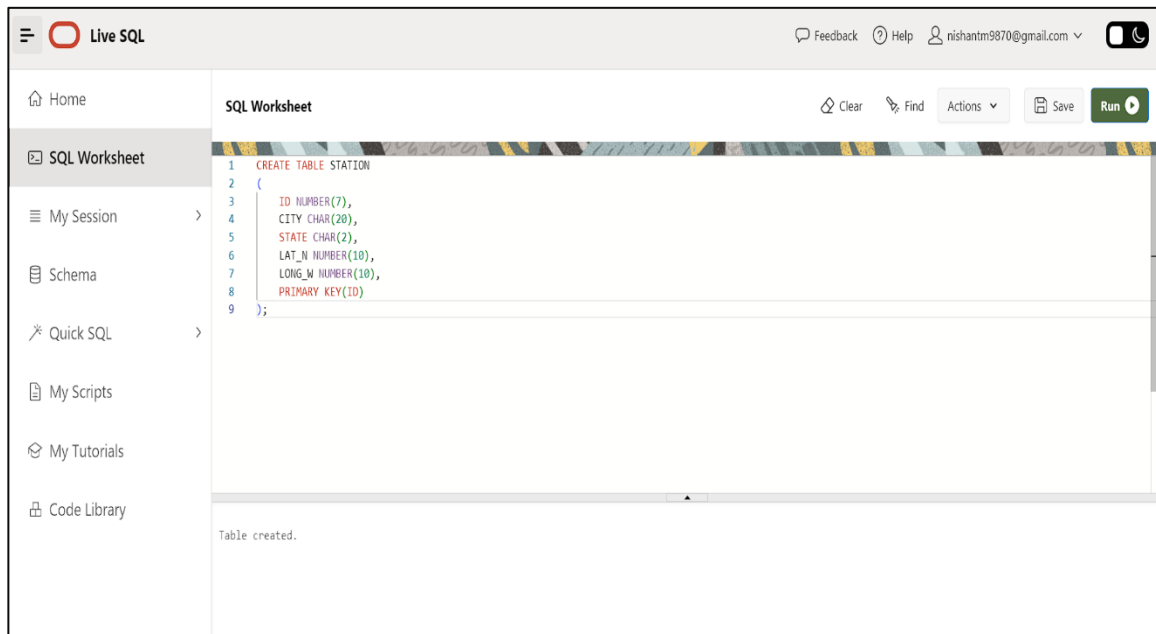


1. Create a table “Station” to store information about weather observation stations:

```
➤ CREATE TABLE STATION  
(  
    ID NUMBER(7),  
    CITY CHAR(20),  
    STATE CHAR(2),  
    LAT_N NUMBER(10),  
    LONG_W NUMBER(10),  
    PRIMARY KEY(ID)  
);
```

SCREENSHOT:

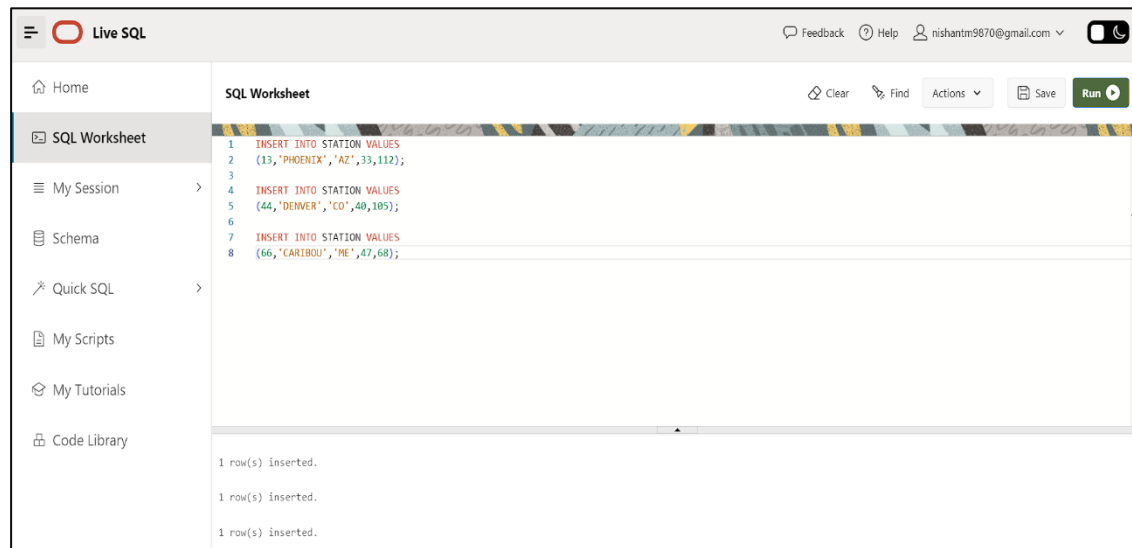


2. Insert the records into the table:

```
➤ INSERT INTO STATION VALUES  
    (13, 'PHOENIX', 'AZ', 33, 112);  
INSERT INTO STATION VALUES  
    (44, 'DENVER', 'CO', 40, 105);  
INSERT INTO STATION VALUES  
    (66, 'CARIBOU', 'ME', 47, 68);
```

ASSIGNMENT

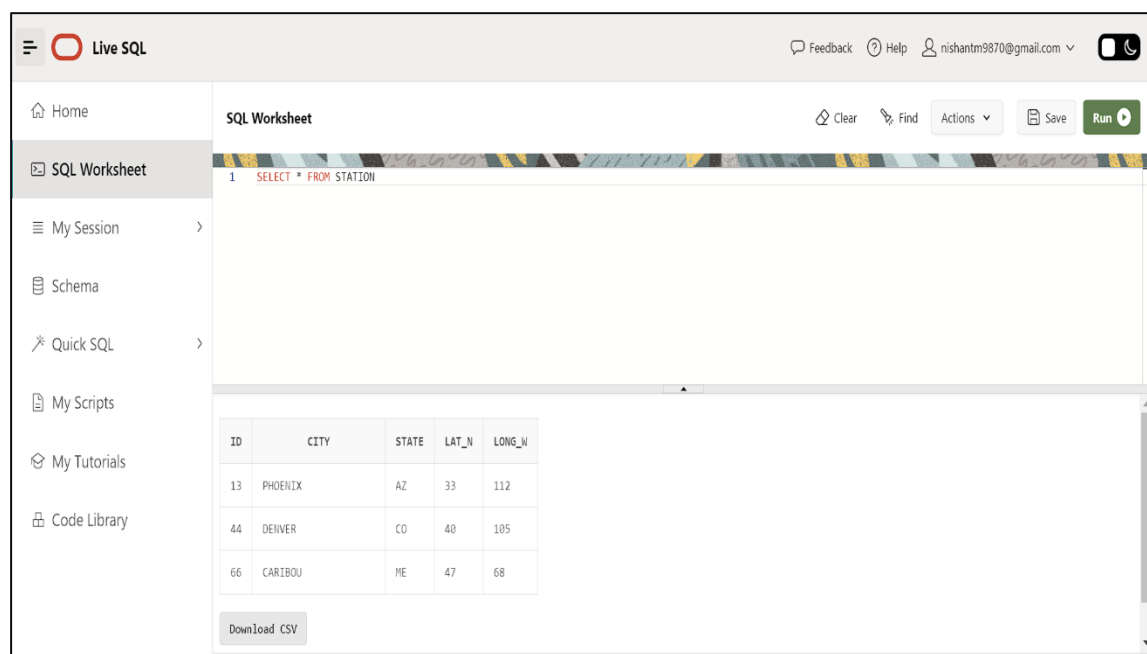
SCREENSHOT:



3. Execute a query to look at table STATION in undefined order:

➤ SELECT * FROM STATION;

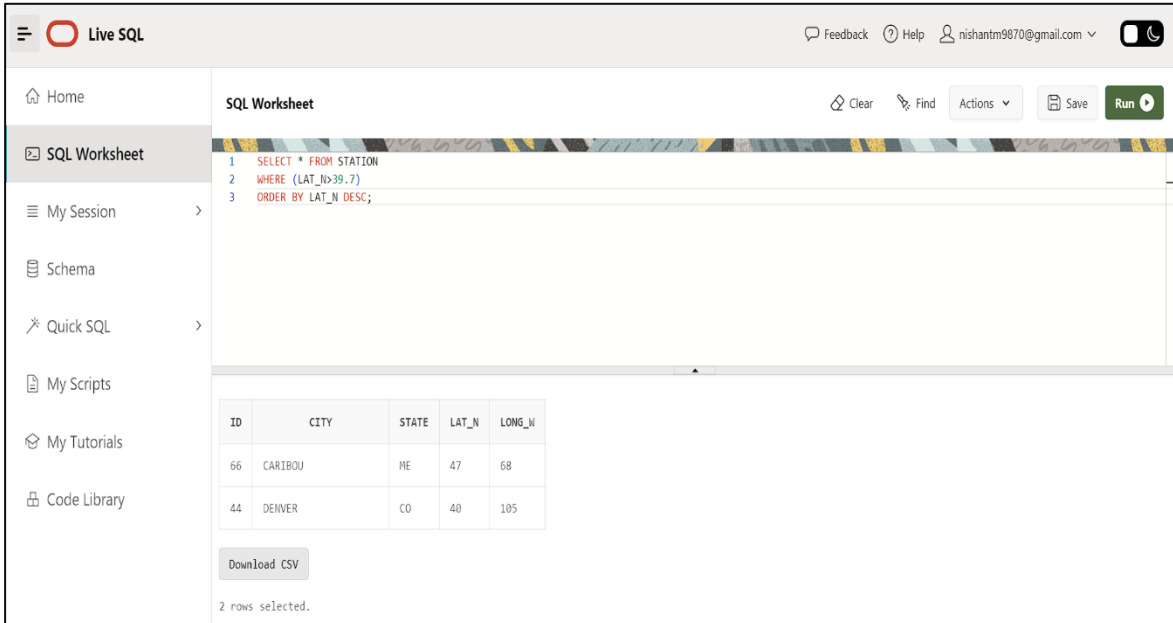
SCREENSHOT:



4. Execute a query to select Northern stations (Northern latitude > 39.7):

```
➤ SELECT * FROM STATION
  WHERE (LAT_N>39.7)
 ORDER BY LAT_N DESC;
```

SCREENSHOT:



The screenshot shows the 'Live SQL' web application interface. On the left is a sidebar with navigation links: Home, SQL Worksheet (selected), My Session, Schema, Quick SQL, My Scripts, My Tutorials, and Code Library. The main area is titled 'SQL Worksheet' and contains the following SQL query:

```
1 SELECT * FROM STATION
2 WHERE (LAT_N>39.7)
3 ORDER BY LAT_N DESC;
```

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

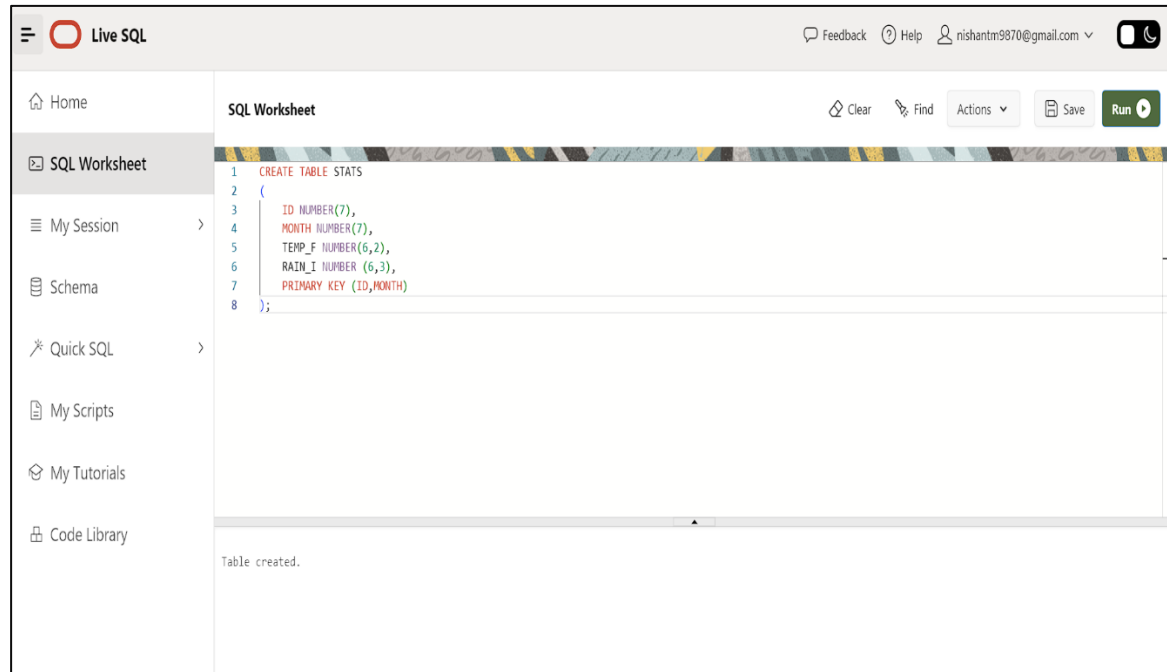
ID	CITY	STATE	LAT_N	LONG_W
66	CARIBOU	ME	47	68
44	DENVER	CO	40	105

At the bottom of the results, there is a 'Download CSV' button and a status message: '2 rows selected.'

5. Create another table, 'STATS', to store normalized temperature and precipitation data:

```
➤ CREATE TABLE STATS
(
  ID NUMBER(7),
  MONTH NUMBER(7),
  TEMP_F NUMBER(6,2),
  RAIN_I NUMBER (6,3),
  PRIMARY KEY (ID, MONTH)
);
```

SCREENSHOT:

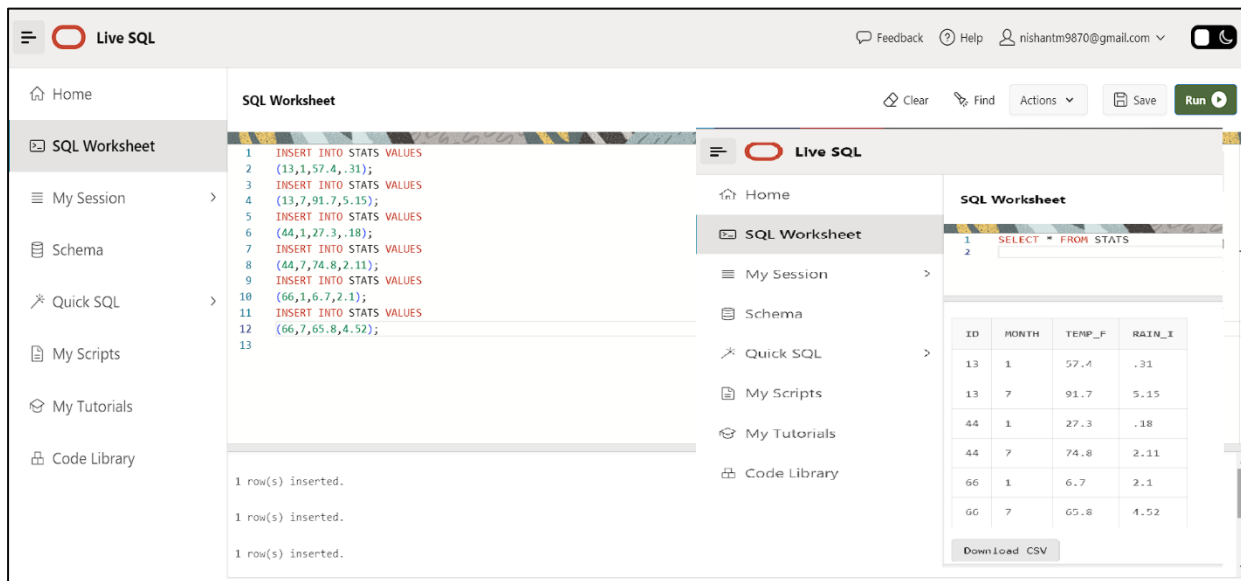


6. Populate the table STATS with some statistics for January and July:

- INSERT INTO STATS VALUES
(13,1,57.4,.31);
- INSERT INTO STATS VALUES
(13,7,91.7,5.15);
- INSERT INTO STATS VALUES
(44,1,27.3,.18);
- INSERT INTO STATS VALUES
(44,7,74.8,2.11);
- INSERT INTO STATS VALUES
(66,1,6.7,2.1);
- INSERT INTO STATS VALUES
(66,7,65.8,4.52);

ASSIGNMENT

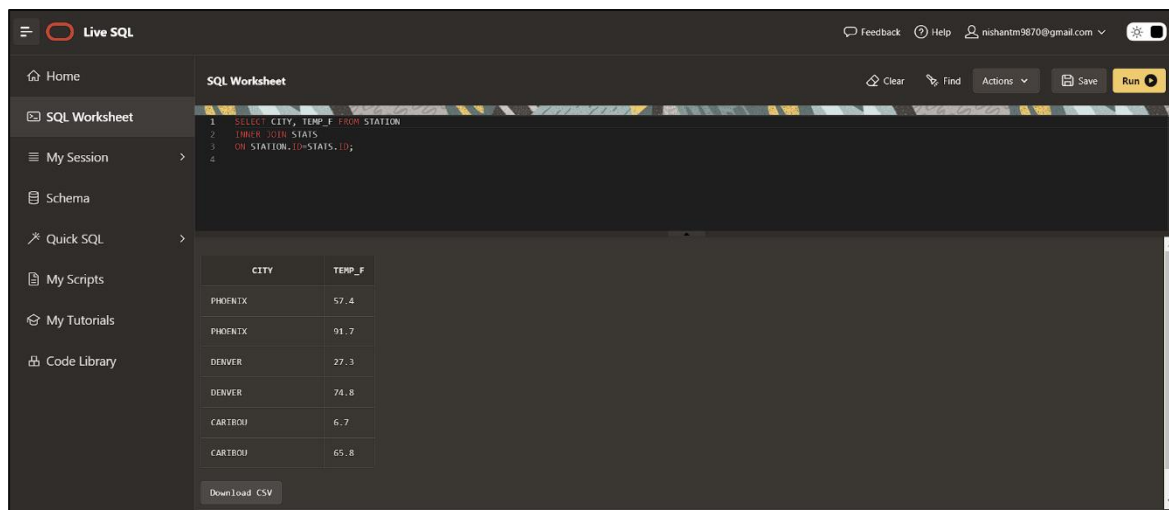
SCREENSHOT:



7. Execute a query to display temperature stats (from STATS table) for each city (from the Station table):

- SELECT CITY, TEMP_F FROM M STATION
INNER JOIN STATS
ON STATION.ID=STATS.ID;

SCREENSHOT:

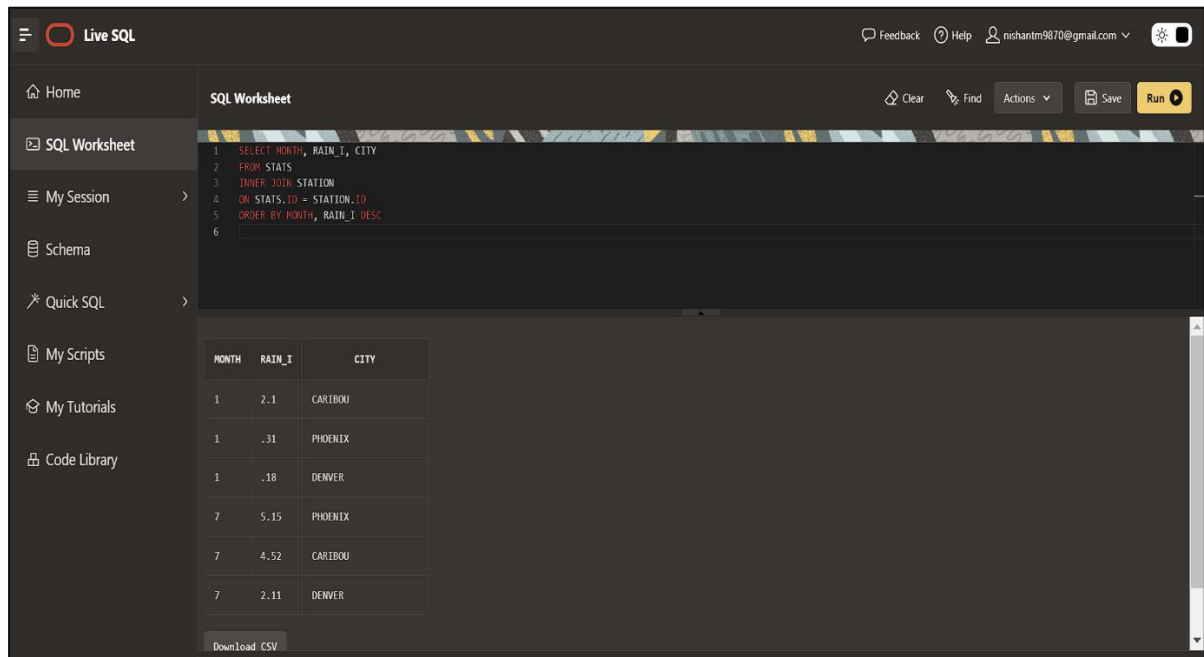


8. Execute a query to look at the table STATS, ordered by month and greatest rainfall, with columns rearranged. It should also show the corresponding cities:

ASSIGNMENT

- SELECT MONTH, RAIN_I, CITY
FROM STATS
INNER JOIN STATION
ON STATS.ID = STATION.ID
ORDER BY MONTH, RAIN_I DESC;

SCREENSHOT:



9. Execute a query to look at temperatures for July from table STATS, lowest temperatures first, picking up city name and latitude.

- SELECT CITY, LAT_N AS LATITUDE, TEMP_F AS TEMPERATURE, MONTH, FROM
STATION
INNER JOIN STATS
ON STATION.ID = STATS.ID
WHERE MONTH=7
ORDER BY TEMP_F;

SCREENSHOT:

ASSIGNMENT

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

```
1 SELECT CITY, LAT_N AS LATITUDE, TEMP_F AS TEMPERATURE, MONTH
2 FROM STATION
3 INNER JOIN STATS
4 ON STATION.ID=STATS.ID
5 WHERE MONTH =7
6 ORDER BY TEMP_F;
```

The results table is as follows:

CITY	LATITUDE	TEMPERATURE	MONTH
CARIBOU	47	65.8	7
DENVER	40	74.8	7
PHOENIX	33	91.7	7

Download CSV

2023 Oracle · Live SQL 23.1.1, running Oracle Database 19c EE Extreme Perf - 19.17.0.0.0 · Database Documentation · Ask Tom · Dev Gym
Built with ❤️ using Oracle APEX · Privacy · Terms of Use

10. Execute a query to show MAX and MIN temperatures as well as average rainfall for each city:

- SELECT CITY, MAX(TEMP_F) AS MAX_TEMP, MIN(TEMP_F) AS MIN_TEMP, AVG(RAIN_I) AS RAINFALL FROM STATION
INNER JOIN STATS
ON STATS.ID=STATS.ID
GROUP BY CITY;

SCREENSHOT:

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

```
1 SELECT CITY, MAX(TEMP_F) AS MAX_TEMP, MIN(TEMP_F) AS MIN_TEMP, AVG(RAIN_I) AS RAINFALL
2 FROM STATION
3 INNER JOIN STATS
4 ON STATION.ID=STATS.ID
5 GROUP BY CITY;
```

The results table is as follows:

CITY	MAX_TEMP	MIN_TEMP	RAINFALL
CARIBOU	65.8	6.7	3.31
DENVER	74.8	27.3	1.145
PHOENIX	91.7	57.4	2.73

11. Execute a query to display each city's monthly Celcius temperature and Centimeter rainfall:

- CREATE VIEW STATS_B (ID, MONTH, TEMP_C, RAIN_CM) AS
SELECT ID, MONTH, (TEMP_F-32)*5/9, RAIN_I*2.54
FROM STATS;

SCREENSHOT:

The screenshot shows the Live SQL interface with a SQL Worksheet. The query executed is:

```
1 CREATE VIEW STATS_B (ID, MONTH, TEMP_C, RAIN_CM) AS
2 SELECT ID, MONTH, (TEMP_F-32)*5/9, RAIN_I*2.54
3 FROM STATS;
```

The results are displayed in a table with the following data:

ID	MONTH	TEMP_C	RAIN_CM
13	1	14.833333333333333	.7874
13	7	33.800000000000004	13.4801
44	1	-2.641111111111111	.4572
44	7	23.777777777777777	5.3504
66	1	14.450000000000001	3.644
66	7	18.11111111111111	11.3808

12. Update all rows of table STATS to compensate for faulty rain gauges known to read 0.01 inches low.

- UPDATE STATS SET RAIN_I = RAIN_I + 0.01;

SCREENSHOT:

The screenshot shows the Live SQL interface with a SQL Worksheet. The query executed is:

```
1 UPDATE STATS SET RAIN_I = RAIN_I + 0.01;
```

The results show "6 row(s) updated." and the updated table data:

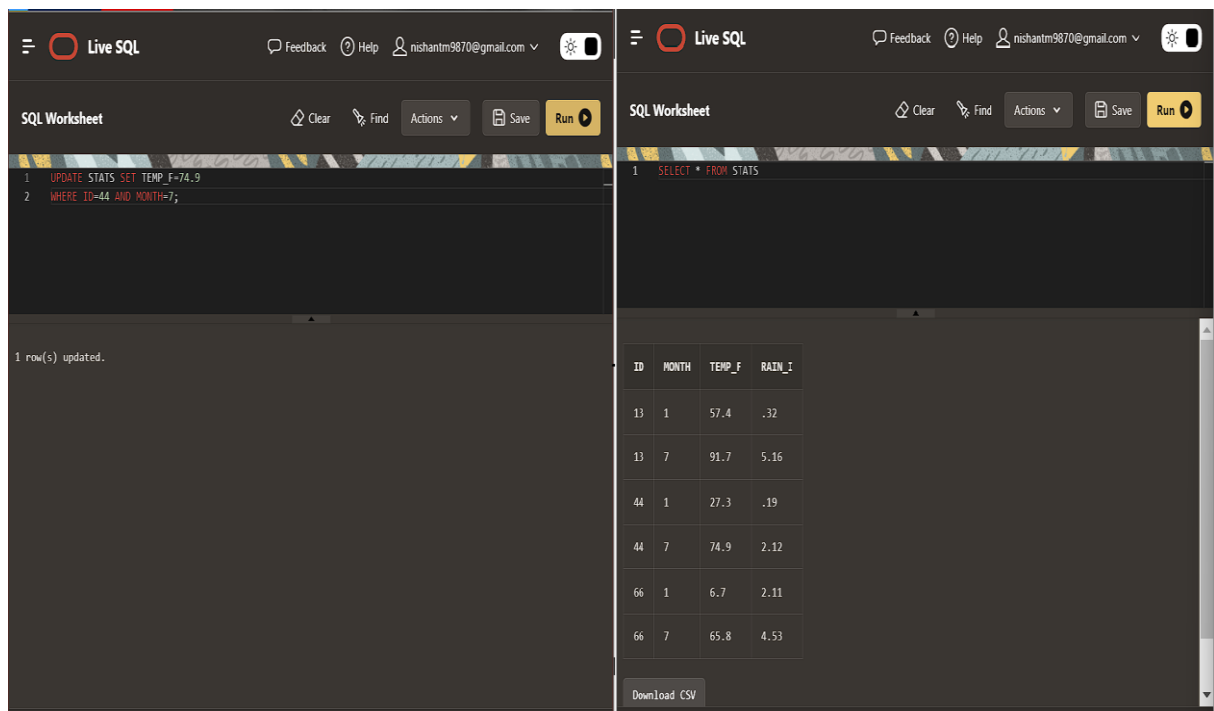
ID	MONTH	TEMP_F	RAIN_I
13	1	57.4	.32
13	7	91.7	5.15
44	1	27.9	.19
44	7	74.8	2.12
66	1	6.7	2.11
66	7	65.8	4.53

13. Update Denver's July temperature reading as 74.9:

➤ UPDATE STATS SET TEMP_F=74.9

WHERE ID=44 AND MONTH=7;

SCREENSHOT:



The left screenshot shows the SQL Worksheet with the following code:

```
1 UPDATE STATS SET TEMP_F=74.9
2 WHERE ID=44 AND MONTH=7;
```

The result shows: 1 row(s) updated.

The right screenshot shows the SQL Worksheet with the following code:

```
1 SELECT * FROM STATS
```

The result is a table with the following data:

ID	MONTH	TEMP_F	RAIN_I
13	1	57.4	.32
13	7	91.7	5.16
44	1	27.3	.19
44	7	74.9	2.12
66	1	6.7	2.11
66	7	65.8	4.53

Download CSV