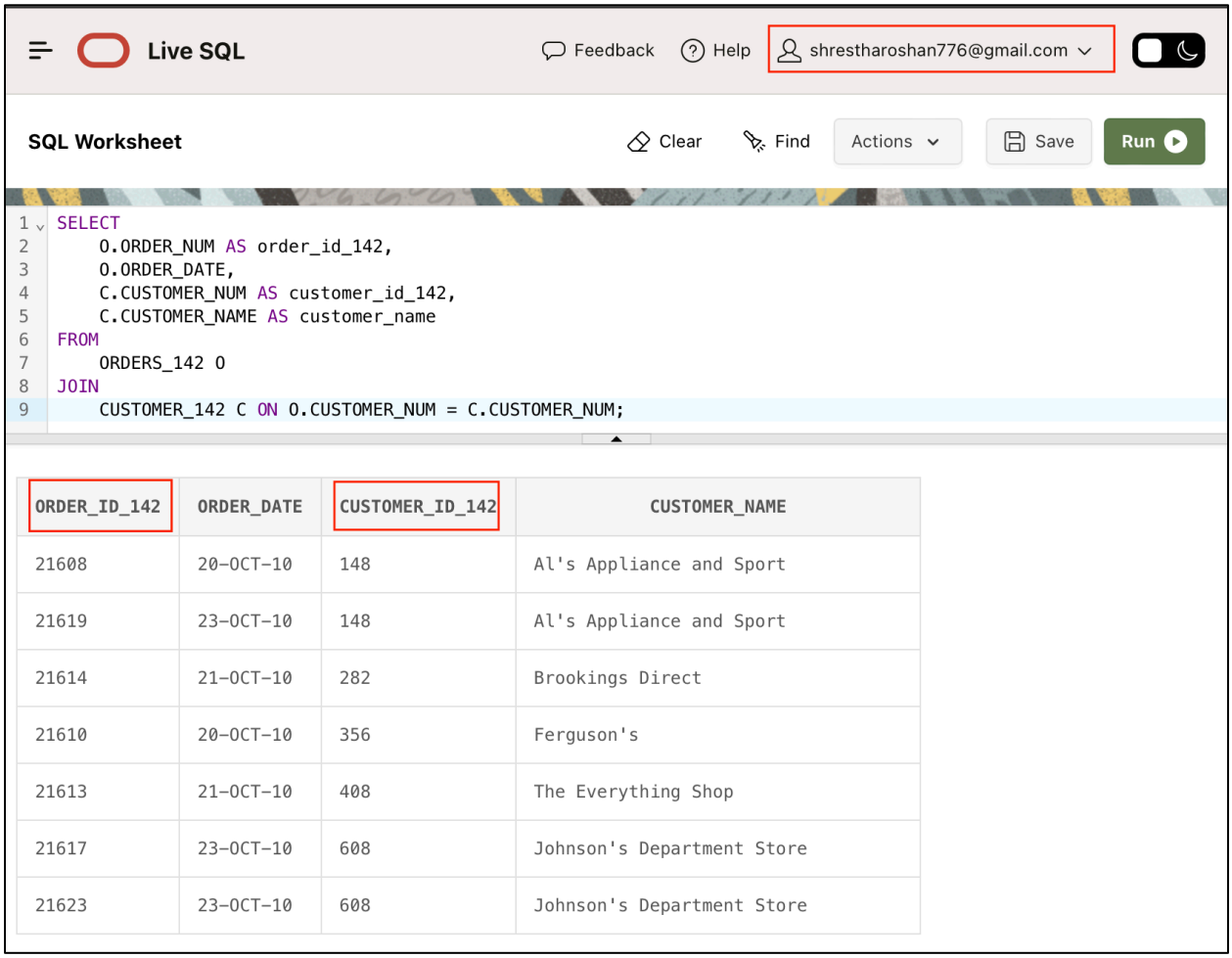
Practical Activity#1

1. For each order, list the order number and order date along with the number and name of the customer that placed the order.

To retrieve the above mentioned data, we can use the script below:

```
SELECT
        O.ORDER_NUM AS order_id_142,
        O.ORDER_DATE,
        C.CUSTOMER_NUM AS customer_id_142,
        C.CUSTOMER_NAME AS customer_name
FROM
        ORDERS_142 O
JOIN
        CUSTOMER_142 C ON O.CUSTOMER_NUM = C.CUSTOMER_NUM;
```

This SQL query retrieves data from two tables, **ORDERS_142** and **CUSTOMER_142**, using a **JOIN** operation. It selects specific columns from each table and aliases them as **order_id_142**, **ORDER_DATE**, **customer_id_142**, and **customer_name**. The **JOIN** is performed based on the matching **CUSTOMER_NUM** in both tables. The output of the query is as below:



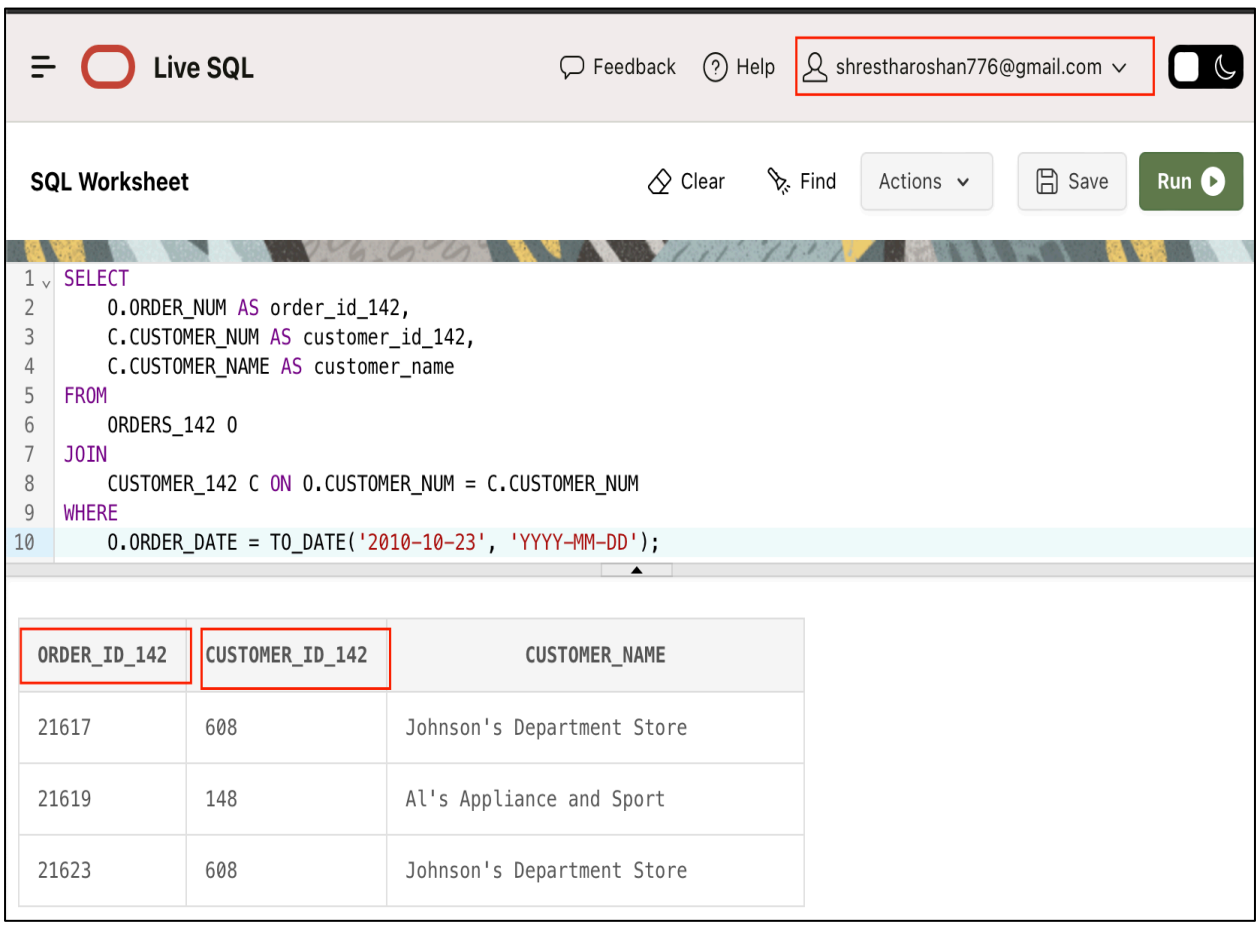
ORDER_ID_142	ORDER_DATE	CUSTOMER_ID_142	CUSTOMER_NAME
21608	20-OCT-10	148	Al's Appliance and Sport
21619	23-OCT-10	148	Al's Appliance and Sport
21614	21-OCT-10	282	Brookings Direct
21610	20-OCT-10	356	Ferguson's
21613	21-OCT-10	408	The Everything Shop
21617	23-OCT-10	608	Johnson's Department Store
21623	23-OCT-10	608	Johnson's Department Store

2. For each order placed on October 23, 2010, list the order number along with the number and name of the customer that placed the order.

To get the list of order that was placed on October 23,2010, we can we the SQL script below:

```
SELECT
        O.ORDER_NUM AS order_id_142,
        C.CUSTOMER_NUM AS customer_id_142,
        C.CUSTOMER_NAME AS customer_name
FROM
        ORDERS_142 O
JOIN
        CUSTOMER_142 C ON O.CUSTOMER_NUM = C.CUSTOMER_NUM
WHERE
        O.ORDER_DATE = TO_DATE('2010-10-23', 'YYYY-MM-DD');
```

This SQL query performs a **JOIN** operation on the tables **ORDERS_142** and **CUSTOMER_142** to get specified columns (**order_id_142**, **customer_id_142**, and **customer_name**). Based on the matching **CUSTOMER_NUM** in both tables, the **JOIN** is carried out. The **WHERE** clause in the query limits the records that are displayed to those where the **ORDER_DATE** equals **"2010-10-23"**. The result of the query is below:



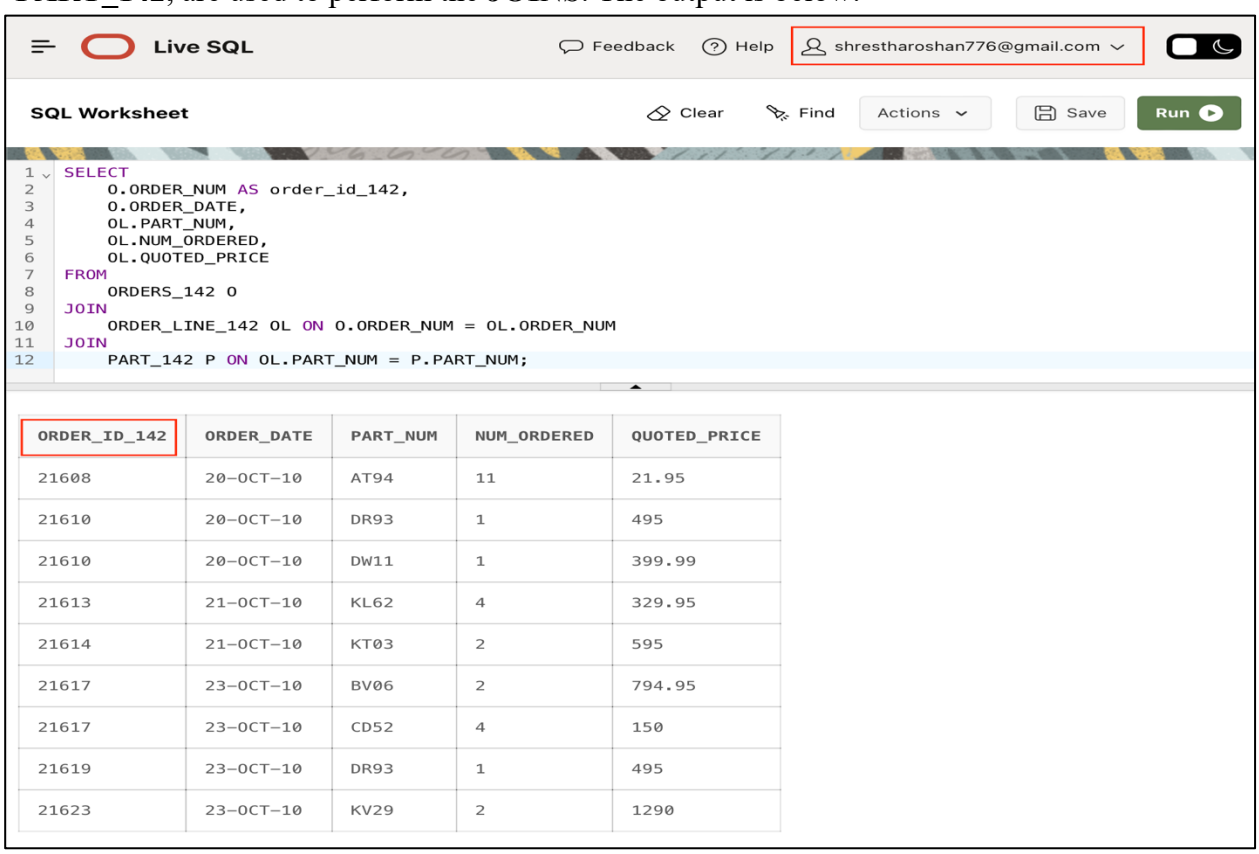
ORDER_ID_142	CUSTOMER_ID_142	CUSTOMER_NAME
21617	608	Johnson's Department Store
21619	148	Al's Appliance and Sport
21623	608	Johnson's Department Store

3. For each order, list the order number, order date, part number, number of units ordered, and quoted price for each order line that makes up the order. (Hint :order , orderline).

To get the above mentioned set of data we can execute the query below:

```
SELECT
        O.ORDER_NUM AS order_id_142,
        O.ORDER_DATE,
        OL.PART_NUM,
        OL.NUM_ORDERED,
        OL.QUOTED_PRICE
FROM
        ORDERS_142 O
JOIN
        ORDER_LINE_142 OL ON O.ORDER_NUM = OL.ORDER_NUM
JOIN
        PART_142 P ON OL.PART_NUM = P.PART_NUM;
```

Using two **JOIN** procedures, this SQL query pulls information from the three tables **ORDERS_142**, **ORDER_LINE_142**, and **PART_142**. Order_Num (also known as **order_id_142**), Order Date, Part Number, Number Ordered, and Quoted Price are the columns that are chosen. The matching **ORDER_NUM** between **ORDERS_142** and **ORDER_LINE_142**, as well as the matching **PART_NUM** between **ORDER_LINE_142** and **PART_142**, are used to perform the **JOINS**. The output is below:



ORDER_ID_142	ORDER_DATE	PART_NUM	NUM_ORDERED	QUOTED_PRICE
21608	20-OCT-10	AT94	11	21.95
21610	20-OCT-10	DR93	1	495
21610	20-OCT-10	DW11	1	399.99
21613	21-OCT-10	KL62	4	329.95
21614	21-OCT-10	KT03	2	595
21617	23-OCT-10	BV06	2	794.95
21617	23-OCT-10	CD52	4	150
21619	23-OCT-10	DR93	1	495
21623	23-OCT-10	KV29	2	1290

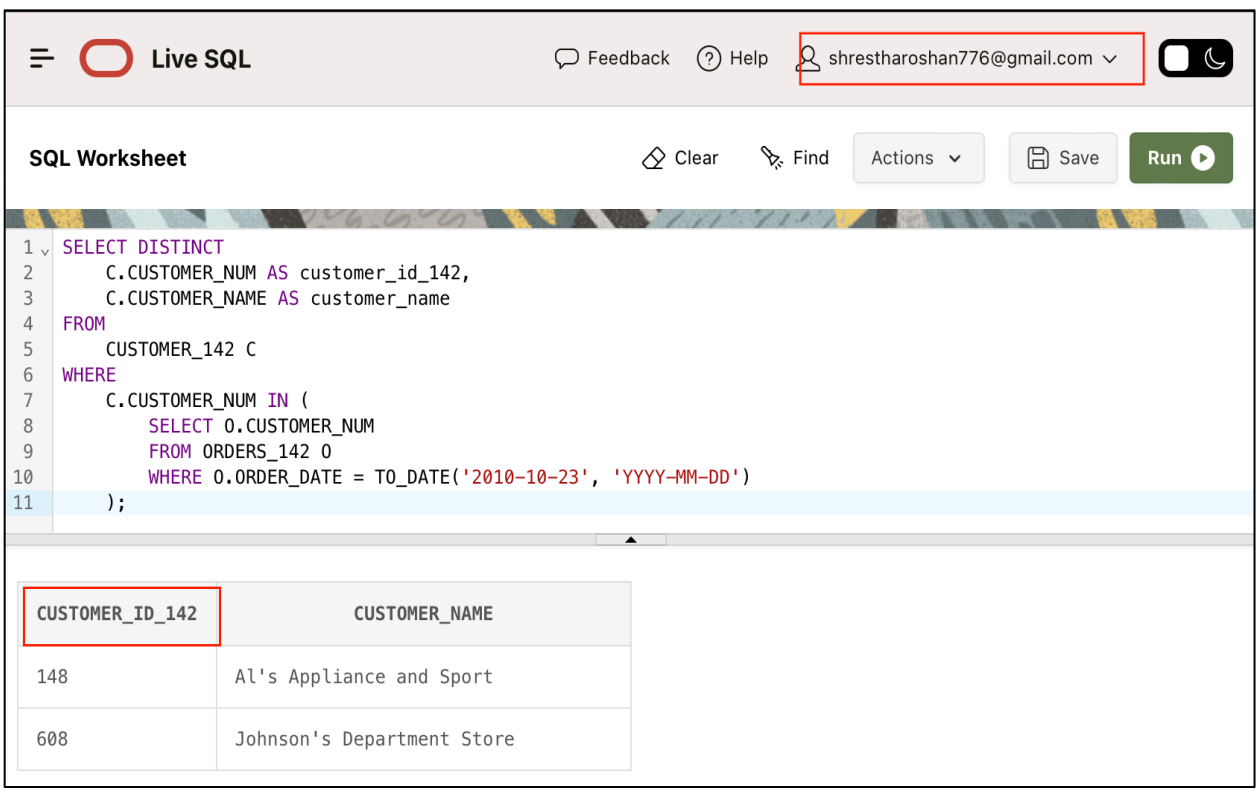
Practical Activity#2

4. Use the IN operator to find the number and name of each customer that placed an order on October 23, 2010.

We can use the query below to find the number and name of each customer that places an order on October 23, 2010, using **IN** operator:

```
SELECT DISTINCT
        C.CUSTOMER_NUM AS customer_id_142,
        C.CUSTOMER_NAME AS customer_name
FROM
        CUSTOMER_142 C
WHERE
        C.CUSTOMER_NUM IN (
        SELECT O.CUSTOMER_NUM
        FROM ORDERS_142 O
        WHERE O.ORDER_DATE = TO_DATE('2010-10-23', 'YYYY-MM-DD') );
```

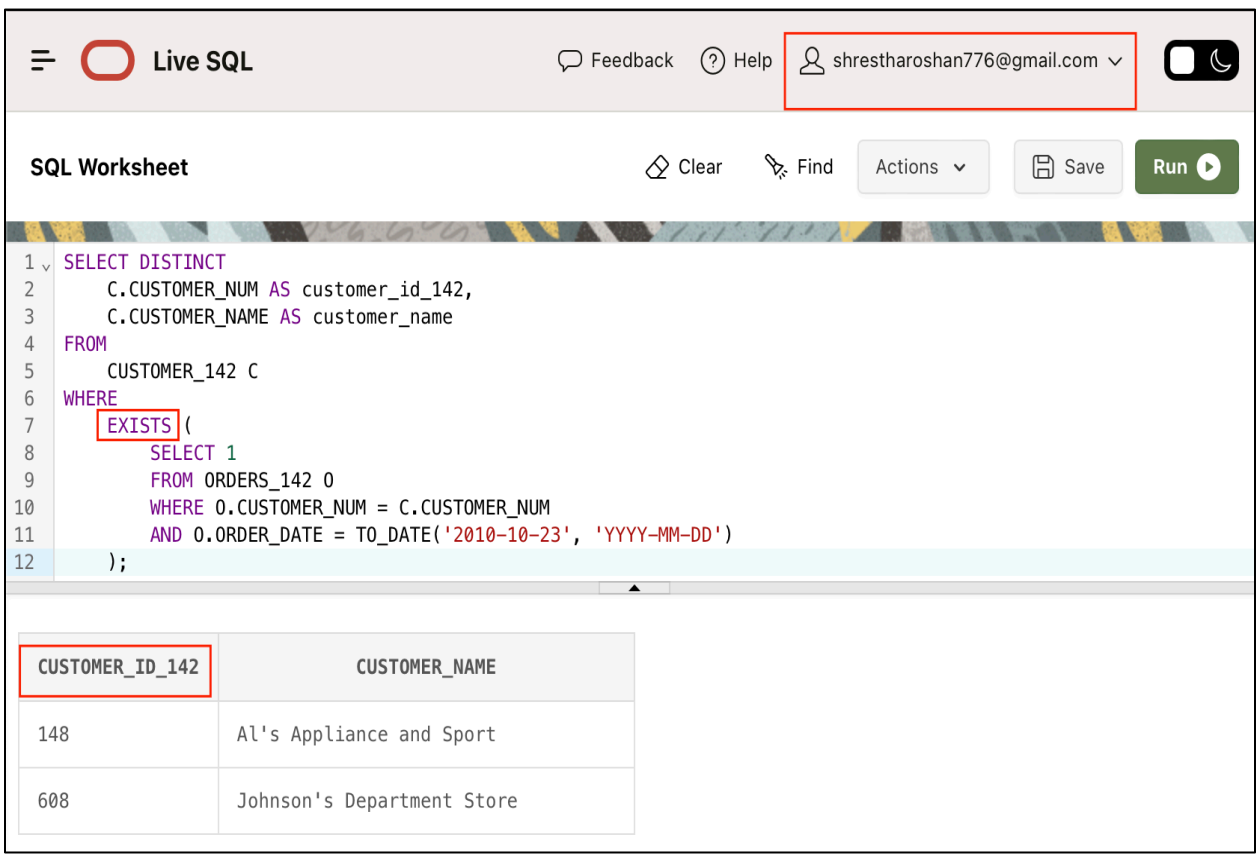
The **CUSTOMER_142** table is queried using SQL to retrieve specific customer data. Columns **CUSTOMER_NUM** (also known as **customer_id_142**) and **CUSTOMER_NAME** is chosen. The **WHERE** clause in the query limits the list of customers in the results to those whose **CUSTOMER_NUM** appears in the list of **CUSTOMER_NUM** values from the **ORDERS_142** table where the **ORDER_DATE** is **2010-10-23**. Simply said, it retrieves specific customer information for consumers who made an order on the designated date. The output from executing the query is below:



5. Repeat Exercise 4, but this time use the EXISTS operator in your answer.
Here we are going to find the number and name of each customer that places an order on October 23, 2010, but this time using the **EXISTS** operator, the SQL query can be found below:

```
SELECT DISTINCT
        C.CUSTOMER_NUM AS customer_id_142,
         C.CUSTOMER_NAME AS customer_name
FROM
        CUSTOMER_142 C
WHERE
        EXISTS (
        SELECT 1
        FROM ORDERS_142 O
        WHERE O.CUSTOMER_NUM = C.CUSTOMER_NUM
        AND O.ORDER_DATE = TO_DATE('2010-10-23', 'YYYY-MM-DD'));
```

The **CUSTOMER_142** table is queried using SQL to retrieve specific customer data. Columns **CUSTOMER_NUM** (also known as **customer_id_142**) and **CUSTOMER_NAME** are chosen. The query has an **EXISTS** subquery and a **WHERE** clause. The **EXISTS** subquery determines whether the **ORDERS_142** table contains at least one record with the **CUSTOMER_NUMBER** matching the one in the primary query and the **ORDER_DATE** being **2010-10-23** Simply said, it retrieves specific customer information for consumers who made an order on the designated date. The output is as below:

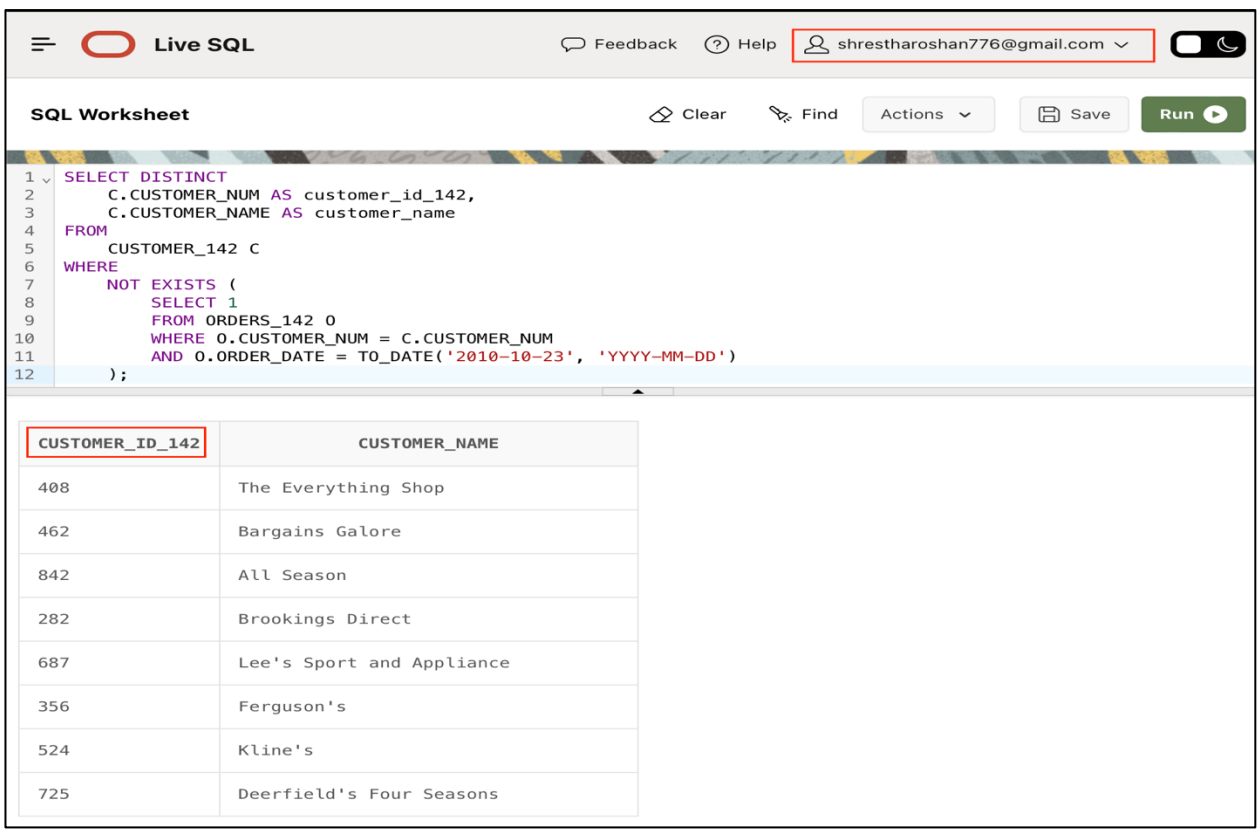


6. Find the number and name of each customer that did not place an order on October 23, 2010.

In order to retrieve the data including name and number of each customer that did not place an order on October 23, 2010, we can use the below SQL query:

```
SELECT DISTINCT
        C.CUSTOMER_NUM AS customer_id_142,
         C.CUSTOMER_NAME AS customer_name
FROM
        CUSTOMER_142 C
WHERE
        NOT EXISTS (
        SELECT 1
        FROM ORDERS_142 O
        WHERE O.CUSTOMER_NUM = C.CUSTOMER_NUM
        AND O.ORDER_DATE = TO_DATE('2010-10-23', 'YYYY-MM-DD'));
```

The **CUSTOMER_142** table is queried for unique customer information in this SQL query. Columns **CUSTOMER_NUM** (aliased as **customer_id_142**) and **CUSTOMER_NAME** is chosen. The query has a **WHERE** clause as well as a **NOT EXISTS** subquery. The **NOT EXISTS** subquery checks to see if there are any records in the **ORDERS_142** table with the same **CUSTOMER_NUM** as the main query and the **ORDER_DATE** of **2010-10-23**. In a nutshell, it retrieves unique customer information for consumers who did not place a purchase on the provided date. The output can be visualized below:



CUSTOMER_ID_142	CUSTOMER_NAME
408	The Everything Shop
462	Bargains Galore
842	All Season
282	Brookings Direct
687	Lee's Sport and Appliance
356	Ferguson's
524	Kline's
725	Deerfield's Four Seasons

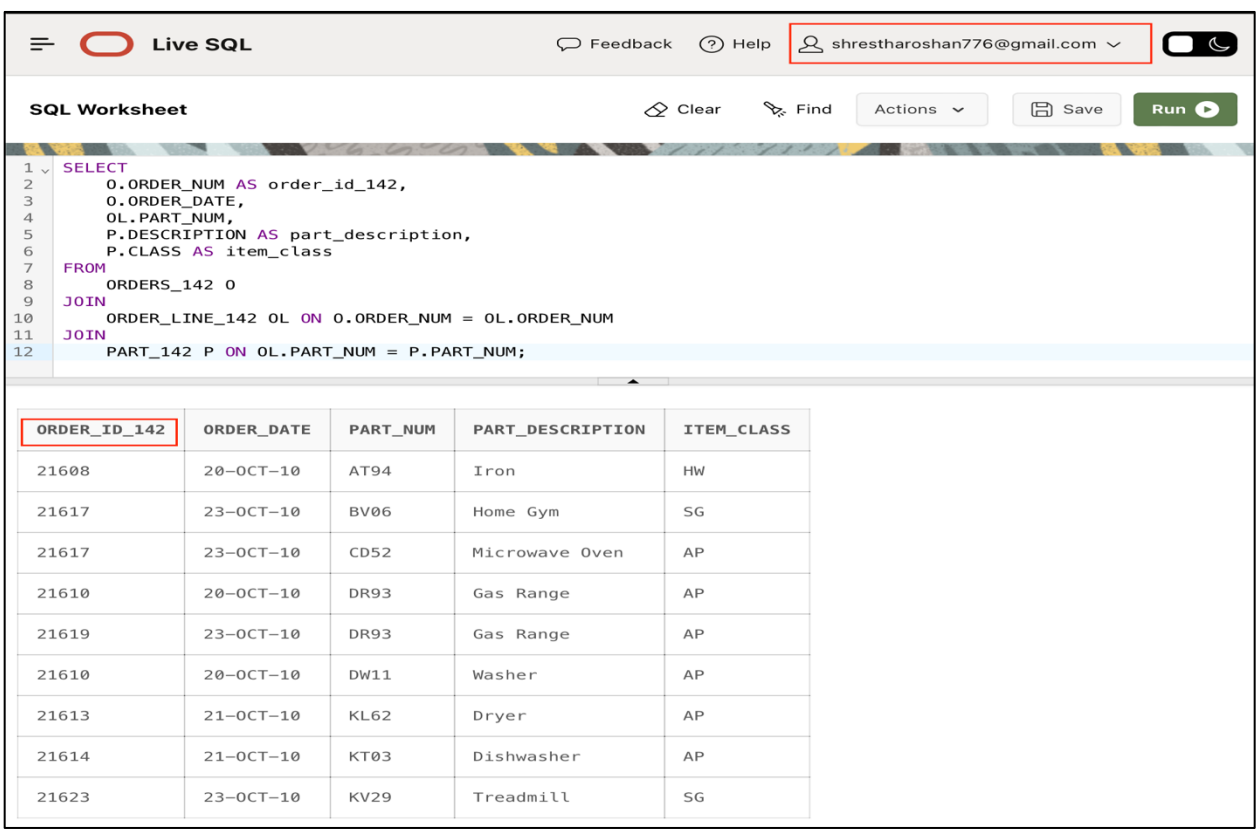
Practical Activity#3

7. For each order, list the order number, order date, part number, part description, and item class for each part that makes up the order.

To get the desired data we can use the query below:

```
SELECT
        O.ORDER_NUM AS order_id_142,
        O.ORDER_DATE,
        OL.PART_NUM,
        P.DESCRIPTION AS part_description,
        P.CLASS AS item_class
FROM
        ORDERS_142 O
JOIN
        ORDER_LINE_142 OL ON O.ORDER_NUM = OL.ORDER_NUM
JOIN
        PART_142 P ON OL.PART_NUM = P.PART_NUM;
```

Using **JOIN** procedures, this SQL query obtains specified columns from three tables: **ORDERS_142**, **ORDER_LINE_142**, and **PART_142**. It pulls **ORDER_NUM** (aliased as **order_id_142**), **ORDER_DATE**, and **PART_NUM** from the **PART_142** database, as well as **DESCRIPTION** (aliased as **part_description**) and **CLASS** (aliased as **item_class**). Matching **ORDER_NUM** between **ORDERS_142** and **ORDER_LINE_142**, as well as matching **PART_NUM** between **ORDER_LINE_142** and **PART_142**, is used to perform the **JOINS**. For each order line, it integrates order information with corresponding part details. The output is below:

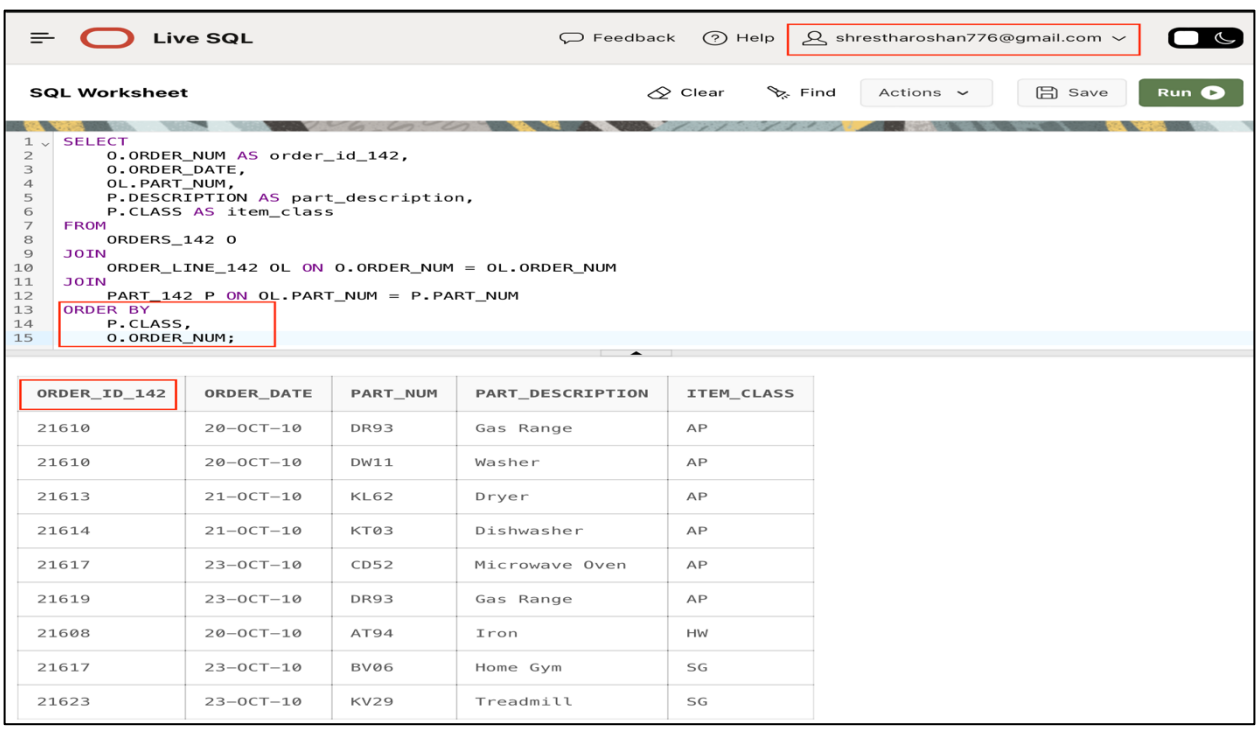


ORDER_ID_142	ORDER_DATE	PART_NUM	PART_DESCRIPTION	ITEM_CLASS
21608	20-OCT-10	AT94	Iron	HW
21617	23-OCT-10	BV06	Home Gym	SG
21617	23-OCT-10	CD52	Microwave Oven	AP
21610	20-OCT-10	DR93	Gas Range	AP
21619	23-OCT-10	DR93	Gas Range	AP
21610	20-OCT-10	DW11	Washer	AP
21613	21-OCT-10	KL62	Dryer	AP
21614	21-OCT-10	KT03	Dishwasher	AP
21623	23-OCT-10	KV29	Treadmill	SG

8. Repeat Exercise 7, but this time order the rows by item class and then by order number.
We can achieve the similar result as for exercise 7, and order the rows by item class and then by order number using the script below:

```
SELECT
   O.ORDER_NUM AS order_id_142,
   O.ORDER_DATE,
   OL.PART_NUM,
   P.DESCRIPTION AS part_description,
   P.CLASS AS item_class
FROM
   ORDERS_142 O
JOIN
   ORDER_LINE_142 OL ON O.ORDER_NUM = OL.ORDER_NUM
JOIN
   PART_142 P ON OL.PART_NUM = P.PART_NUM
ORDER BY
   P.CLASS,
   O.ORDER_NUM;
```

Using **JOIN** procedures, this SQL query obtains specified columns from the three tables **ORDERS_142**, **ORDER_LINE_142**, and **PART_142**. From the **PART_142** table, it chooses **ORDER_NUM** (also known as **order_id_142**), **ORDER_DATE**, **PART_NUM**, and also inserts **DESCRIPTION** (also known as **part_description**) and **CLASS** (also known as **item_class**). Both matching **ORDER_NUM** between **ORDERS_142** and **ORDER_LINE_142** and matching **PART_NUM** between **ORDER_LINE_142** and **PART_142** are used to conduct the **JOINS**. Following that, the query sorts the outcomes by **order_id_142** and **item_class** in ascending order. In essence, it receives order and part information, combines it, and then groups the results by item class and order number, in that order. The output is below:



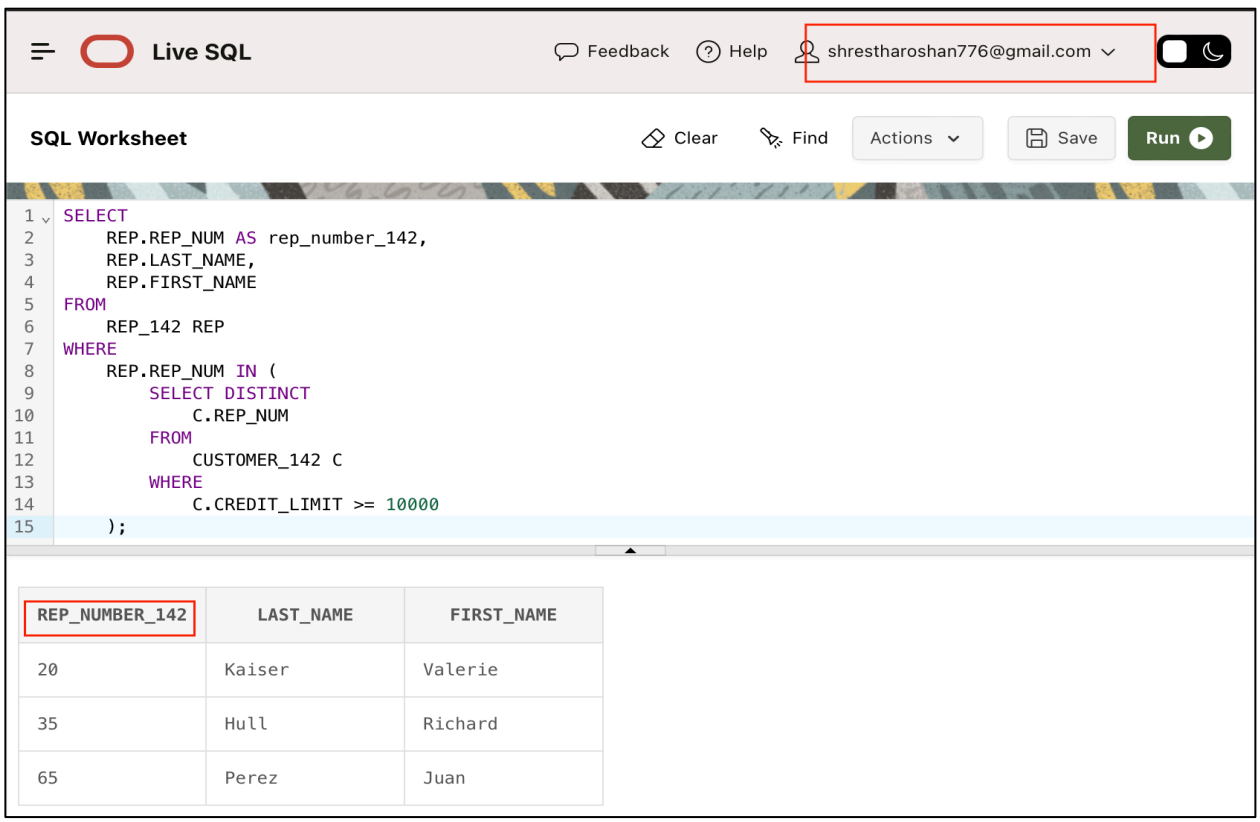
ORDER_ID_142	ORDER_DATE	PART_NUM	PART_DESCRIPTION	ITEM_CLASS
21610	20-OCT-10	DR93	Gas Range	AP
21610	20-OCT-10	DW11	Washer	AP
21613	21-OCT-10	KL62	Dryer	AP
21614	21-OCT-10	KT03	Dishwasher	AP
21617	23-OCT-10	CD52	Microwave Oven	AP
21619	23-OCT-10	DR93	Gas Range	AP
21608	20-OCT-10	AT94	Iron	HW
21617	23-OCT-10	BV06	Home Gym	SG
21623	23-OCT-10	KV29	Treadmill	SG

Practical Activity#4

9. Use a subquery to find the rep number, last name, and first name of each sales rep who represents at least one customer with a credit limit of $10,000. List each sales rep only once in the results.

```
SELECT
   REP.REP_NUM AS rep_number_142,
   REP.LAST_NAME,
   REP.FIRST_NAME
FROM
   REP_142 REP
WHERE
   REP.REP_NUM IN (
      SELECT DISTINCT
         C.REP_NUM
      FROM
         CUSTOMER_142 C
      WHERE
         C.CREDIT_LIMIT >= 10000);
```

From **the REP_142** table, this SQL query retrieves representative data. It chooses the columns **LAST_NAME**, **FIRST_NAME**, and **REP_NUM** (also known as **rep_number_142**). The subquery that verifies if the **REP_NUM** is present in the list of different **REP_NUM** values from the **CUSTOMER_142** database where the **CREDIT_LIMIT** is more than or equal to 10000 is included in the query's **WHERE** clause. It retrieves representative information for those who have clients with a credit limit of at least $10,000, to put it briefly.

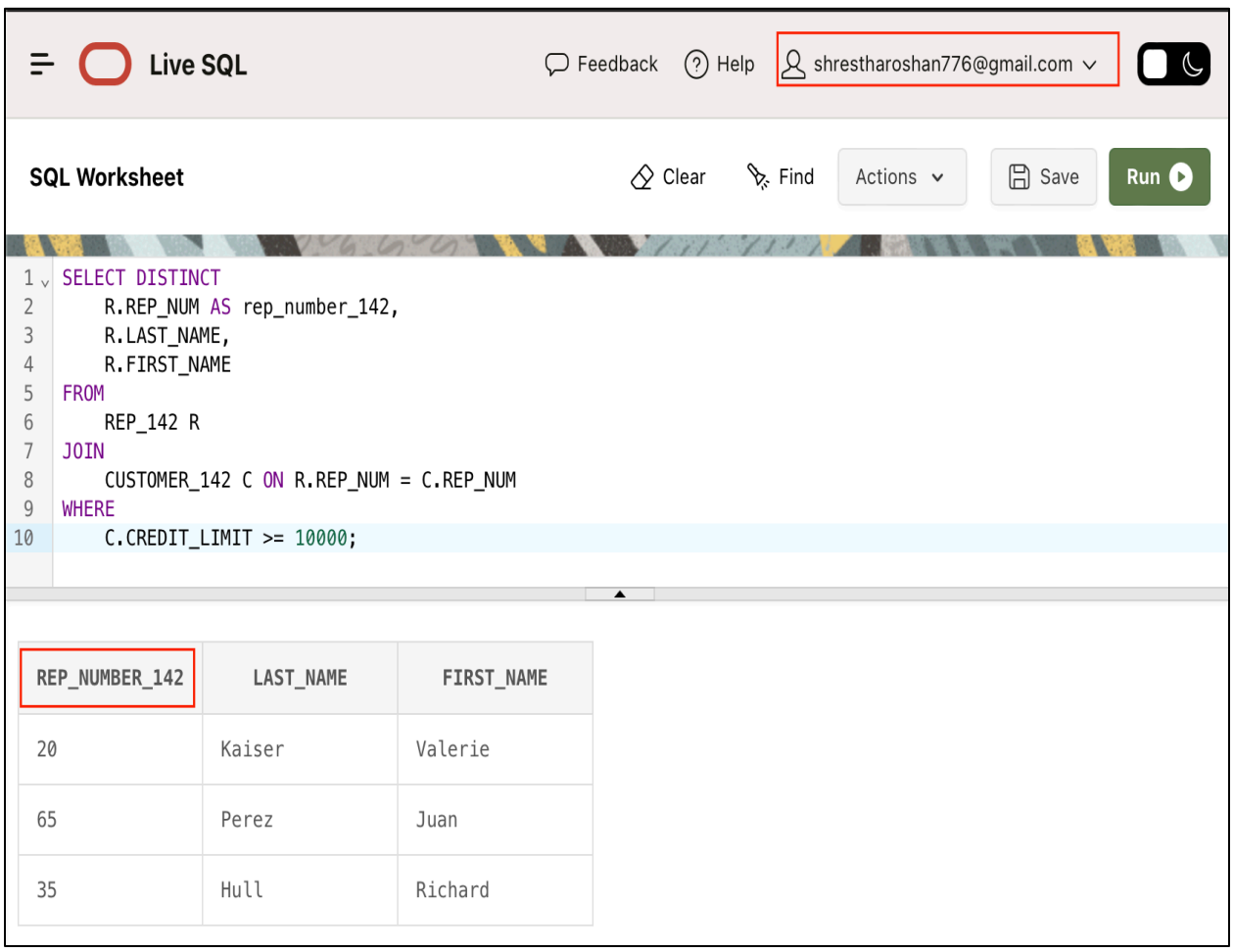


10. Repeat Exercise 9, but this time do not use a subquery.

The same exercise 9 without using subquery can be represented by below SQL script:

```
SELECT DISTINCT
   R.REP_NUM AS rep_number_142,
   R.LAST_NAME,
   R.FIRST_NAME
FROM
   REP_142 R
JOIN
   CUSTOMER_142 C ON R.REP_NUM = C.REP_NUM
WHERE
   C.CREDIT_LIMIT >= 10000;
```

This SQL query obtains representative data from the **REP_142** table by specifying different **REP_NUM** (aliased as **rep_number_142**), **LAST_NAME**, and **FIRST_NAME** values. Based on the matching **REP_NUM**, it executes a **JOIN** transaction with the **CUSTOMER_142** table. A **WHERE** clause is used to restrict the results to reveal only reps who have customers with a credit limit of at least $10,000. In summary, it retrieves representative information for those who have customers with credit limits of $10,000 or greater. The output is below:



REP_NUMBER_142	LAST_NAME	FIRST_NAME
20	Kaiser	Valerie
65	Perez	Juan
35	Hull	Richard