**Lab 11**

**Create a new workspace and use the productdb script provided in eLearning. For each problem, enter your query and attach a screen shot of the result set.**

1. **When was the last time that the statistics were updated on the LINE table? Show the query and write the date. (2 pts)**

**Query:**

**analyze table line**

**compute statistics;**

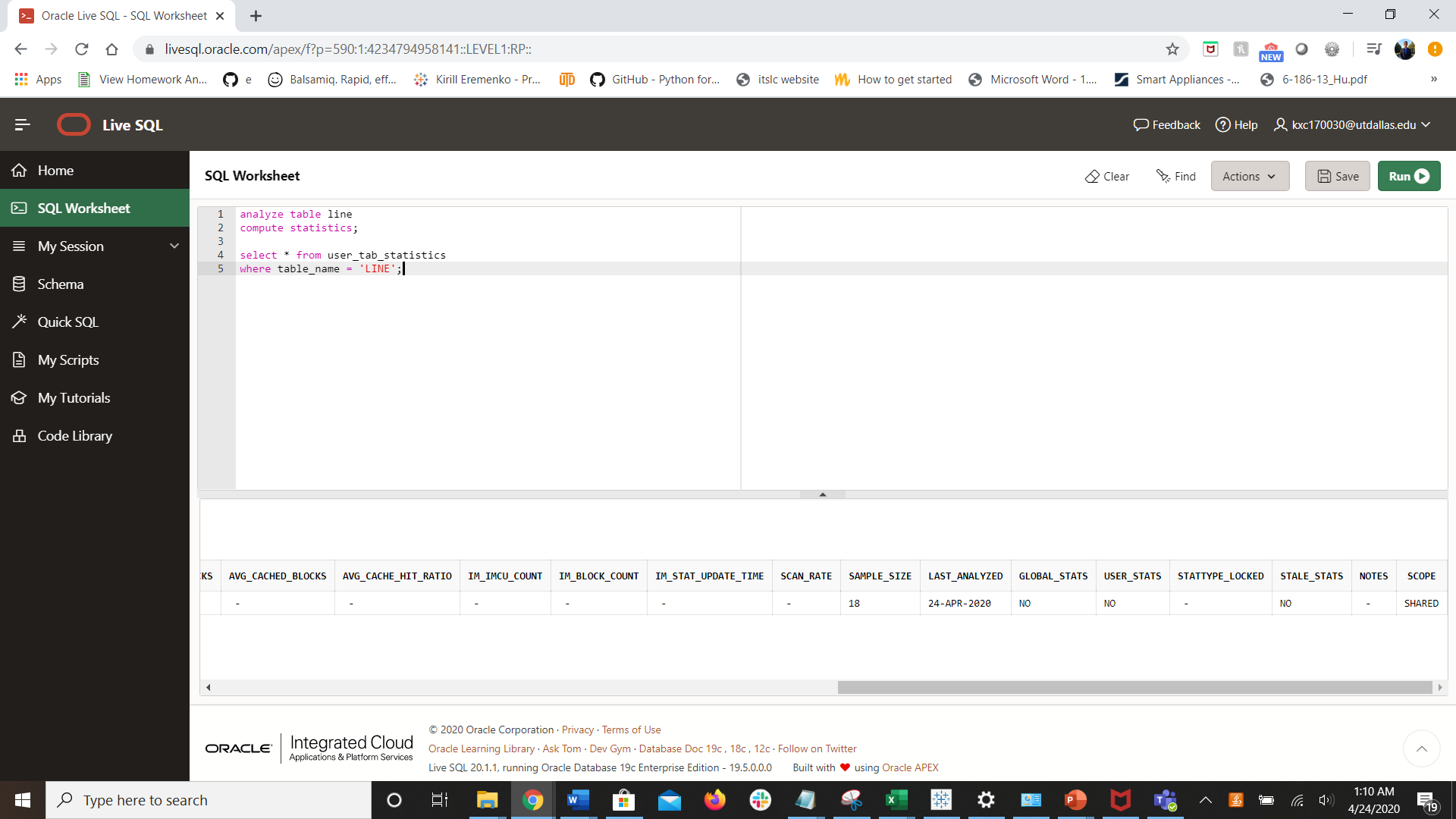
**select \* from user\_tab\_statistics**

**where table\_name = 'LINE';**

**Date: 24-APR-2020**

A screenshot of a computer screen

Description automatically generated



1. **Are there any indexes on the PRODUCT, VENDOR, or LINE tables? Show the query you would use to answer the question and then list any indexes. (2 pts)**

**select \***

**from user\_indexes**

**where table\_name in ('LINE', 'PRODUCT', 'VENDOR');**

A screenshot of a computer screen

Description automatically generated

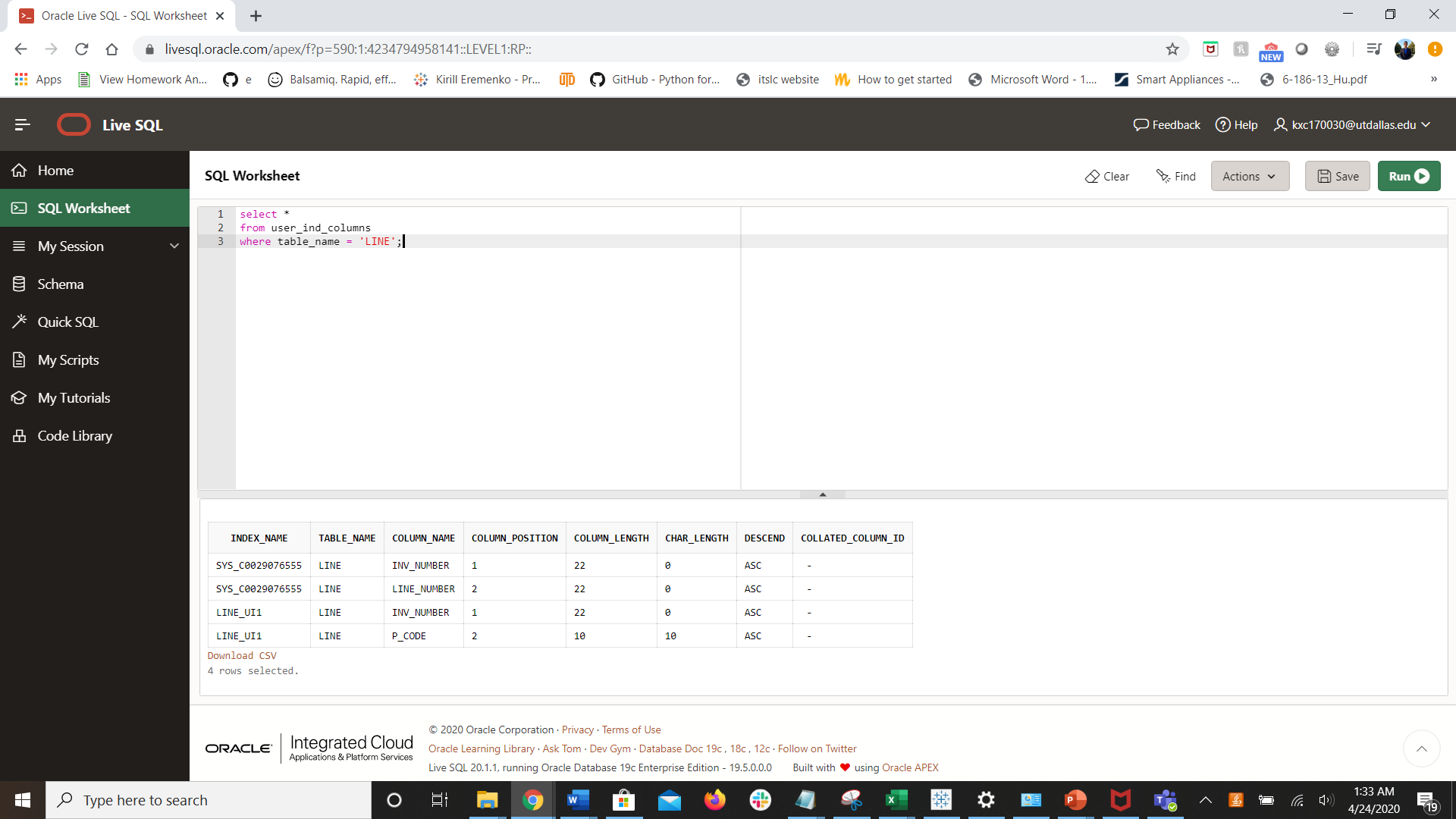
1. **If you want to see the columns that are indexed in an index, which table would you query ? Query that table and look at the line table to see which columns are indexed. (2 pts)**

**user\_ind\_columns**

**select \***

**from user\_ind\_columns**

**where table\_name = 'LINE';**



1. **Execute the following query. Run the explain plan for the query and view the results. Add the index on line\_units column, run the explain plan and view the results. Make sure that you update the statistics after creating the index. Can you improve performance by adding an index on the line\_units column? (3 pts)**

**Query:**

select p\_code, sum(line\_units)

from line

group by p\_code

having sum(line\_units) >

(select(max(line\_units)) from line)

**explain plan for**

**select p\_code, sum(line\_units)**

**from line**

**group by p\_code**

**having sum(line\_units) > (select(max(line\_units)) from line);**

**SELECT \* FROM table(dbms\_xplan.display);**

**A screenshot of a computer screen

Description automatically generated**

**A screenshot of a computer screen

Description automatically generated**

**create index line\_units\_index**

**on line (line\_units)**

**A screenshot of a computer screen

Description automatically generated**

**explain plan for**

**select p\_code, sum(line\_units)**

**from line**

**group by p\_code**

**having sum(line\_units) > (select(max(line\_units)) from line);**

**SELECT \* FROM table(dbms\_xplan.display);**

**A screenshot of a computer screen

Description automatically generatedA screenshot of a computer screen

Description automatically generated**

**The performance is improved by adding an index on line\_items as the number of rows scanned, number of bytes used, and the cost (%CPU) is reduced.**

**5. Consider the query below. Can you rewrite the query and/or add indexes to improve the performance of the query? Write the new query and list any indexes you created. (There are multiple things you can do to improve the query). Explain the performance improvement gained after making these changes and show the explain plan. (3 pts)**

select p\_code, p\_descript, p\_price, p.v\_code, v\_state

from product p, vendor v

where p.v\_code = v.v\_code

and v\_state = 'FL'

and v\_areacode = '904'

order by p\_price

**explain plan for**

**select p\_code, p\_descript, p\_price, p.v\_code, v\_state**

**from product p, vendor v**

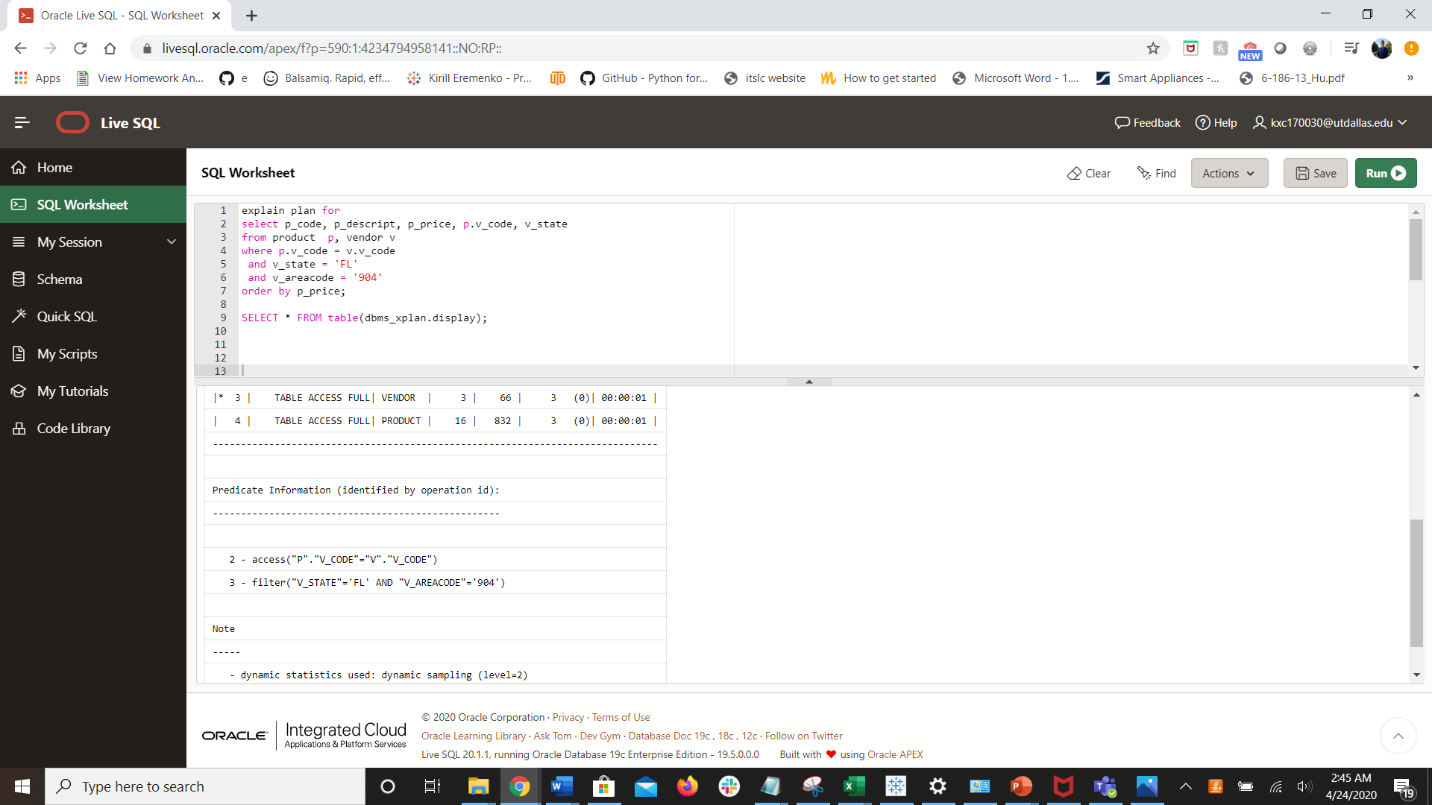
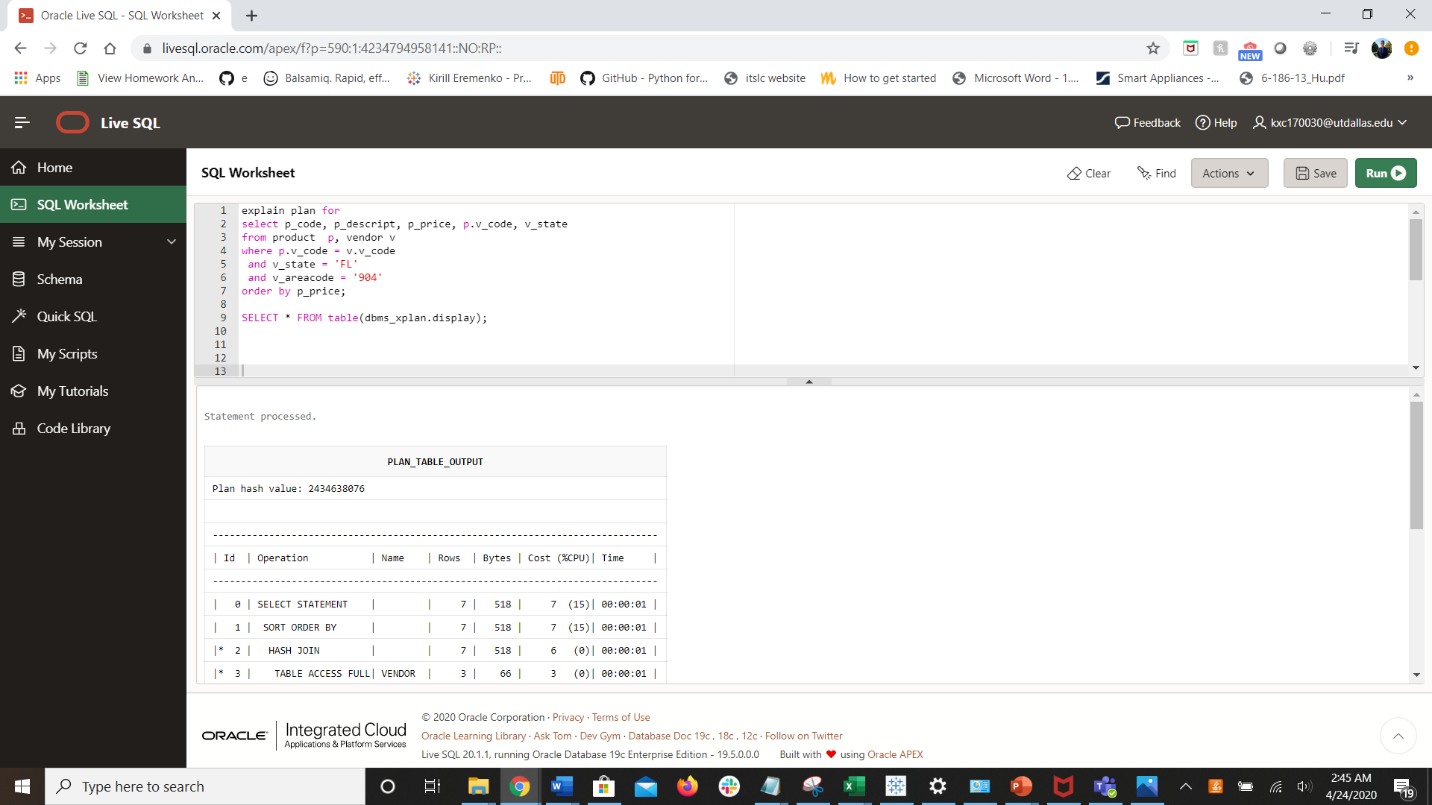
**where p.v\_code = v.v\_code**

**and v\_state = 'FL'**

**and v\_areacode = '904'**

**order by p\_price;**

**SELECT \* FROM table(dbms\_xplan.display);**

****

**create index vendor\_new\_index**

**on vendor (v\_code, v\_state, v\_areacode);**

**/\*multiple column index – you can also treat this as 3 indexes\*/**

**explain plan for**

**select p\_code, p\_descript, p\_price, p.v\_code, v\_state**

**from product p, vendor v**

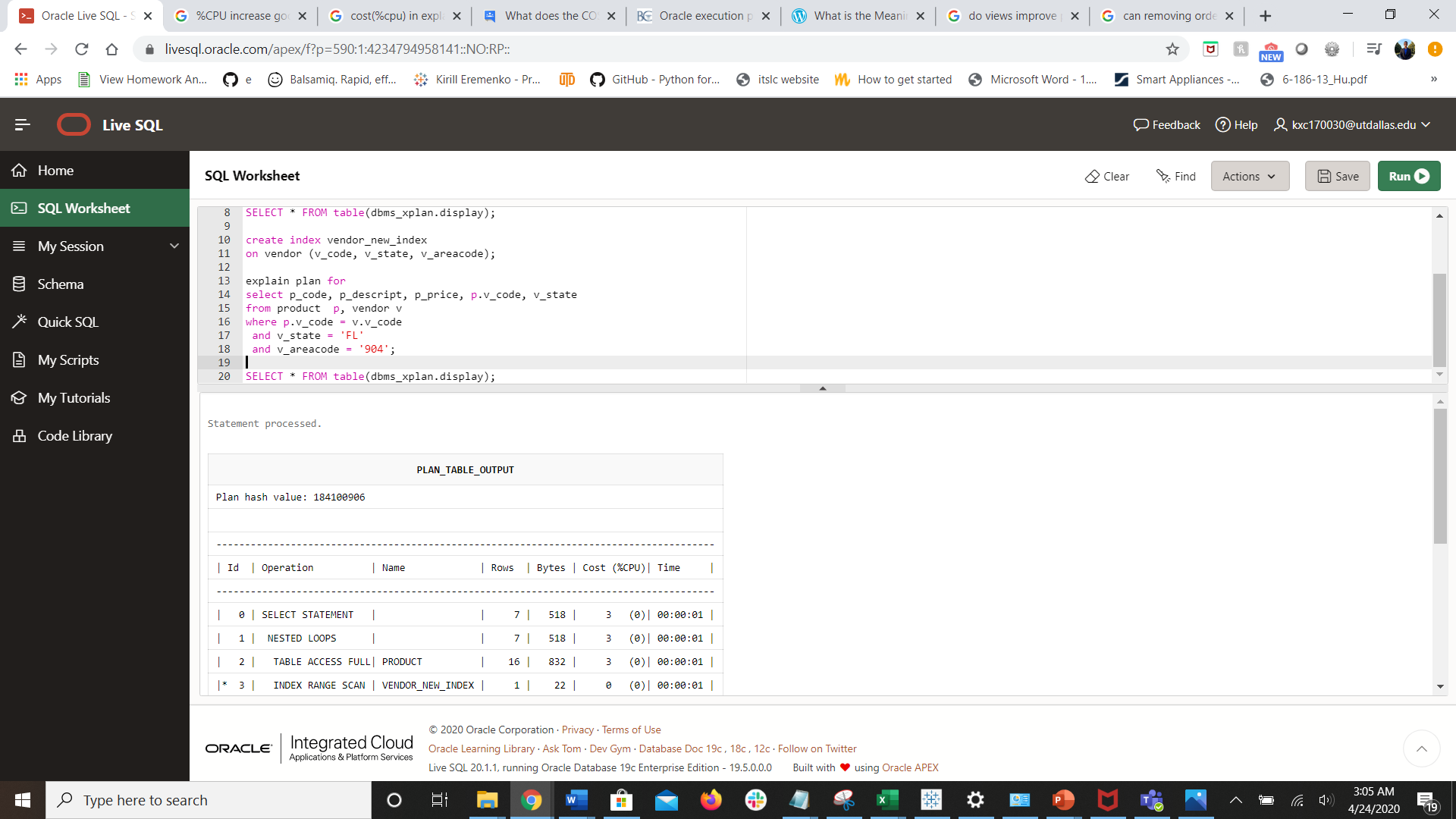
**where p.v\_code = v.v\_code**

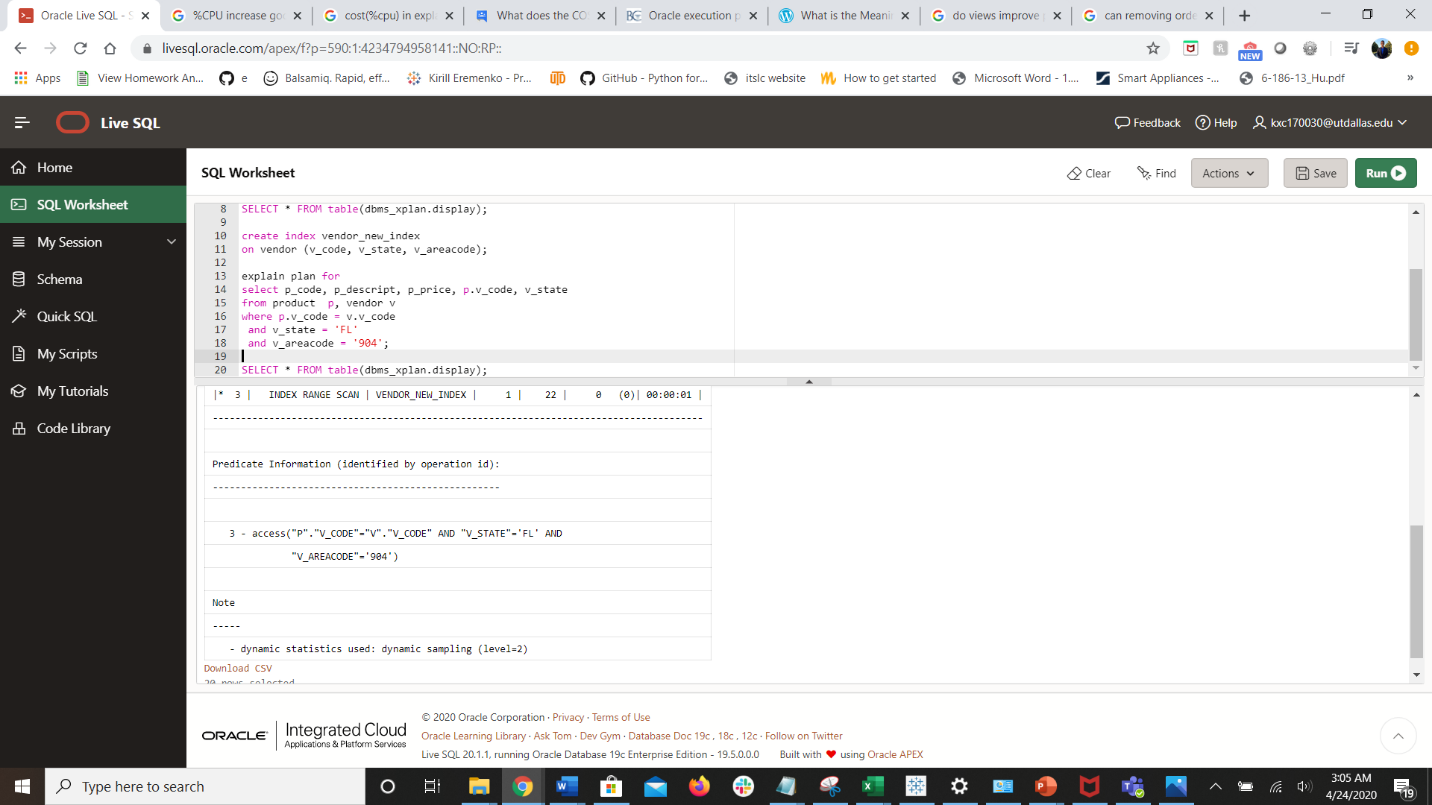
**and v\_state = 'FL'**

**and v\_areacode = '904';**

**/\*removed the order by clause\*/**

**SELECT \* FROM table(dbms\_xplan.display);**

****

****

**Performance improvement:**

* **Cost (%CPU) decreases significantly for SELECT STATEMENT**
* **TABLE ACCESS FULL on the VENDOR table is replaced with INDEX RANGE SCAN**
  + **Rows, bytes, and cost (%CPU) decrease significantly**
* **HASH JOIN is replaced with NESTED LOOPS**
  + **Cost (%CPU) decreases**
* **SORT ORDER BY is removed so there are less total rows scanned, bytes used, and cost (%CPU)**