

Assignment 1

DBAS 3018 – Data Movement and Integration



January 29, 2021

jamie lu

W0441213

1. The guys in shipping want to know what quantities are being sold each month for each piece of inventory, so they can pre-order inventory from the suppliers.

**SQL STATEMENT:**

-- creating a table that has all the base level information

-- people in shipping can now run queries against the provided information or the base level information

CREATE TABLE SALESQTYINFO AS (

SELECT OL\_QUANTITY,

INV\_ID,

O\_DATE

FROM ORDER\_LINE OL JOIN ORDERS O ON (OL.O\_ID = O.O\_ID)

)

-- selecting all information from SALESQTYINFO

SELECT \* FROM SALESQTYINFO

-- selecting the date of the month and adding up the total quantity sold each day of the month, and the number of orders

SELECT

TO\_CHAR(O\_DATE, 'DD-MON') AS Date\_Month,

SUM(OL\_QUANTITY) AS Total\_Number\_Sold,

COUNT(O\_DATE) AS Total\_Number\_of\_Orders

FROM SALESQTYINFO

GROUP BY O\_DATE

ORDER BY TO\_CHAR(O\_DATE, 'DD-MON') DESC;

**EXPLANATION:**

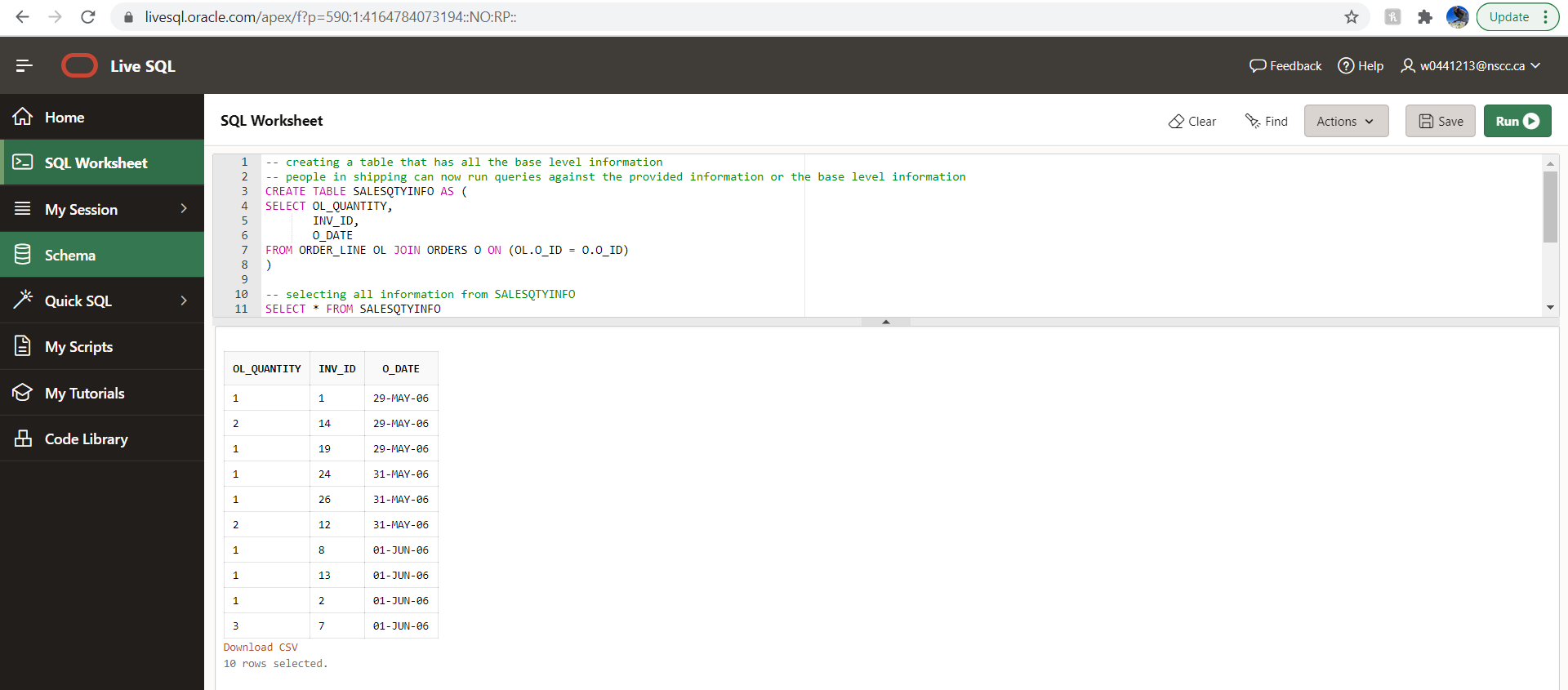
As described in the code, I created a table so that the guys in shipping would have access to all the information they require, which is the base level information. This way, it can be used for future queries without having difficulty.

I have shown a query showing the entirety of the SALESQTYINFO table that was created.

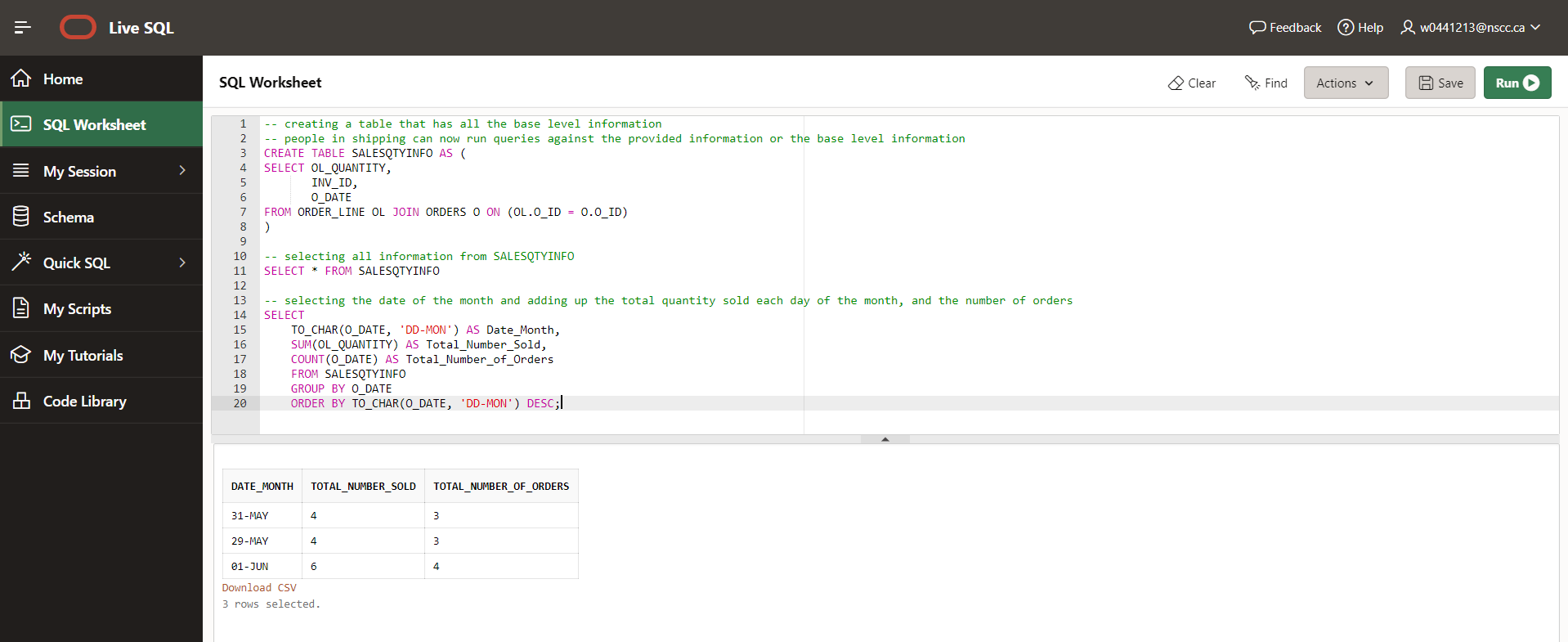
I have created a query wherein the date of the month is shown, along with the total quantity that was sold for that specified date of the month, as well as the total number of orders.

**SCREENSHOT:**

Below is a screenshot of the create table sql code and output as well as the query for showing all the contents of the new table created.



Below is the screenshot of the code and output of the second query created.



1. The guys in shipping also want to know if, for an inventory, when they order it does it usually arrive when they expect it? Or does it come early, or late? Hints: this is a tough one. (1) later date - earlier date gives the number of days between as a real number, so round it to an integer. (2) you don't have shipment\_lines for all of your inv\_ids. (3) you may want to test this as a query, then use it as a subquery in the FROM clause so that for each inv\_id, you know on average how reliable the ship\_date\_expected is.

**SQL STATEMENT:**

SELECT sd.ship\_id,

TO\_CHAR(ship\_date\_expected, 'Month') Mnth,

sd.inv\_id,

CASE

WHEN sd.diff IS NULL THEN 'Late, not arrived'

WHEN sd.diff = 0 THEN 'On time'

WHEN sd.diff > 0 THEN 'Late'

WHEN sd.diff < 0 THEN 'Arrived early'

END AS ETA

--include subquery in FROM statement. Subquery finds difference between expected and actual shipping date.

FROM (SELECT s.ship\_id AS ship\_id,

inv\_id,

ROUND(sl\_date\_received – ship\_date\_expected) AS diff

FROM shipment s JOIN shipment\_line sl ON (s.ship\_id = sl.ship\_id)) sd JOIN shipment ship –Alias as table to call in CASE as part of main SELECT

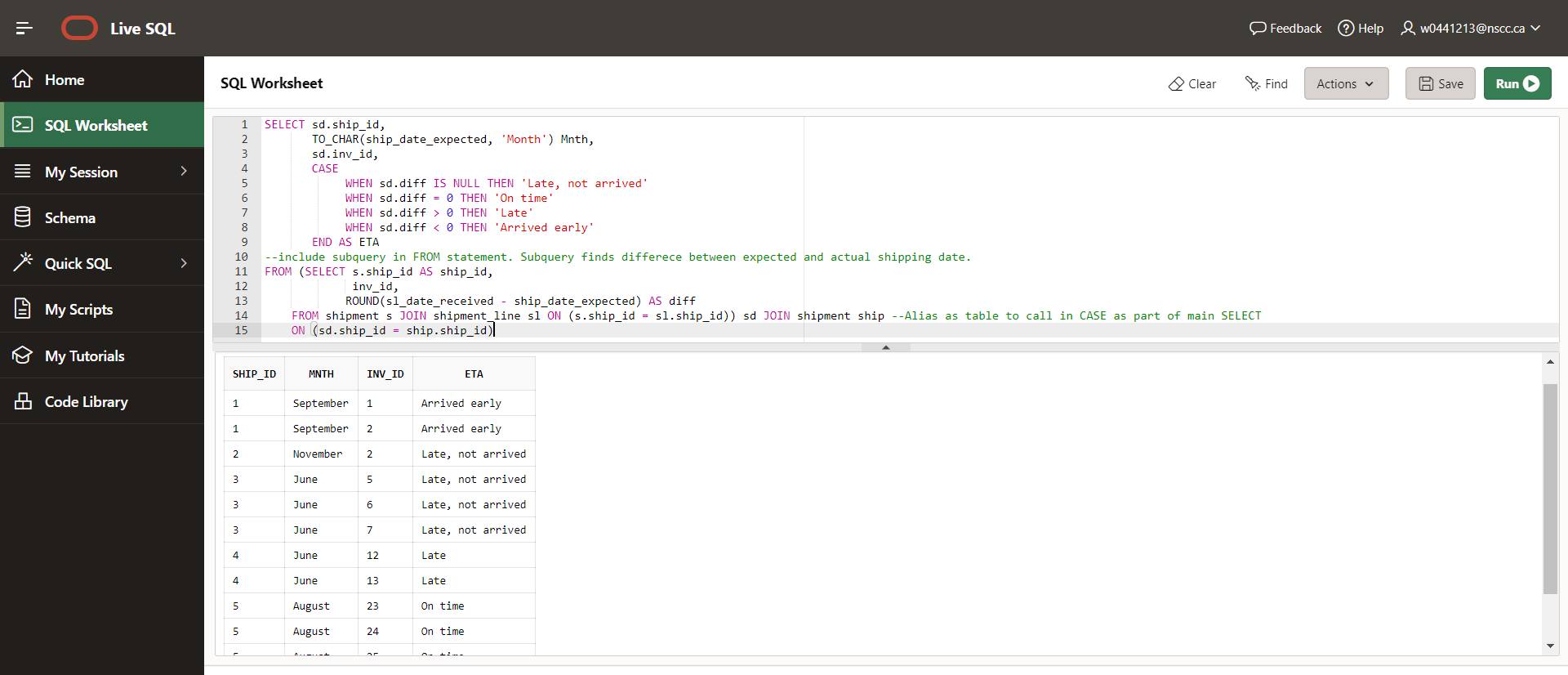
ON (sd.ship\_id = ship.ship\_id)

**EXPLANATION:**

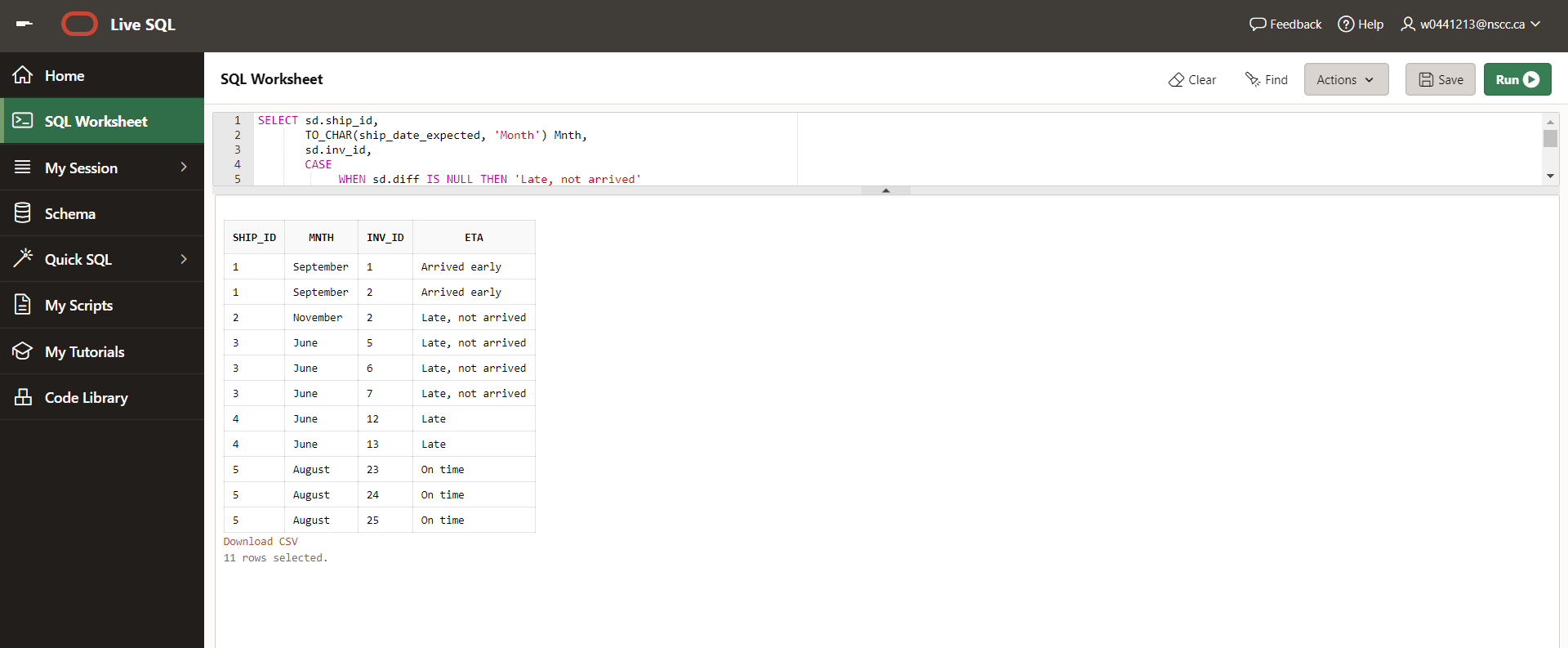
I put \_\_ in the SQL clause because the customer needs to see\_

**SCREENSHOT:**

SQL Code and part of output



Output



1. The guys in sales want to know revenue generated by customer, ‘cause they are all on commission. They THINK it might change by season, and that some customers buy more of certain colors, and some customers buy most stuff from some particular category.

**SQL STATEMENT:**

SELECT C\_ID AS CUSTOMER, OS\_ID AS SEASON, COLOR, CAT\_ID AS CATEGORY, (OL\_QUANTITY\*INV\_PRICE) AS REVENUE

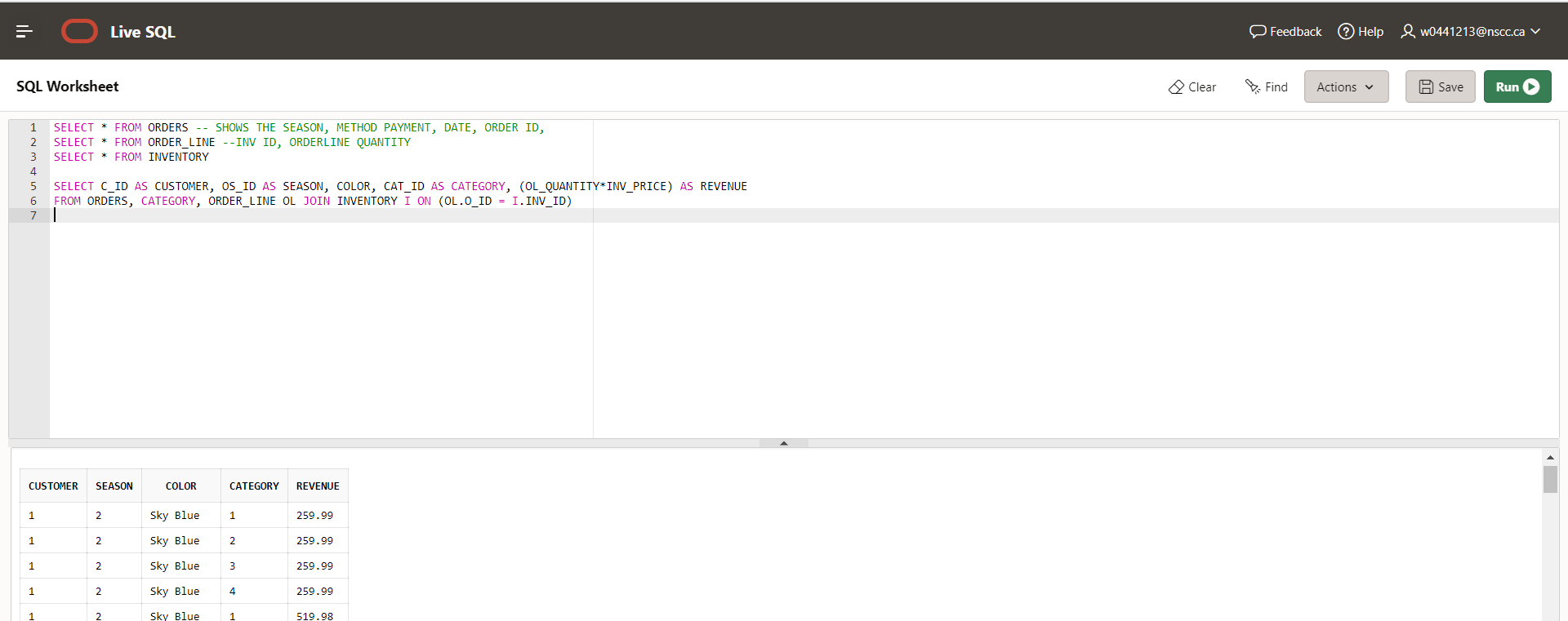
FROM ORDERS, CATEGORY, ORDER\_LINE OL JOIN INVENTORY I ON (OL.O\_ID = I.INV\_ID)

**EXPLANATION:**

By joining the order\_line and inventory table we can grab the information necessary to create the revenue. I included the customer, season, color, and category id as well so that the guys in sales would be able to check what can affect revenue. Once this is loaded into a visualization platform, it will be clear for the guys in sales to determine what can affect revenue the most.

**SCREENSHOT:**

SQL code and part of the output



Output





(4) The warehouse manager wants to optimize shelf space usage in the warehouse by planning ahead to anticipate monthly sales demand for each inv\_id. They would like to see, for each inv\_id, by month,

the name of the inv\_id

the name of the category

the size of the inventory

the month name

the historic monthly average quantity demanded by sales for the next month (don't forget to include the current month)

the historic monthly average quantity ordered by shipping in the current month

**SQL STATEMENT:**

-- to get the structure

SELECT \* FROM ORDERS

SELECT \* FROM ORDER\_LINE

SELECT \* FROM AVG\_SOLD

--average sold

CREATE TABLE AVG\_SOLD AS

SELECT TO\_CHAR(O\_DATE, 'MONTH') MON,

INV\_ID,

SUM(OL\_QUANTITY)AS "How\_Many"

FROM ORDER\_LINE OL JOIN ORDERS O ON (OL.O\_ID = O.O\_ID)

GROUP BY TO\_CHAR(O\_DATE, 'MONTH'), INV\_ID

ORDER BY TO\_CHAR(O\_DATE, 'MONTH'), INV\_ID

--joining with months

SELECT ORDER\_LINE.INV\_ID

HowMany

FROM ORDERS JOIN ORDER\_LINE ON (ORDERS.O\_ID = ORDER\_LINE.O\_ID)

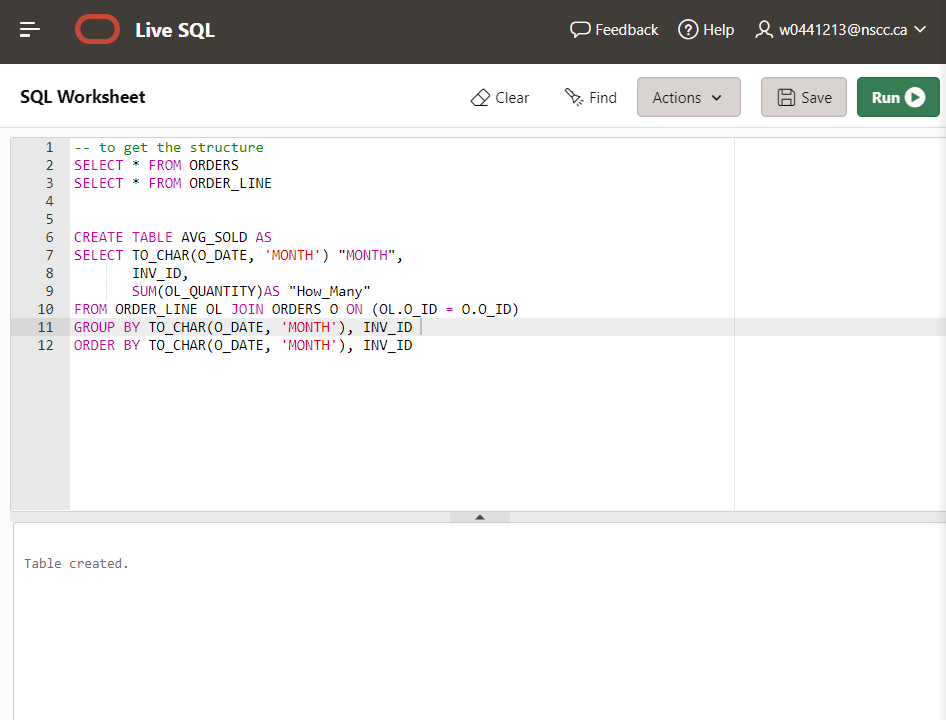
JOIN AVG\_SOLD ON ((ORDER\_LINE.INV\_ID = AVG\_SOLD.INV\_ID) AND

(TO\_CHAR(ADD\_MONTHS(ORDERS.O\_DATE, 1), 'Month') =

AVG\_SOLD.MON))

**SCREENSHOT:**

Creating the Avg\_Sold table



Creating the join with months with the output.



(5) What is the potential for creating a single "source of truth" that could satisfy all 4 of the above scenarios? Compare the four SELECT statements. Consider what is in each of their SELECT clauses. Consider the FROM clause table joins. Could you satisfy them all with one SELECT statement? Answer why and/or why not. Build the statement if you can! Justify your answer by counting result set rows. Super Hint: UNION, INTERSECT, MINUS, UNION ALL, etc

I think there is some potential in doing everything in one SELECT statement. However, I do not think it is best to do it all in one SELECT statement without having it look very messy and unorganized. I prefer to see code that is easy to read and understand, that is why I think that I would want everything in one SELECT statement. I do not think I would be able to easily create the code to satisfy all the scenarios in one, but it might be possible to have:

SELECT statement one minus SELECT statement two using a FROM clause. You have to subtract the outer join. If there is four SELECT statements, only the fourth one can have an ORDER BY clause. Column titles are going to be the column aliases of the very first query in the stack.