Daily Health Check Database

(COVID Symptom Check)

Umakanth Sai Balguri

Nynali Gopu

Sri Venkata Venu Gopal Guddati

Aishwaryaa Mekala

Akshitha Reddy Mallipeddi

Dheeraj Sai Chava

Table of Contents

1. Executive Summary

Due to the recent emergence of COVID pandemic, there has been a need to immediate changes in the way organisations operate. Since it's a contagious disease, contact tracking has become an important aspect of its monitor and control. Unlike workplace, Universities are areas which are accessed by thousands of individuals each day without any particular pattern. Hence, It has become extremely important from educational institutions point of view to control the flow of individuals to campus locations through monitoring systems. The allowance should depend on their health status and the criticality factor. Overall, deriving these statistics and actively monitoring them becomes an extremely important task to control the spread of the disease and maintain wellbeing of every individual.

The proposed database design is a copy of "**USF Archivum Return to Campus**" with minor tweaks to ensure all the data required for analysis is gathered. It intends to collect campus visit details, checks for various COVID symptoms and finally issues a pass if there is no risk associated with the individual's health condition. The primary focus has been on creating and implementing the full normalised database design including all the constrains to ensure there is no data redundancy, protection against inconsistent data (submission of blank values). The central table, Visit has been loaded with 15000 records to have diverse classification, this flows to all the other linked tables which use the same Primary Key. An SQL file has been created for DDL statements and a CSV file for data (attached to appendix section) for the purposes of easy migration/re-use in other connections.

For performance tuning experiments we have created a visit test table and demonstrated how using indexing, selectivity indexing, point queries and execution plan effectively reduces the cost and time. We also experimented with parallel execution and partitioning of visit table by month showing decrease in response times.

As Data visualization tools such as Tableau, Microsoft Power BI help us visualize the data available in different graphical representations which can be used to answer key questions, we created visualizations for key data insights such as daily pass issues, number of symptomatic individuals reported each day, daily numbers of individuals reported each symptom.

1.1. Area Wise Weightage

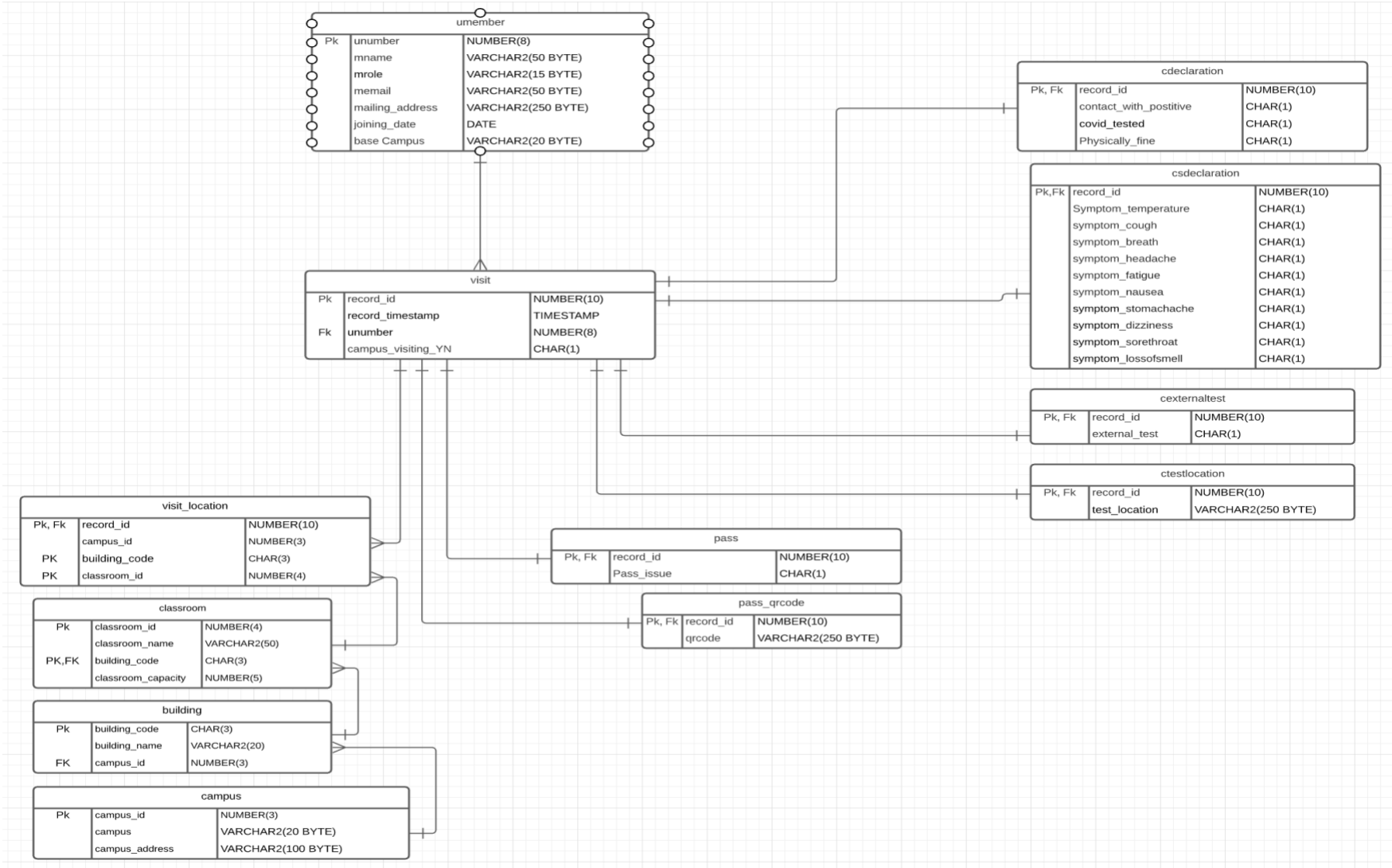
Topic Area	Description	Group-2 Points
Database Design	This part should include a logical database design (for the relational model), using normalization to control redundancy and integrity constraints for data quality.	**30**
Query Writing	This part is another chance to write SQL queries, explore transactions, and even do some database programming for stored procedures.	**30**
Performance Tuning	In this section, you can capitalize and extend your prior experiments with indexing, optimizer modes, partitioning, parallel execution and any other techniques you want to further explore.	**20**
Other Topics	Here you are free to explore any other topics of interest. Suggestions include DBA scripts, database security, interface design, data visualization, data mining, and NoSQL databases.	**20**

1.2. Team

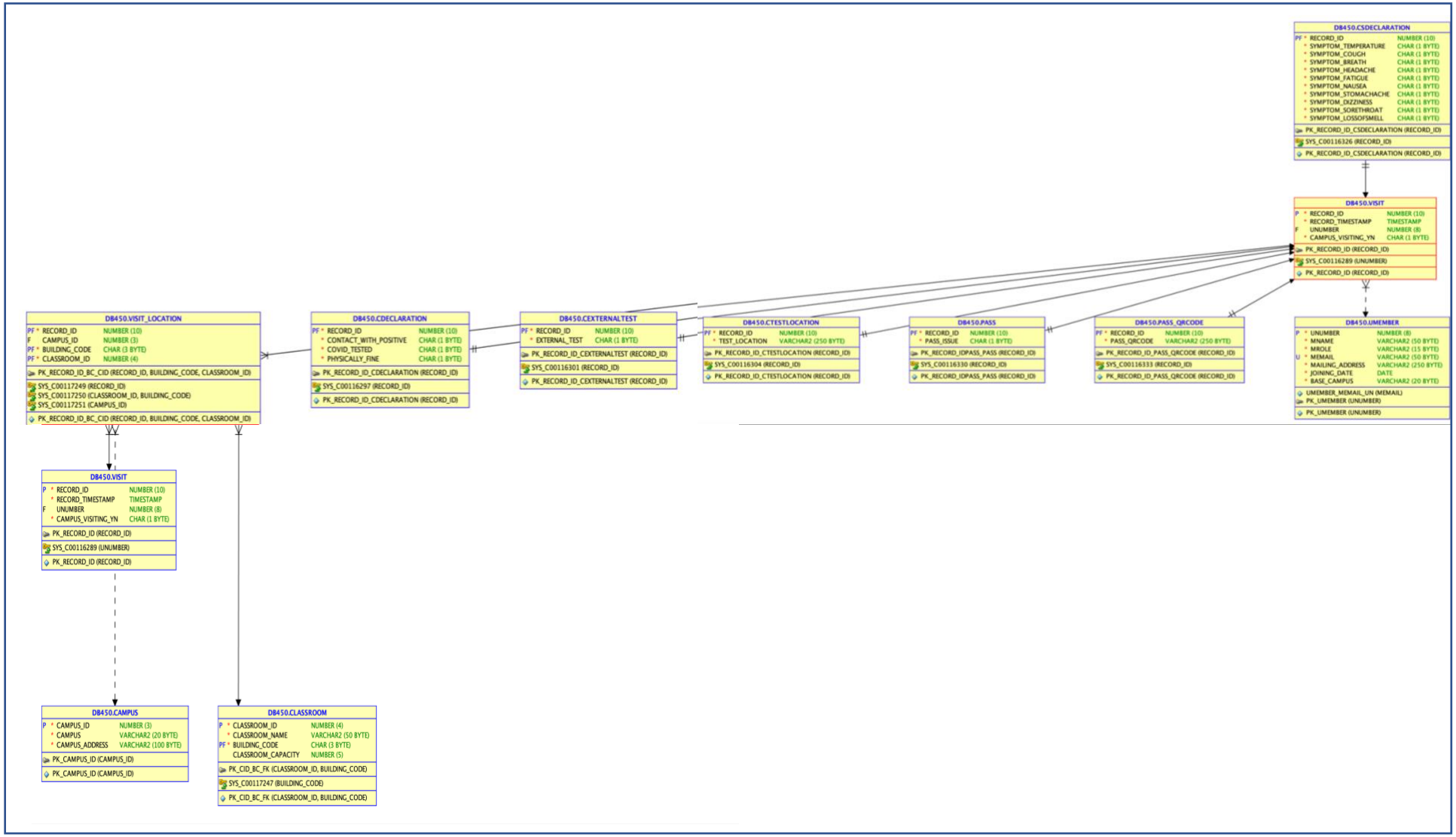
- Umakanth Sai Balguri
- Nynali Gopu
- Sri Venkata Venu Gopal Guddati
- Aishwaryaa Mekala
- Akshitha Reddy Mallipeddi
- Dheeraj Sai Chava

2. Database Design

2.1. Proposed database design



2.2. Implemented design in Oracle SQL



2.3. Data Dictionary

Table	Field Name	Data Type	Field Lenth/Format	Constraint	Description	Example
Umember	unumber	NUMBER	8	PRIMARY KEY	Unique ID for every individual	10000001
	mname	VARCHAR2	50	NOT NULL	Name of the individual	Alex
	mrole	VARCHAR2	15	NOT NULL	Role of the Individual	Student
	memail	VARCHAR2	50	NOT NULL	Email of the Individual	alex@usf.edu
	mailing_address	VARCHAR2	250	NOT NULL, UNIQUE	Mailing address of the Individual	35, Community Centre, New Friends Colony
	joining_date	Date	DD-MMM-YY	NOT NULL	Joining date of the Individual	12-Jan-20
	base_campus	VARCHAR2	20	NOT NULL, IN ('Tampa','Sarasota','St. Petersburg')	Base campus of the Individual	Sarasota
visit	Record_ID	NUMBER	10	PRIMARY KEY	Entry ID for the submission of declaration from Individual	202000018
	Record_Timestamp	TIMESTAMP	YYYY/MM/DD HH:MM:SS.XXXXX	NOT NULL	Timestamp of the entry submission	2020/2/7 21:55:58
	Unumber	NUMBER	8	FOREIGN KEY	Unique ID for every individual	10000018
	Campus_Visiting_YN	CHAR	1	NOT NULL, IN ('Y', 'N')	Individual visiting campus on that day ? Y/N	Y
Campus	campus_id	NUMBER	3	PRIMARY KEY	Id of the campus	100
	Campus	VARCHAR2	20	NOT NULL	Name of the campus	Tampa
	campus_address	VARCHAR2	100	NOT NULL	Address of the campus	Hope Lodge - American Cancer Society1004

building	building_code	CHAR	3	PRIMARY KEY	Building code- of the building	ACS
	building_name	VARCHAR2	50	NOT NULL	Name of the building	Hope Lodge - American Cancer Society1004
	campus_id	NUMBER	3	NOT NULL	Id of the campus	100
classroom	classroom_id	NUMBER	4	PRIMARY KEY	Id of the classroom	1004
	classroom_name	VARCHAR2	50	NOT NULL	Name of the classroom building	Hope Lodge - American Cancer Society1004
	building_code	CHAR	3	PRIMARY KEY, FOREIGN KEY	Building code- of the building	ACS
	classroom_capacity	NUMBER	5	NOT NULL	Capacity of the classroom/location	100
visit_location	record_id	NUMBER	10	PRIMARY KEY, FOREIGN KEY	Entry ID for the submission of declaration from Individual	202000018
	campus_id	NUMBER	3	NOT NULL, FOREIGN KEY	Id of the campus being visited by the Individual	100
	building_code	CHAR	3	PRIMARY KEY, FOREIGN KEY	Building code- of the building being visited by the individual	ACS
	classroom_id	NUMBER	4	PRIMARY KEY, FOREIGN KEY	Id of the classroom being visited by the Individual	1004
cdecleration	Record_ID	NUMBER	10	PRIMARY KEY, FOREIGN KEY	Entry ID for the submission of declaration from Individual	202000018

	Contact_With_Positive	CHAR	1	NOT NULL, IN ('Y', 'N')	Contact with any COVID positive in past 24Hrs? Y/N	Y
	COVID_Tested	CHAR	1	NOT NULL, IN ('Y', 'N')	COVID tested in past 24Hrs? Y/N	Y
	Physically_Fine	CHAR	1	NOT NULL, IN ('Y', 'N')	Feeling physically fine from past 24Hrs? Y/N	Y
cexternaltest	Record_ID	NUMBER	10	PRIMARY KEY, FOREIGN KEY	Entry ID for the submission of declaration from Individual	202000018
	External_Test	CHAR	1	NOT NULL, IN ('Y', 'N')	If the candidate was externally tested in past 24Hrs? Y/N	Y
ctestlocation	Record_ID	NUMBER	10	PRIMARY KEY, FOREIGN KEY	Entry ID for the submission of declaration from Individual	202000018
	Test_Location	VARCHAR2	250	NOT NULL	External Test location	13681 DOCTORS WAY
csdeclaration	Record_ID	NUMBER	10	PRIMARY KEY, FOREIGN KEY	Entry ID for the submission of declaration from Individual	202000018
	symptom_temperature	CHAR	1	NOT NULL, IN ('Y', 'N')	Check for symptom_temperature in last 24 hrs? Y/N	Y
	symptom_cough	CHAR	1	NOT NULL, IN ('Y', 'N')	Check for symptom_cough in last 24 hrs? Y/N	Y
	symptom_breath	CHAR	1	NOT NULL, IN ('Y', 'N')	Check for symptom_breath in last 24 hrs? Y/N	Y

	symptom_headache	CHAR	1	NOT NULL, IN ('Y', 'N')	Check for symptom_headache in last 24 hrs? Y/N	Y
	symptom_fatigue	CHAR	1	NOT NULL, IN ('Y', 'N')	Check for symptom_fatigue in last 24 hrs? Y/N	Y
	symptom_nausea	CHAR	1	NOT NULL, IN ('Y', 'N')	Check for symptom_nausea in last 24 hrs? Y/N	Y
	symptom_stomachache	CHAR	1	NOT NULL, IN ('Y', 'N')	Check for symptom_stomachache in last 24 hrs? Y/N	Y
	symptom_dizziness	CHAR	1	NOT NULL, IN ('Y', 'N')	Check for symptom_dizziness in last 24 hrs? Y/N	Y
	symptom_sorethroat	CHAR	1	NOT NULL, IN ('Y', 'N')	Check for symptom_sorethroat in last 24 hrs? Y/N	Y
	symptom_lossofsmell	CHAR	1	NOT NULL, IN ('Y', 'N')	Check for symptom_lossofsmell in last 24 hrs? Y/N	Y
pass	Record_ID	NUMBER	10	PRIMARY KEY, FOREIGN KEY	Entry ID for the submission of declaration from Individual	202000018
	Pass Issue	CHAR	1	NOT NULL, IN ('Y', 'N')	Record of whether pass was issued for the record Y/N?	Y
pass_qrcode	Record_ID	NUMBER	10	PRIMARY KEY, FOREIGN KEY	Entry ID for the submission of declaration from Individual	202000018
	Pass_qrcode	VARCHAR2	250	NOT NULL, UNIQUE	Qrcode string for the pass issue	Passid_test032

2.4. Design Assumptions

The database design was created basing USF Archivum Return to Campus application questionnaire, currently being used to collect COVID symptom check. Additionally, attributes have been added to achieve data integrity and COVID Data Analytics. Below are the design assumptions:

- Every Student or Faculty member is identified by a unique number (U-number)
- USF holds 3 campuses and every member is tagged to one of those as a base campus
- Pass will only be issued if the individual is visiting campus and has no Covid related symptoms.
- Pass will not be issued if the individual is not visiting the campus or has any of the Covid related symptom.
- An individual can visit multiple locations on a given day and he will be issued passes to all the selected locations based on his selections.
- All the visit locations will be tagged to one web form submission (record_id), which individual is expected to fill only once per day.
- A check constraint hasn't been used on external test location to provide flexibility to input test locations across the world. (In case of incoming International students)
- Pass Issue Y/N is being separately tracked to make the design future proof. (If pass was meant to be issued for mild COVID symptoms)
- Pass Issued will only be valid for the day for a particular location(s) submitted in the declaration.
- Pass QR code will be a unumber based uniquely generated string.

2.5. Data Integrity

All the constrains have been documented in Data Dictionary (Section 2.3 of this Document).

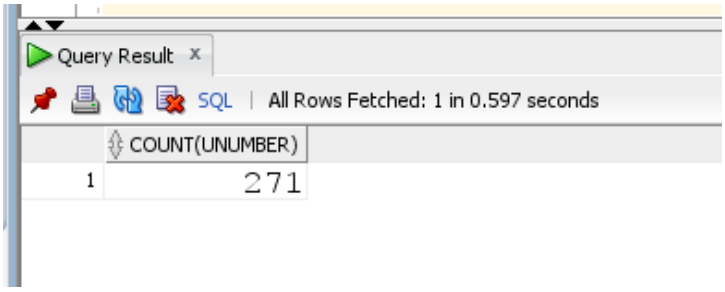
- The reason for having null constrains is to avoid incomplete data as that would cause issue to data analysis.
- Unique constrains were used to avoid duplication. (As an additional protection to front end development)
- Check constrains were used to protect the database from garbage values. As these are intended to be used in data visualization.
- DBA script for Creation of tables – Create Tables DBA Script

2.6. Data Generation and Loading

- Excel was used to initially create data
- Names, addresses were copied from the csv files downloaded from Kaggle.com
- We used online random timestamp generator for generating date and time
- In excel, Random functions, String concatenation were used to generate derived fields. Eg: Email addresses

- Note that the formulas were finally removed (using paste values) to avoid issues wile loading the data to SQL.
- Row counts-
  - Umember-5k
  - Visit-15k
  - Visit_location-8k
  - Cdeclaration-8k
  - Csdeclaration-2k
  - Cexternaltest-3k
  - Ctestlocation-2k
  - Pass-8k
  - Pass_qrcode-2.5k
  - Campus- 3 records
  - Building-12 records
  - Classroom- 29 records
- Data was uploaded to Oracle SQL using “Data Impot” wizard
- Note that all the records mentioned above have been loaded to the group database account.
- Data File- Data File

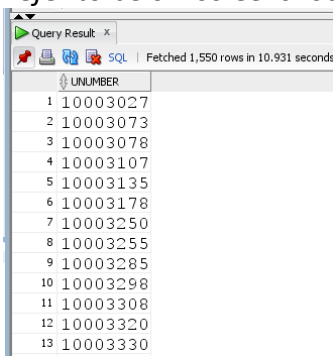
3. Query Writing

Categories	Query and Result
Student Enrollments & Campus Visits	**1) How many members were issued a pass in the month of March?** SELECT DISTINCT COUNT (unumber) FROM visit vs INNER JOIN pass p ON p.record_id=vs.record_id WHERE vs.record_timestamp LIKE '%MAR%' AND p.pass_issue = 'Y'; 

2) List the U# of all the members who never visited campus in the last 3 months.

```
SELECT DISTINCT unumber
FROM
    visit
WHERE
     campus_visiting_yn = 'N' AND record_timestamp between TO_DATE ('2020-07-15T00:00:00', 'YYYY-MM-DD"T"HH24:MI:SS') AND TO_DATE('2020-10-15T00:00:00', 'YYYY-MM-DD"T"HH24:MI:SS');
```

(Please refer to below screenshot for an excerpt of query result)



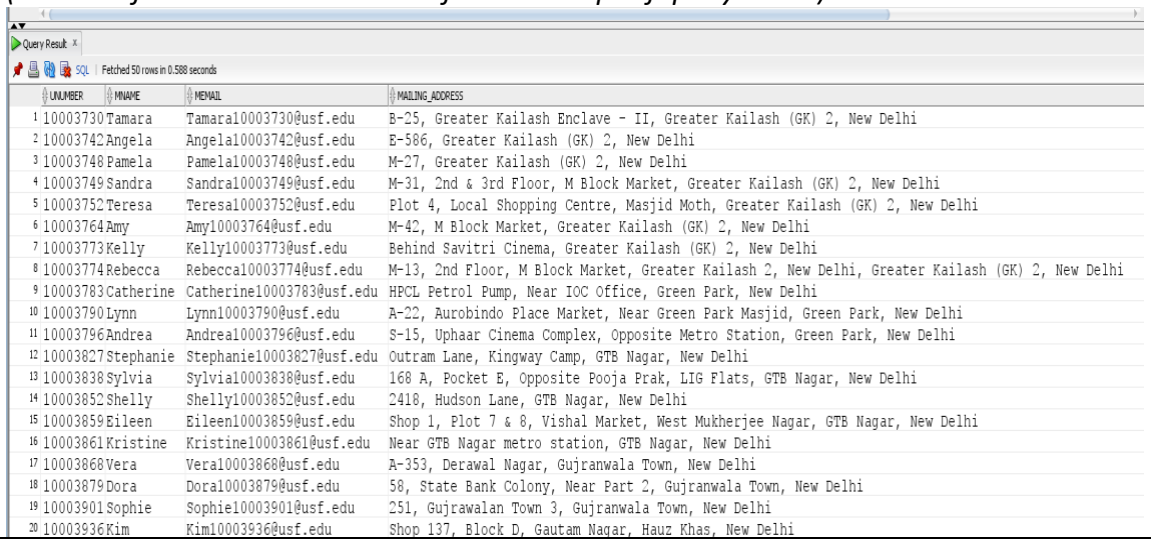
Query Result | SQL | Fetched 1,550 rows in 10.931 seconds

	UNUMBER
1	10003027
2	10003073
3	10003078
4	10003107
5	10003135
6	10003178
7	10003250
8	10003255
9	10003285
10	10003298
11	10003308
12	10003320
13	10003330

3) Display the student details who enrolled in fall 2020 but never visited campus.

```
SELECT DISTINCT um.unumber, um.mname, um.memail, um.mailing_address, um.joining_date, um.base_campus
FROM
    umember um
INNER JOIN visit vs
    ON vs.unumber=um.unumber
WHERE
    um.mrole='Student' AND vs.campus_visiting_yn = 'N' AND joining_date LIKE '%AUG%';
```

(Please refer to below screenshot for an excerpt of query result)



Query Result | SQL | Fetched 50 rows in 0.588 seconds

	UNUMBER	MNAME	MEMAIL	MAILING_ADDRESS
1	10003730	Tamara	Tamara10003730@usf.edu	B-25, Greater Kailash Enclave - II, Greater Kailash (GK) 2, New Delhi
2	10003742	Angela	Angela10003742@usf.edu	E-586, Greater Kailash (GK) 2, New Delhi
3	10003748	Pamela	Pamela10003748@usf.edu	M-27, Greater Kailash (GK) 2, New Delhi
4	10003749	Sandra	Sandra10003749@usf.edu	M-31, 2nd & 3rd Floor, M Block Market, Greater Kailash (GK) 2, New Delhi
5	10003752	Teresa	Teresa10003752@usf.edu	Plot 4, Local Shopping Centre, Masjid Moth, Greater Kailash (GK) 2, New Delhi
6	10003764	Amy	Amy10003764@usf.edu	M-42, M Block Market, Greater Kailash (GK) 2, New Delhi
7	10003773	Kelly	Kelly10003773@usf.edu	Behind Savitri Cinema, Greater Kailash (GK) 2, New Delhi
8	10003774	Rebecca	Rebecca10003774@usf.edu	M-13, 2nd Floor, M Block Market, Greater Kailash 2, New Delhi, Greater Kailash (GK) 2, New Delhi
9	10003783	Catherine	Catherine10003783@usf.edu	HPCL Petrol Pump, Near IOC Office, Green Park, New Delhi
10	10003790	Lynn	Lynn10003790@usf.edu	A-22, Aurobindo Place Market, Near Green Park Masjid, Green Park, New Delhi
11	10003796	Andrea	Andrea10003796@usf.edu	S-15, Uphaar Cinema Complex, Opposite Metro Station, Green Park, New Delhi
12	10003827	Stephanie	Stephanie10003827@usf.edu	Outram Lane, Kingway Camp, GTB Nagar, New Delhi
13	10003838	Sylvia	Sylvia10003838@usf.edu	168 A, Pocket E, Opposite Pooja Prak, LIG Flats, GTB Nagar, New Delhi
14	10003852	Shelly	Shelly10003852@usf.edu	2418, Hudson Lane, GTB Nagar, New Delhi
15	10003859	Eileen	Eileen10003859@usf.edu	Shop 1, Plot 7 & 8, Vishal Market, West Mukherjee Nagar, GTB Nagar, New Delhi
16	10003861	Kristine	Kristine10003861@usf.edu	Near GTB Nagar metro station, GTB Nagar, New Delhi
17	10003868	Vera	Vera10003868@usf.edu	A-353, Derawal Nagar, Gujranwala Town, New Delhi
18	10003879	Dora	Dora10003879@usf.edu	58, State Bank Colony, Near Part 2, Gujranwala Town, New Delhi
19	10003901	Sophie	Sophie10003901@usf.edu	251, Gujrawalan Town 3, Gujranwala Town, New Delhi
20	10003936	Kim	Kim10003936@usf.edu	Shop 137, Block D, Gautam Nagar, Hauz Khas, New Delhi

4) List of all members who visited Tampa campus in October 2020 & show temperature symptom.

```
SELECT um.mname, um.unumber
FROM
   visit    vs
 INNER JOIN visit_location   vsl

            ON vs.record_id = vsl.record_id
                        AND vs.campus_visiting_yn = 'Y'
                        AND CAST(record_timestamp AS DATE) BETWEEN DATE
'2020-10-01' AND DATE '2020-10-31'
 INNER JOIN campus cm
        ON cm.campus_id=vsl.campus_id
```

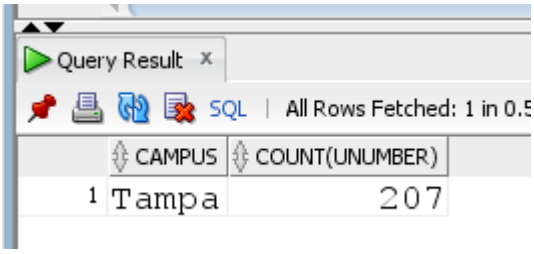
Query Result | SQL | All Rows Fetched: 19 in 0.

	MNAME	UNUMBER
1	Minnie	10000068
2	Dorothy	10000072
3	Sophie	10000106
4	Mary	10000126
5	Gladys	10000141
6	Helen	10000155
7	Martha	10000157
8	Elizabeth	10000158
9	Alice	10000163
10	Betty	10000167
11	Mary	10000423
12	Clara	10000446
13	Cecelia	10000458
14	Shirley	10000468
15	Shirley	10000481
16	Dorothy	10000485
17	Ruth	10000526
18	Betty	10000534
19	Evelyn	10000548

5)Visiting which campus has people showing all COVID symptoms.

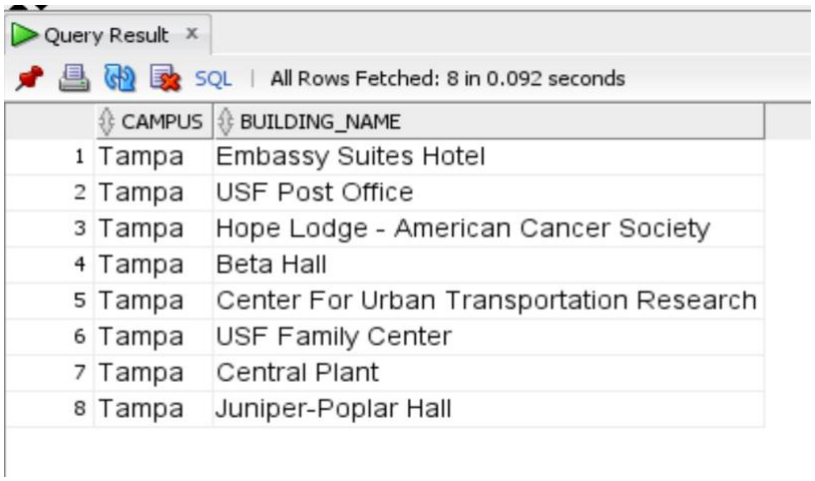
```
SELECT
   campus,
   COUNT (unumber)
FROM
   (
      SELECT
         um.unumber,
         vsl.campus
      FROM
         umember   um
         INNER JOIN visit vs
         ON um.unumber = vs.unumber
         AND vs.campus_visiting_yn = 'Y'
         INNER JOIN visit_location vsl
         on vs.record_id = vsl.record_id
```

```
            INNER JOIN campus cm
            on cm.campus_id=vsl.campus_id
            INNER JOIN (
              SELECT
                record_id
              FROM
                csdeclaration
              WHERE
                symptom_temperature = 'Y'
                AND symptom_cough = 'Y'
                AND symptom_breath = 'Y'
                AND symptom_headache = 'Y'
                AND symptom_fatigue = 'Y'
                AND symptom_nausea = 'Y'
                AND symptom_stomachache = 'Y'
                AND symptom_dizziness = 'Y'
                AND symptom_sorethroat = 'Y'
                AND symptom_lossofsmell = 'Y'
            ) csd ON vs.record_id = csd.record_id
        ) x
GROUP BY
    Campus
```

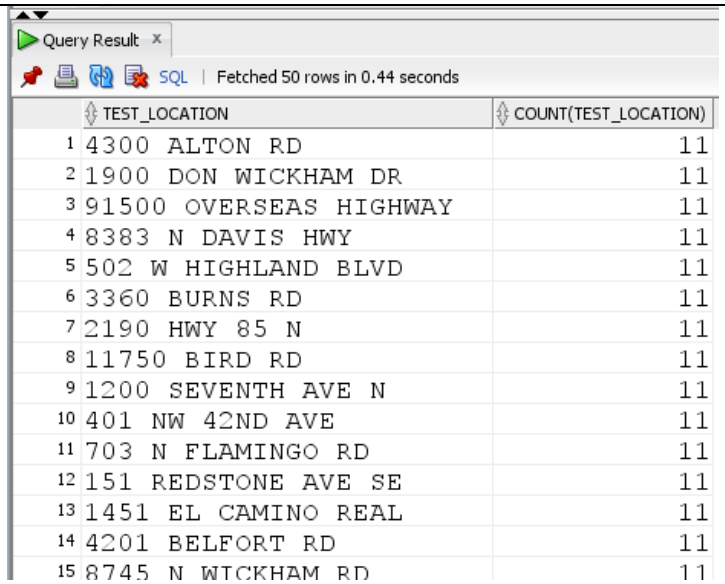


6)Which campus & building did the students who were issued pass visit in January 2020?

```
SELECT DISTINCT campus, building_name
FROM
   visit  vs
 INNER JOIN visit_location   vsl
            ON vs.record_id = vsl.record_id
                        AND vs.campus_visiting_yn = 'Y'
                       AND CAST(record_timestamp AS DATE) BETWEEN DATE
'2020-01-01' AND DATE '2020-01-31'
INNER JOIN campus cm
          ON cm.campus_id=vsl.campus_id
INNER JOIN building bl
```

	on bl.campus_id=cm.campus_id INNER JOIN umember um ON um.unumber = vs.unumber AND um.mrole = 'Student' INNER JOIN pass p ON vs.record_id = p.record_id AND p.pass_issue= 'Y' 
COVID Test Locations	**1) Display list of COVID testing locations. Show the most visited ones first.** SELECT test_location, count(test_location) FROM ctestlocation GROUP BY test_location ORDER BY count(test_location) DESC; *(Please refer to below screenshot for an excerpt of query result)*

	CAMPUS	BUILDING_NAME
1	Tampa	Embassy Suites Hotel
2	Tampa	USF Post Office
3	Tampa	Hope Lodge - American Cancer Society
4	Tampa	Beta Hall
5	Tampa	Center For Urban Transportation Research
6	Tampa	USF Family Center
7	Tampa	Central Plant
8	Tampa	Juniper-Poplar Hall



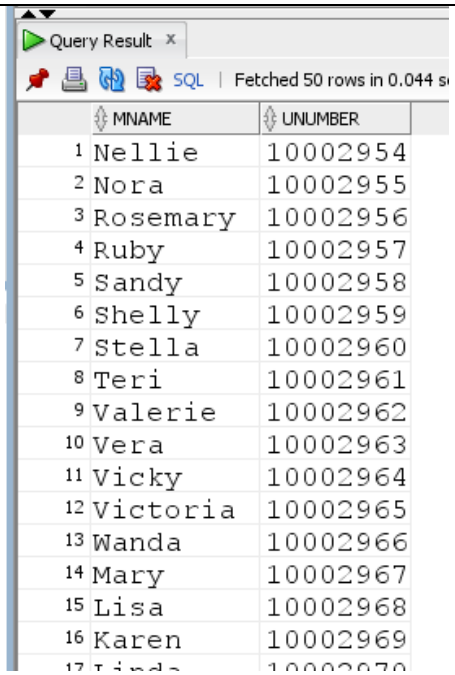
Query Result

SQL | Fetched 50 rows in 0.44 seconds

	TEST_LOCATION	COUNT(TEST_LOCATION)
1	4300 ALTON RD	11
2	1900 DON WICKHAM DR	11
3	91500 OVERSEAS HIGHWAY	11
4	8383 N DAVIS HWY	11
5	502 W HIGHLAND BLVD	11
6	3360 BURNS RD	11
7	2190 HWY 85 N	11
8	11750 BIRD RD	11
9	1200 SEVENTH AVE N	11
10	401 NW 42ND AVE	11
11	703 N FLAMINGO RD	11
12	151 REDSTONE AVE SE	11
13	1451 EL CAMINO REAL	11
14	4201 BELFORT RD	11
15	8745 N WICKHAM RD	11

2) List of all people who got tested in external location after coming into contact with COVID-positive person.

```
SELECT  um.mname, um.unumber
        FROM    cdeclaration    cd
        INNER JOIN cexternaltest   ce
                ON cd.record_id = ce.record_id
                AND cd.contact_with_positive = 'Y'
         INNER JOIN visit vs
                ON vs.record_id = ce.record_id
         INNER JOIN umember  um
                ON vs.unumber = um.unumber
```



Query Result | SQL | Fetched 50 rows in 0.044 s

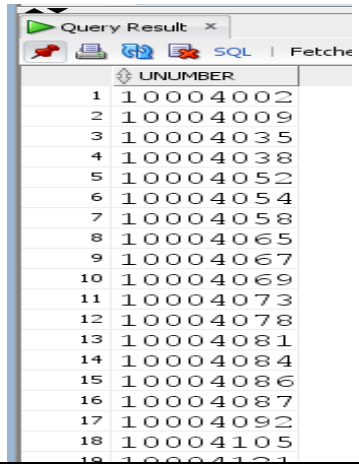
	MNAME	UNUMBER
1	Nellie	10002954
2	Nora	10002955
3	Rosemary	10002956
4	Ruby	10002957
5	Sandy	10002958
6	Shelly	10002959
7	Stella	10002960
8	Teri	10002961
9	Valerie	10002962
10	Vera	10002963
11	Vicky	10002964
12	Victoria	10002965
13	Wanda	10002966
14	Mary	10002967
15	Lisa	10002968
16	Karen	10002969

Student Health (COVID Symptoms)

1) List the members who did not contact COVID positive person but are physically not fine.

```
SELECT DISTINCT (unumber)
FROM
    visit vs
INNER JOIN
    cdeclaration cd
    ON cd.record_id=vs.record_id
WHERE
     contact_with_positive = 'N' AND physically_fine = 'N';
```

(Please refer to below screenshot for an excerpt of query result)

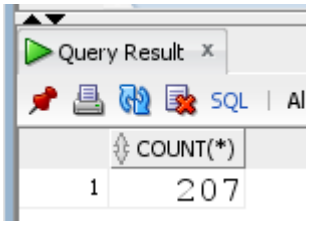


Query Result | SQL | Fetche

	UNUMBER
1	10004002
2	10004009
3	10004035
4	10004038
5	10004052
6	10004054
7	10004058
8	10004065
9	10004067
10	10004069
11	10004073
12	10004078
13	10004081
14	10004084
15	10004086
16	10004087
17	10004092
18	10004105

2) List the number of people who showed all the COVID symptoms when they were not physically fine.

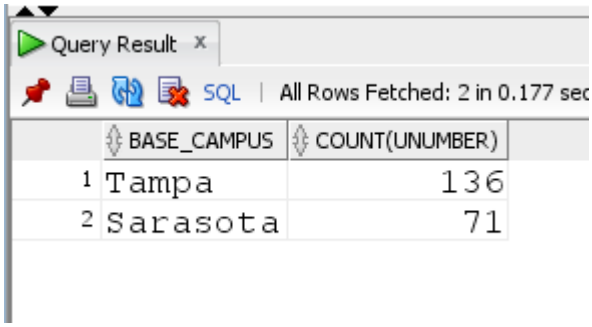
```
SELECT  COUNT (*)
FROM
    csdeclaration csd
INNER JOIN cdeclaration cd
        ON csd.record_id=cd.record_id
WHERE
        physically_fine = 'N' AND
symptom_temperature ='Y' AND
symptom_cough ='Y' AND
symptom_breath ='Y' AND
symptom_headache ='Y' AND
symptom_fatigue ='Y' AND
symptom_nausea ='Y' AND
symptom_stomachache ='Y' AND
symptom_dizziness ='Y' AND
symptom_sorethroat ='Y' AND
symptom_lossofsmell ='Y';
```



3) Which campus has the most number of people showing all COVID symptoms?

```
SELECT
   base_campus, COUNT(unumber)
FROM
   (
      SELECT
         um.unumber,
         um.base_campus
      FROM
         umember   um
         INNER JOIN visit vs
         ON um.unumber = vs.unumber
         AND vs.campus_visiting_yn = 'Y'
         INNER JOIN (
```

```
            SELECT   record_id FROM  csdeclaration
            WHERE
               symptom_temperature = 'Y'
               AND symptom_cough = 'Y'
               AND symptom_breath = 'Y'
               AND symptom_headache = 'Y'
               AND symptom_fatigue = 'Y'
               AND symptom_nausea = 'Y'
               AND symptom_stomachache = 'Y'
               AND symptom_dizziness = 'Y'
               AND symptom_sorethroat = 'Y'
               AND symptom_lossofsmell = 'Y'
         ) csd ON vs.record_id = csd.record_id
   ) x
GROUP BY    base_campus
```



4) When did the person who showed all COVID symptoms last visit the campus. Also, provide the location of the campus.

```
SELECT
   um.unumber,
   um.mname,
   vst.curr_visit_timestamp,
   vst.prev_visit_timestamp,
   vst.curr_visit_campus,
   vst.prev_visit_campus
FROM
   (
      SELECT
         record_id
      FROM
         csdeclaration
      WHERE
```

```
        symptom_temperature = 'Y'
        AND symptom_cough = 'Y'
        AND symptom_breath = 'Y'
        AND symptom_headache = 'Y'
        AND symptom_fatigue = 'Y'
        AND symptom_nausea = 'Y'
        AND symptom_stomachache = 'Y'
        AND symptom_dizziness = 'Y'
        AND symptom_sorethroat = 'Y'
        AND symptom_lossofsmell = 'Y'
    ) sym
    INNER JOIN (
      SELECT
        record_id,
        unumber,
        record_timestamp   AS curr_visit_timestamp,
        LAG(record_timestamp) OVER(
          PARTITION BY unumber
          ORDER BY
            record_timestamp DESC
        ) AS prev_visit_timestamp,
        campus  AS curr_visit_campus,
        LAG(campus) OVER(
          PARTITION BY unumber
          ORDER BY
            record_timestamp DESC
        ) AS prev_visit_campus
      FROM
        (
          SELECT
            vs.record_id,
            vs.unumber,
            vs.record_timestamp,
            cm.campus
          FROM
            visit  vs
            INNER JOIN visit_location   vsl
            ON vs.record_id = vsl.record_id
            AND vs.campus_visiting_yn = 'Y'
            INNER JOIN campus cm
            ON cm.campus_id = vsl.campus_id
        ) temp
    ) vst
    ON sym.record_id = vst.record_id
```

```
    AND vst.prev_visit_timestamp IS NOT NULL
INNER JOIN umember um
ON vst.unumber = um.unumber
```

Query Result

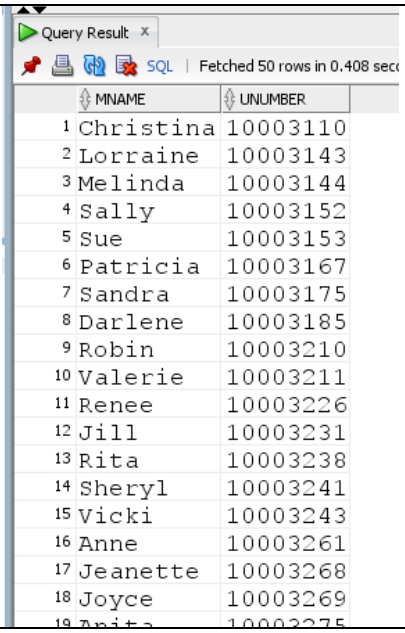
SQL | Fetched 50 rows in 0.062 seconds

	UNUMBER	MNAME	CURR_VISIT_TIMESTAMP	PREV_VISIT_TIMESTAMP	CURR_VISIT_CAMPUS	PREV_VISIT_CAMPUS
1	10000003	Anna	27-MAY-20 09.33.30.000000000 PM	09-SEP-20 07.10.27.000000000 AM	Tampa	Tampa
2	10000004	Margaret	15-JUN-20 09.18.24.000000000 AM	27-SEP-20 07.08.13.000000000 PM	Tampa	Tampa
3	10000005	Helen	22-MAY-20 04.13.26.000000000 PM	27-AUG-20 04.54.37.000000000 PM	Tampa	Tampa
4	10000007	Lucy	09-FEB-20 05.18.52.000000000 PM	14-FEB-20 05.40.35.000000000 PM	Tampa	Tampa
5	10000011	Ruth	01-MAR-20 10.26.28.000000000 AM	10-SEP-20 12.33.25.000000000 PM	Tampa	Tampa
6	10000018	Anna	07-FEB-20 09.55.58.000000000 PM	06-JUL-20 05.30.05.000000000 PM	Tampa	Tampa
7	10000021	Jean	19-MAR-20 12.24.53.000000000 AM	04-AUG-20 06.15.00.000000000 AM	Tampa	Tampa
8	10000022	Ruth	13-JUL-20 08.52.36.000000000 AM	30-JUL-20 07.41.04.000000000 AM	Tampa	Tampa
9	10000024	Esther	17-JAN-20 03.58.43.000000000 PM	26-AUG-20 10.07.10.000000000 PM	Tampa	Tampa
10	10000026	Margaret	24-MAR-20 04.26.31.000000000 AM	14-OCT-20 02.22.17.000000000 PM	Tampa	Tampa
11	10000027	Marie	23-APR-20 09.23.39.000000000 AM	13-JUN-20 03.26.12.000000000 PM	Tampa	Tampa
12	10000028	Mary	12-JUL-20 05.55.38.000000000 AM	17-JUL-20 09.57.05.000000000 PM	Tampa	Tampa
13	10000029	Elizabeth	24-FEB-20 08.16.30.000000000 AM	01-JUN-20 05.37.20.000000000 AM	Tampa	Tampa
14	10000031	Helen	22-FEB-20 05.13.30.000000000 AM	11-SEP-20 09.51.28.000000000 AM	Tampa	Tampa
15	10000032	Alice	09-JAN-20 11.19.48.000000000 PM	17-AUG-20 01.01.52.000000000 AM	Tampa	Tampa

Faculty visits

1) List of the entire faculty who visited Tampa campus in August 2020.

```
SELECT
   um.mname, um.unumber
FROM
   visit vs
 INNER JOIN visit_location   vsl
          ON vs.record_id = vsl.record_id
          AND vs.campus_visiting_yn = 'Y'
          AND CAST(vs.record_timestamp AS DATE) BETWEEN DATE '2020-08-01'
AND DATE '2020-08-31'
 INNER JOIN campus cm
         ON cm.campus_id=vsl.campus_id
          AND campus = 'Tampa'
 INNER JOIN umember um
          ON um.unumber = vs.unumber
          AND um.mrole = 'Faculty';
```



4. Performance Tuning

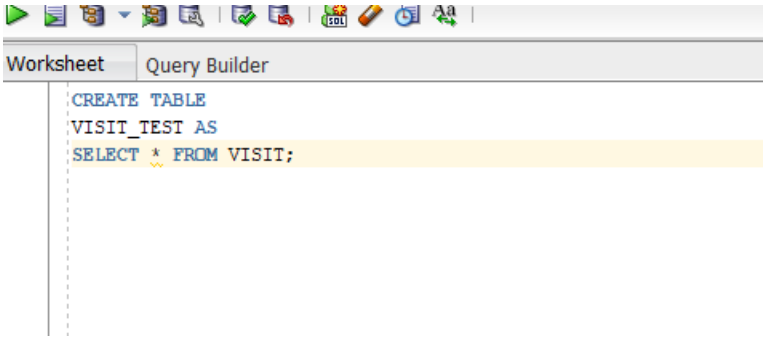
4.1. Indexing

Indexes are associated with tables and are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries. Indexes are one of many means of reducing disk I/O and improve query performance.

Demonstrating the concept with COVID Symptom database:

Procedure:

1. Create duplicate table VISIT_TEST for VISIT table:



2. Checking the Index on VISIT_TEST table.

   No results as there are no Indexes

```
SELECT *
FROM ALL_INDEXES
WHERE TABLE_NAME='VISIT_TEST';
```

Query Result

SQL | All Rows Fetched: 0 in 0.488 seconds

OWNER	INDEX_...	INDEX_...	TABLE_...	TABLE_...	TABLE_...	UNIQU...	COMP...	PREFIX...	TABLE...	INI_TR...	MAX_T...	INITIAL...	NEXT_...	MIN_E...	MAX_E...	PCT_IN...

3. Performance of the query before creating the Index.

Time taken: 0.153seconds

```
SELECT *
FROM ALL_INDEXES
WHERE TABLE_NAME='VISIT_TEST';

SELECT *
FROM visit_test
WHERE RECORD_ID BETWEEN 202000001 AND 202001000;
```

Query Result

SQL | Fetched 50 rows in 0.153 seconds

	RECORD_ID	RECORD_TIMESTAMP	UNUMBER	CAMPUS_VISITING_YN
1	202000830	27-JAN-20 08.05.25.000000000	10000830	Y
2	202000831	05-JUN-20 21.55.15.000000000	10000831	Y
3	202000832	18-MAY-20 08.06.36.000000000	10000832	Y
4	202000833	06-FEB-20 05.17.48.000000000	10000833	Y
5	202000834	21-JUN-20 03.48.50.000000000	10000834	Y

4.Creating an index on column RECORD_ID:

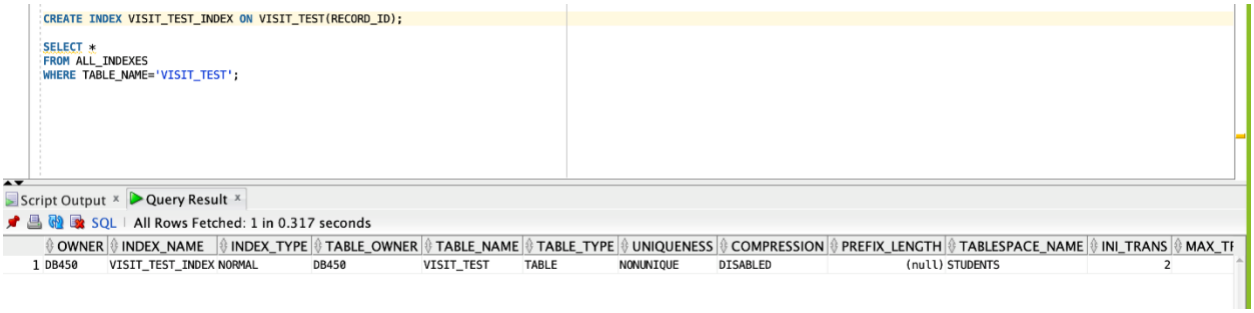
```
CREATE INDEX VISIT_TEST_INDEX ON VISIT_TEST(RECORD_ID);

SELECT *
FROM ALL_INDEXES
WHERE TABLE_NAME='VISIT_TEST';
```

Script Output | Query Result

SQL | All Rows Fetched: 1 in 0.317 seconds

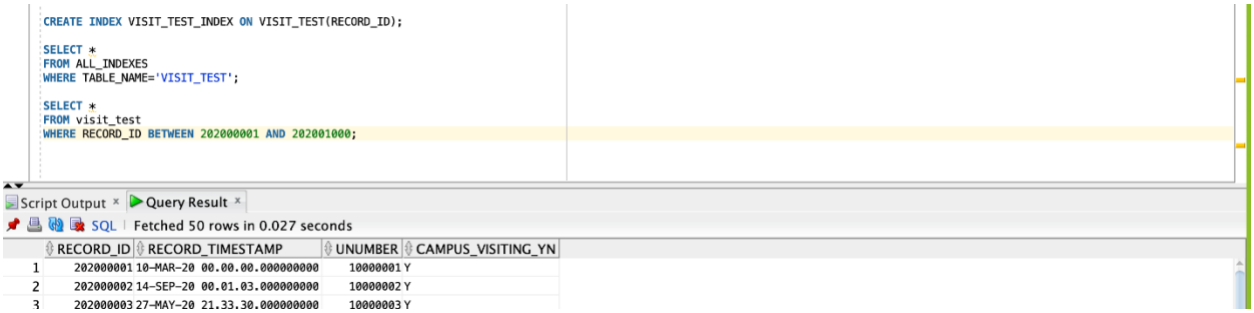
	OWNER	INDEX_NAME	INDEX_TYPE	TABLE_OWNER	TABLE_NAME	TABLE_TYPE	UNIQUENESS	COMPRESSION	PREFIX_LENGTH	TABLESPACE_NAME	INI_TRANS	MAX_TI
1	DB450	VISIT_TEST_INDEX	NORMAL	DB450	VISIT_TEST	TABLE	NONUNIQUE	DISABLED	(null)	STUDENTS	2	

5. Checking the index again on VISIT_TEST. From query result, we can see a record of index now:



6. Performance of the query after creating the Index.

Time taken: 0.027 seconds.



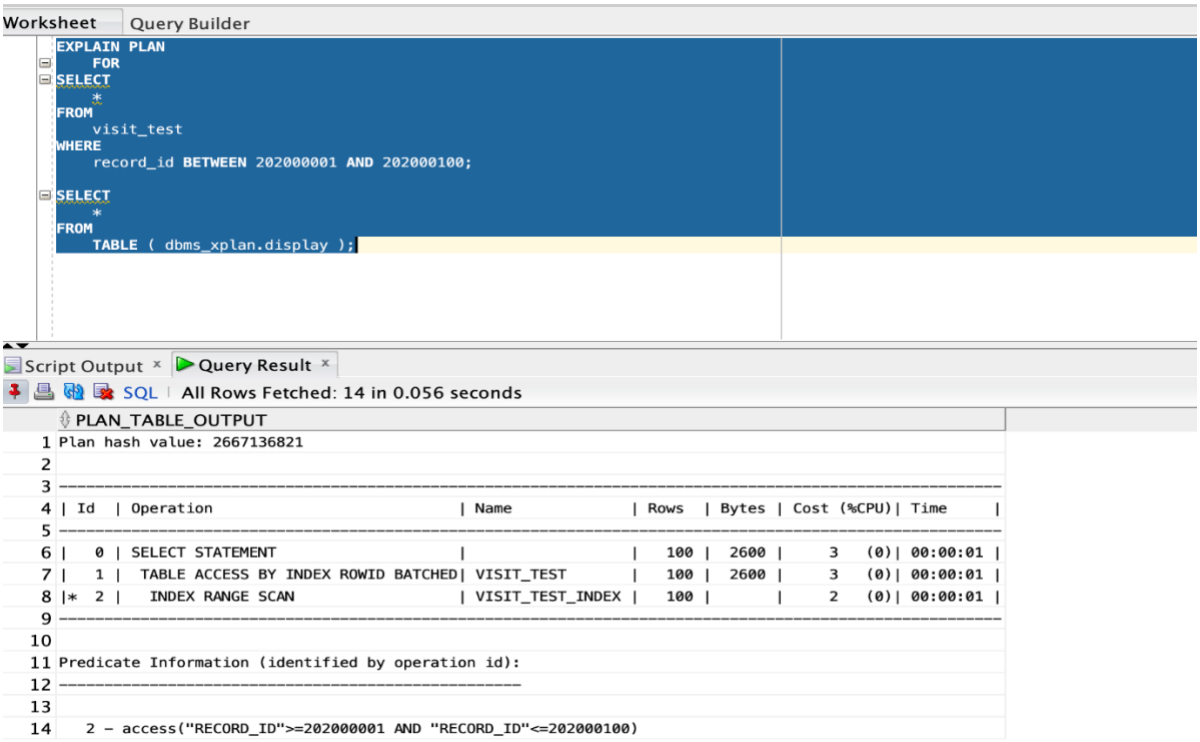
From above results, we can clearly see that Indexing helps in optimizing query execution time. ***we can see that performance is improved.***

Query	Time
Before Indexing	0.153 seconds
After Indexing	0.027 seconds
Performance benefit	0.126 seconds

4.2. Execution Plan

This plan shows execution of the SELECT statement. The table VISIT_TEST is being accessed by using a full table scan.

- Every row in the table VISIT_TEST is accessed.
- For every row, the WHERE clause criteria is evaluated.
- The SELECT statement returns the rows meeting the WHERE clause criteria.



Worksheet Query Builder

```
EXPLAIN PLAN
    FOR
SELECT
    *
FROM
    visit_test
WHERE
    record_id BETWEEN 202000001 AND 202000100;

SELECT
    *
FROM
    TABLE ( dbms_xplan.display );
```

Script Output × Query Result ×

SQL | All Rows Fetched: 14 in 0.056 seconds

```
PLAN_TABLE_OUTPUT
Plan hash value: 2667136821

-----------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name             | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |                  |   100 |  2600 |     3   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| VISIT_TEST       |   100 |  2600 |     3   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | VISIT_TEST_INDEX |   100 |       |     2   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("RECORD_ID">=202000001 AND "RECORD_ID"<=202000100)
```

4.3. Selectivity of the Index

The ratio of the number of distinct values in the indexed column / columns to the number of records in the table represents the selectivity of an index. The ideal selectivity is 1. Such a selectivity can be reached only by unique indexes on NOT NULL columns.

```
select
count (1)
from visit_test;
```

Script Output | Query Result | Query Result 1

SQL | All Rows Fetched: 1 in 0.027 seconds

	COUNT(1)
1	15000

```
select
count (distinct record_id)
from visit_test;
```

Script Output | Query Result | Query Result 1

SQL | All Rows Fetched: 1 in 0.036 seconds

	COUNT(DISTINCTRECORD_ID)
1	15000

SELECTIVITY = 15000 / 15000 = 1

The number of distinct values in the indexed column are 15000 and the number of records in the table are 15000. Since RECORD_ID is the primary key of the table, it has by default Unique and Not Null constraints applied. Hence 100 percent of rows in the table have distinct value for the indexed column.

4.4. Point Query

Point queries are highly targeted, returning a very small answer set (or even a single row).

Experiment: In the COVID Symptom Database, we are trying to retrieve a member whose first name is AGNES and who is a FACULTY. In the database, there is only one person with that specific value.

Part 1) All members with Name = Agnes, we got 34 of them

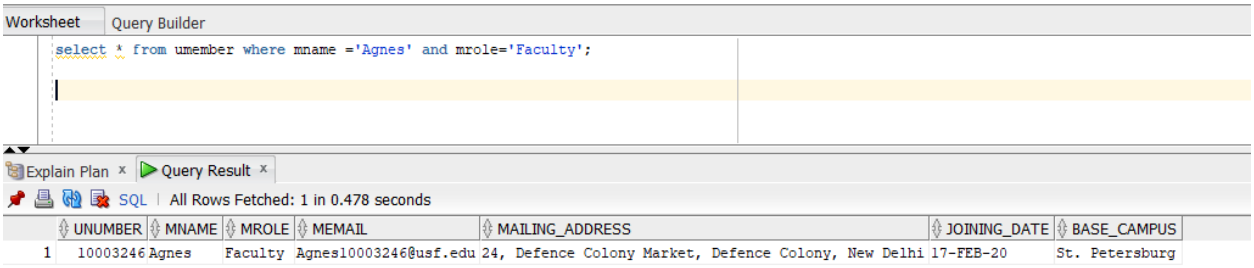
```
select * from umember where mname='Agnes' ;
```

Script Output | Query Result | Query Result 1

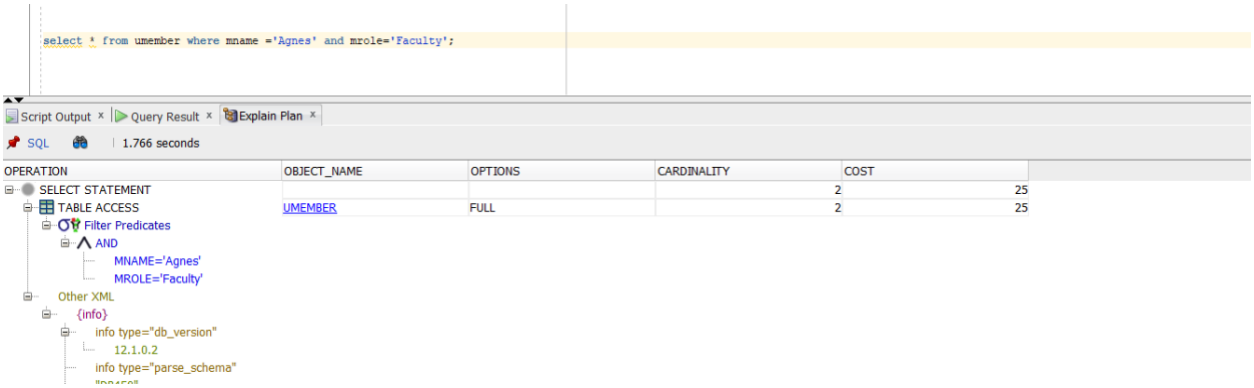
SQL | All Rows Fetched: 34 in 0.162 seconds

	UNUMBER	MNAME	MROLE	MEMAIL	MAILING_ADDRESS
1	10003074	Agnes	Student	Agnes10003074@usf.edu	N-16, N Block, Near Rajiv Chowk Metro Gate 7, Outer Circle, Connaught Pl
2	10003246	Agnes	Faculty	Agnes10003246@usf.edu	24, Defence Colony Market, Defence Colony, New Delhi
3	10002752	Agnes	Student	Agnes10002752@usf.edu	Shop 29, Ground Floor, Durga Place DDA Market, Ashok Vihar Phase 3, New
4	10000017	Agnes	Student	Agnes10000017@usf.edu	Ground Floor, Building G, Solenad 3, Nuvali, Don Jose, Santa Rosa
5	10000044	Agnes	Student	Agnes10000044@usf.edu	Rua GonÎ_alves Dias, 32, Centro, Rio de Janeiro
6	10000077	Agnes	Student	Agnes10000077@usf.edu	EdifÎ_cio ItÌÁlia - 41åi Andar, Avenida Ipiranga, 344, RepÌ_blica, SÌ£o
7	10000137	Agnes	Student	Agnes10000137@usf.edu	Augusta Mall, Augusta, GA 30909

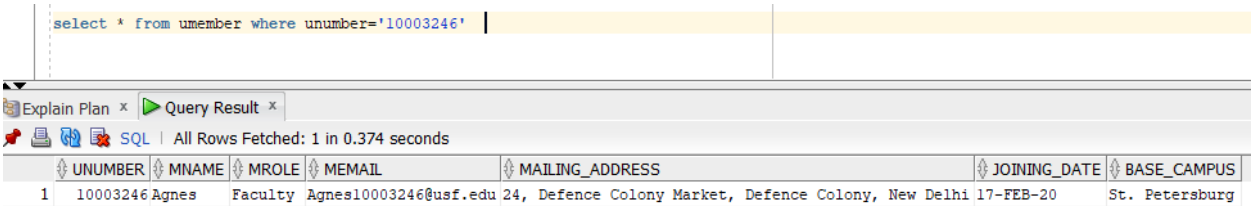
Part 2) We now trying to retrieve member whose Name is ‘Agnes’ and role is ‘Faculty’

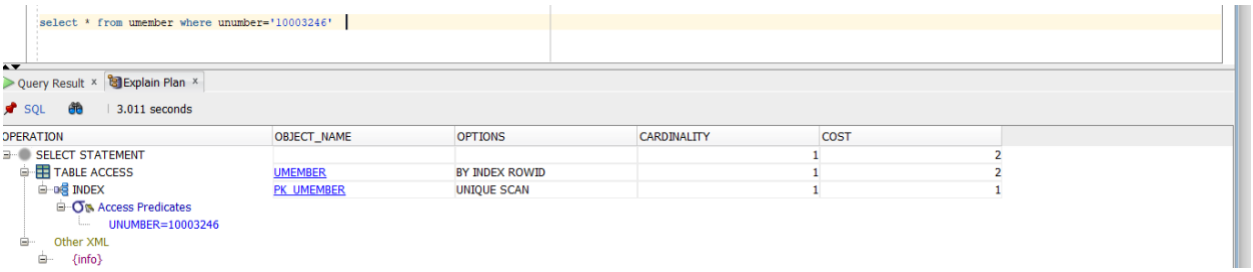


From the execution plan, we see that no index is not accessed while performing this task.



Part 3) Similarly, the single row result set is found by using the primary key (UMEMBER_ID). We can see that this query benefitted from index that is available since this is a primary key column.





Conclusion:

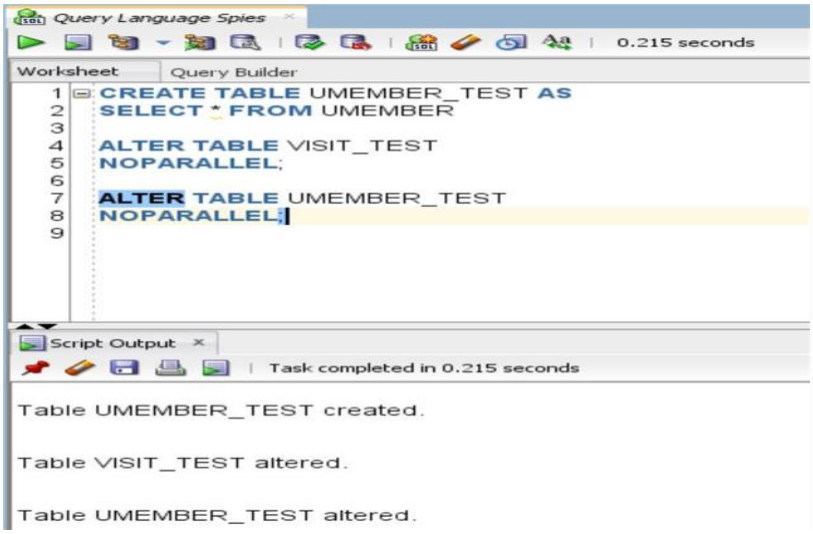
From above results, it is clear that there is an improvement(***cost and performance***) when we use the column UNUMBER_ID (which has an Index because it is a Primary Key)

Query	Time	Cost
Before Indexing	0.478 seconds	25+25=50
After Indexing	0.374 seconds	2+2+1 = 5
Performance benefit	0.104 seconds	45

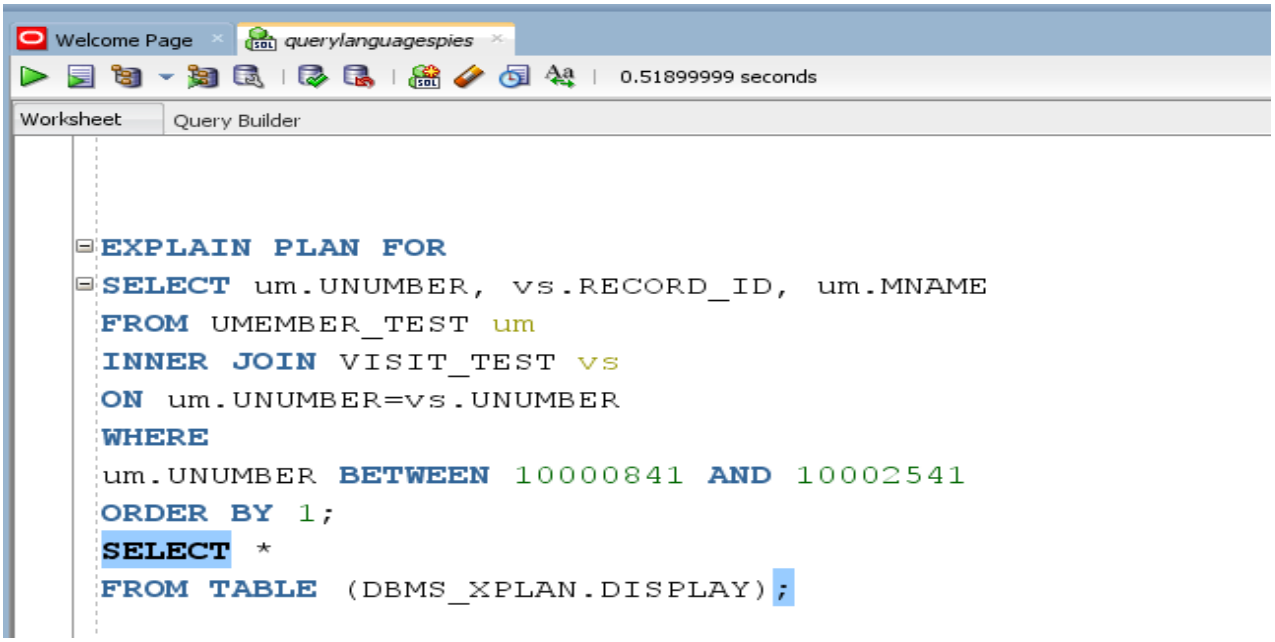
4.5. Parallelism

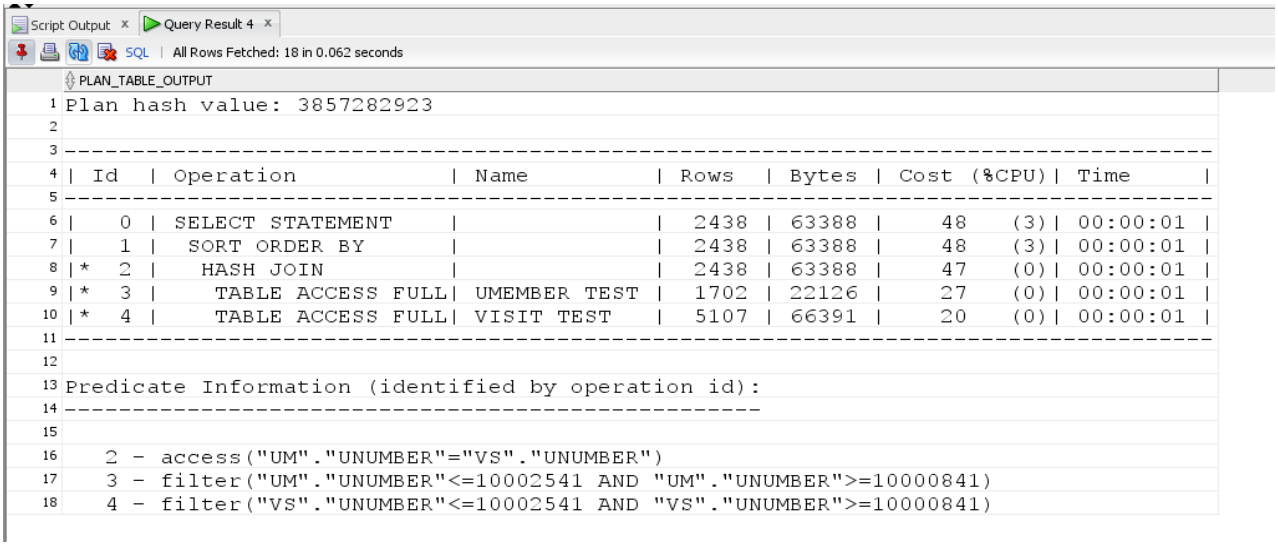
Parallel computing aims at improving performance by applying multiple CPU and I/O resources to the execution of SQL statements. It decomposes tasks into simpler sub tasks thereby increasing output for a group of transactions and response time for a single transaction.

1) Disabling any existing parallelism in the tables used in the query.

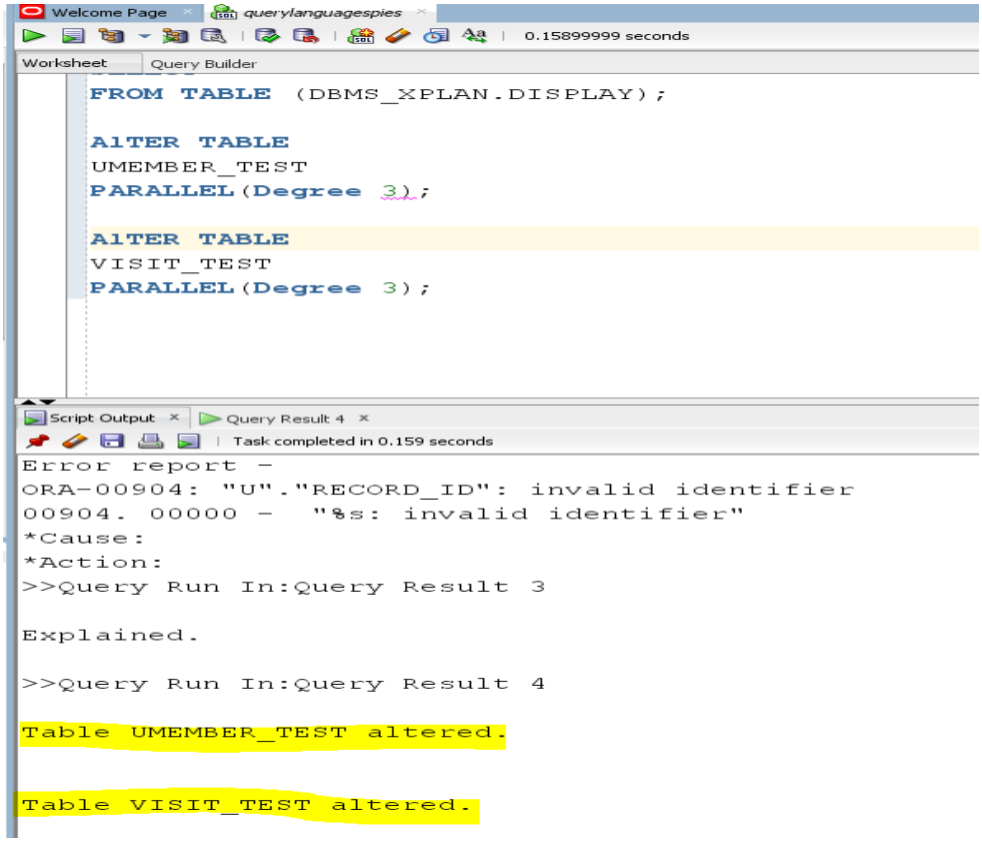


2) Generating Explain Plan for the Query.



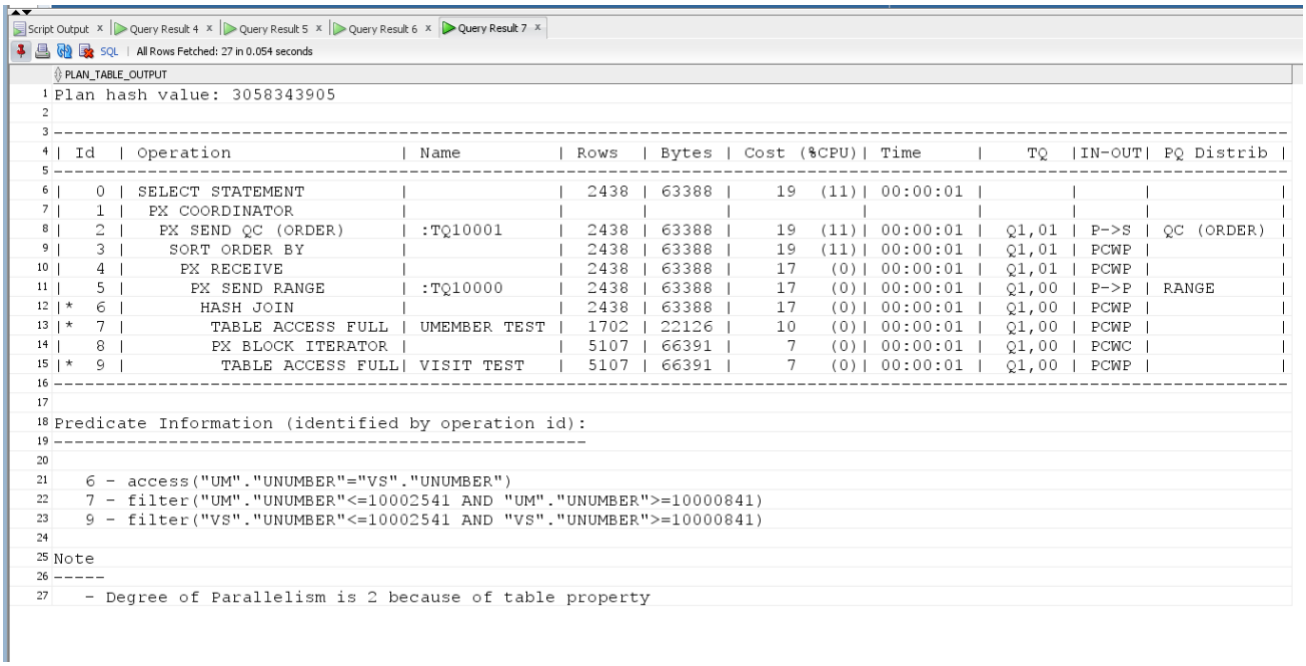


3) Enabling Parallelism. The degree of parallelism (DOP) is the number of parallel execution servers associated with a single operation. It is set to 3 now.



```
Welcome Page | querylanguagespies | 0.15899999 seconds
Worksheet | Query Builder

FROM TABLE (DBMS_XPLAN.DISPLAY);

AlTER TABLE
UMEMBER_TEST
PARALLEL(Degree 3);

AlTER TABLE
VISIT_TEST
PARALLEL(Degree 3);

Script Output | Query Result 4 | Task completed in 0.159 seconds
Error report -
ORA-00904: "U"."RECORD_ID": invalid identifier
00904. 00000 - "%s: invalid identifier"
*Cause:
*Action:
>>Query Run In:Query Result 3

Explained.

>>Query Run In:Query Result 4

Table UMEMBER_TEST altered.

Table VISIT_TEST altered.
```

4) Generating Explain Plan for the same query:



```
Script Output | Query Result 4 | Query Result 5 | Query Result 6 | Query Result 7
SQL | All Rows Fetched: 27 in 0.054 seconds
PLAN_TABLE_OUTPUT
Plan hash value: 3058343905

-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation                | Name          | Rows  | Bytes | Cost (%CPU)| Time     |    TQ  |IN-OUT| PQ Distrib |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |               |  2438 | 63388 |    19  (11)| 00:00:01 |        |      |            |
|   1 |  PX COORDINATOR          |               |       |       |            |          |        |      |            |
|   2 |   PX SEND QC (ORDER)     | :TQ10001      |  2438 | 63388 |    19  (11)| 00:00:01 |  Q1,01 | P->S | QC (ORDER) |
|   3 |    SORT ORDER BY         |               |  2438 | 63388 |    19  (11)| 00:00:01 |  Q1,01 | PCWP |            |
|   4 |     PX RECEIVE           |               |  2438 | 63388 |    17   (0)| 00:00:01 |  Q1,01 | PCWP |            |
|   5 |      PX SEND RANGE       | :TQ10000      |  2438 | 63388 |    17   (0)| 00:00:01 |  Q1,00 | P->P | RANGE      |
|*  6 |       HASH JOIN          |               |  2438 | 63388 |    17   (0)| 00:00:01 |  Q1,00 | PCWP |            |
|*  7 |        TABLE ACCESS FULL | UMEMBER_TEST  |  1702 | 22126 |    10   (0)| 00:00:01 |  Q1,00 | PCWP |            |
|   8 |        PX BLOCK ITERATOR |               |  5107 | 66391 |     7   (0)| 00:00:01 |  Q1,00 | PCWC |            |
|*  9 |         TABLE ACCESS FULL| VISIT_TEST    |  5107 | 66391 |     7   (0)| 00:00:01 |  Q1,00 | PCWP |            |
-----------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   6 - access("UM"."UNUMBER"="VS"."UNUMBER")
   7 - filter("UM"."UNUMBER"<=10002541 AND "UM"."UNUMBER">=10000841)
   9 - filter("VS"."UNUMBER"<=10002541 AND "VS"."UNUMBER">=10000841)

Note
-----
   - Degree of Parallelism is 2 because of table property
```

Conclusion:

From above results, we see notice that Parallelism indeed helped in optimizing the executing time of the query and the cost associated with it.

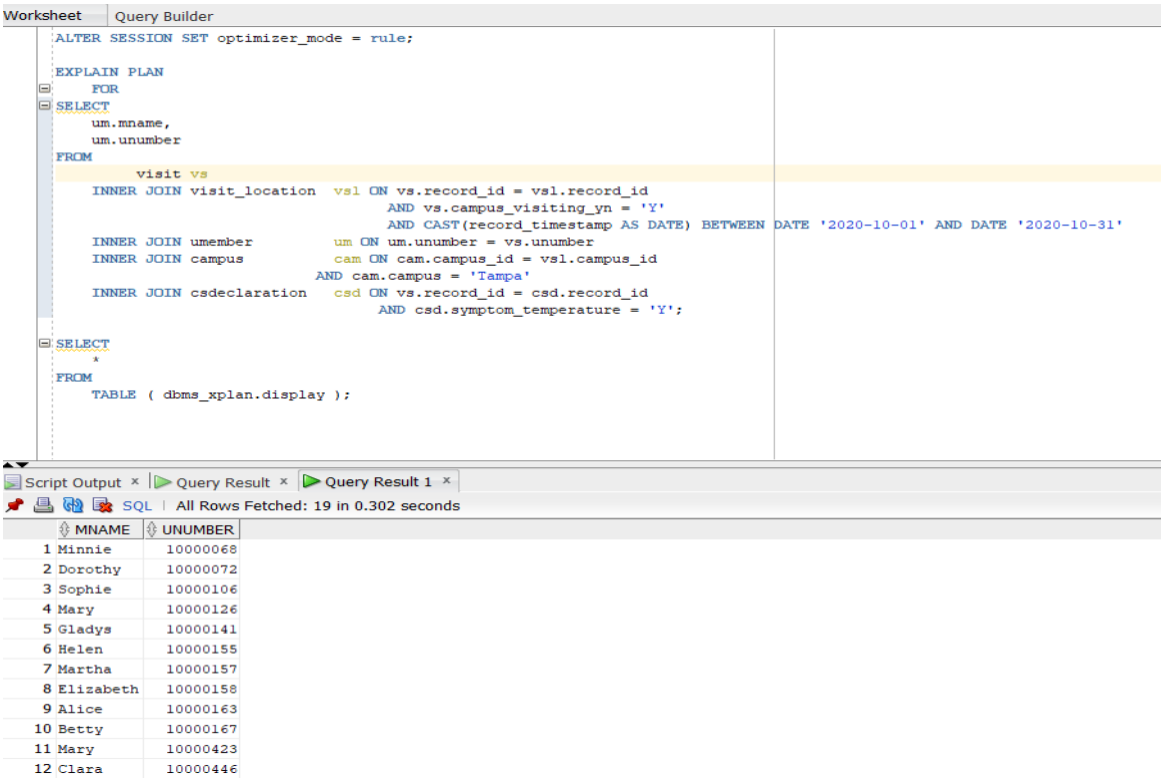
Query	Time	Cost
Before Parallel execution	0.062 seconds	48
After Parallel execution	0.054 seconds	19
Performance benefit	0.008 seconds	29

4.6. Optimizer Modes

1) **Rule Based Approach** – Now obsolete and unsupported but still allowed. Always uses rule-based optimization. Oracle uses heuristics from the data dictionary in order to determine the most effective way to service an Oracle query and translate the declarative SQL command into an actual navigation plan to extract the data. In many pre-Oracle8i systems, rule-based optimization is faster than cost-based.

   Rule-based shortcomings: Often chooses the wrong index to retrieve rows.

   We used one of the queries from Part 2. Query Writing which involved joining multiple tables:



```
ALTER SESSION SET optimizer_mode = rule;

EXPLAIN PLAN
    FOR
SELECT
    um.mname,
    um.unumber
FROM
        visit vs
    INNER JOIN visit_location  vsl ON vs.record_id = vsl.record_id
                                      AND vs.campus_visiting_yn = 'Y'
                                      AND CAST(record_timestamp AS DATE) BETWEEN DATE '2020-10-01' AND DATE '2020-10-31'
    INNER JOIN umember         um ON um.unumber = vs.unumber
    INNER JOIN campus          cam ON cam.campus_id = vsl.campus_id
                               AND cam.campus = 'Tampa'
    INNER JOIN csdeclaration   csd ON vs.record_id = csd.record_id
                                      AND csd.symptom_temperature = 'Y';

SELECT
    *
FROM
    TABLE ( dbms_xplan.display );
```

Script Output | Query Result | Query Result 1

All Rows Fetched: 19 in 0.302 seconds

	MNAME	UNUMBER
1	Minnie	10000068
2	Dorothy	10000072
3	Sophie	10000106
4	Mary	10000126
5	Gladys	10000141
6	Helen	10000155
7	Martha	10000157
8	Elizabeth	10000158
9	Alice	10000163
10	Betty	10000167
11	Mary	10000423
12	Clara	10000446

```
PLAN_TABLE_OUTPUT
Plan hash value: 3667324562

-------------------------------------------------------------------------
| Id  | Operation                          | Name                 |
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |                      |
|   1 |  NESTED LOOPS                      |                      |
|   2 |   NESTED LOOPS                     |                      |
|   3 |    NESTED LOOPS                    |                      |
|   4 |     NESTED LOOPS                   |                      |
|   5 |      NESTED LOOPS                  |                      |
|*  6 |       TABLE ACCESS FULL            | CSDECLARATION        |
|*  7 |       TABLE ACCESS BY INDEX ROWID  | VISIT                |
|*  8 |        INDEX UNIQUE SCAN           | PK_RECORD_ID         |
|   9 |      TABLE ACCESS BY INDEX ROWID   | UMEMBER              |
|* 10 |       INDEX UNIQUE SCAN            | PK_UMEMBER           |
|  11 |     TABLE ACCESS BY INDEX ROWID    | VISIT_LOCATION       |
|* 12 |      INDEX RANGE SCAN              | PK_RECORD_ID_BC_CID  |
|* 13 |    INDEX UNIQUE SCAN               | PK_CAMPUS_ID         |
|* 14 |   TABLE ACCESS BY INDEX ROWID      | CAMPUS               |
-------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   6 - filter("CSD"."SYMPTOM_TEMPERATURE"='Y')
   7 - filter(CAST(INTERNAL_FUNCTION("RECORD_TIMESTAMP") AS
              DATE)<=TO_DATE(' 2020-10-31 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              CAST(INTERNAL_FUNCTION("RECORD_TIMESTAMP") AS DATE)>=TO_DATE('
              2020-10-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              "VS"."CAMPUS_VISITING_YN"='Y')
   8 - access("VS"."RECORD_ID"="CSD"."RECORD_ID")
  10 - access("UM"."UNUMBER"="VS"."UNUMBER")
  12 - access("VS"."RECORD_ID"="VSL"."RECORD_ID")
  13 - access("CAM"."CAMPUS_ID"="VSL"."CAMPUS_ID")
  14 - filter("CAM"."CAMPUS"='Tampa')

Note
```

2) **Cost Based Approach –** This is a cost-based optimizer mode that ensures that the overall query time is minimized, even if it takes longer to receive the first row. This usually involves choosing a parallel full table scan over a full index scan. Because the ALL_ROWS mode favors full table scans, the ALL_ROWS mode is best suited for batch-oriented queries where intermediate rows are not required for viewing.

   Cost-based shortcomings Often performs unnecessary full tables scans, especially when more than three tables are being joined.

   We used the same query that we used for previous approach:

```
ALTER SESSION SET optimizer_mode = ALL_ROWS;

EXPLAIN PLAN
    FOR
SELECT
    um.mname,
    um.unumber
FROM
        visit vs
    INNER JOIN visit_location  vsl ON vs.record_id = vsl.record_id
                                      AND vs.campus_visiting_yn = 'Y'
                                      AND CAST(record_timestamp AS DATE) BETWEEN DATE '2020-10-01' AND DATE '2020-10-31'
    INNER JOIN umember          um ON um.unumber = vs.unumber
    INNER JOIN campus          cam ON cam.campus_id = vsl.campus_id
                                  AND cam.campus = 'Tampa'
    INNER JOIN csdeclaration   csd ON vs.record_id = csd.record_id
                                      AND csd.symptom_temperature = 'Y';

SELECT
    *
```

Script Output × | Query Result × | Query Result 1 ×

SQL | All Rows Fetched: 19 in 0.407 seconds

	MNAME	UNUMBER
1	Evelyn	10000548
2	Betty	10000534
3	Ruth	10000526
4	Dorothy	10000485
5	Shirley	10000481
6	Shirley	10000468
7	Cecelia	10000458
8	Clara	10000446
9	Mary	10000423
10	Betty	10000167
11	Alice	10000163
12	Elizabeth	10000158
13	Martha	10000157

PLAN_TABLE_OUTPUT

```
Plan hash value: 31833640

-------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                |    17 |  1241 |    57   (2)| 00:00:01 |
|   1 |  NESTED LOOPS                 |                |    17 |  1241 |    57   (2)| 00:00:01 |
|   2 |   NESTED LOOPS                |                |    17 |  1241 |    57   (2)| 00:00:01 |
|*  3 |    HASH JOIN                  |                |    17 |  1020 |    40   (3)| 00:00:01 |
|*  4 |     HASH JOIN                 |                |    52 |  2392 |    37   (3)| 00:00:01 |
|*  5 |      HASH JOIN                |                |    52 |  1820 |    27   (0)| 00:00:01 |
|*  6 |       TABLE ACCESS FULL       | VISIT          |   388 | 10088 |    22   (0)| 00:00:01 |
|*  7 |       TABLE ACCESS FULL       | CSDECLARATION  |  1053 |  9477 |     5   (0)| 00:00:01 |
|   8 |      TABLE ACCESS FULL        | VISIT_LOCATION |  8000 | 88000 |     9   (0)| 00:00:01 |
|*  9 |     TABLE ACCESS FULL         | CAMPUS         |     1 |    14 |     3   (0)| 00:00:01 |
|* 10 |    INDEX UNIQUE SCAN          | PK_UMEMBER     |     1 |       |     0   (0)| 00:00:01 |
|  11 |   TABLE ACCESS BY INDEX ROWID| UMEMBER        |     1 |    13 |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("CAM"."CAMPUS_ID"="VSL"."CAMPUS_ID")
   4 - access("VS"."RECORD_ID"="VSL"."RECORD_ID")
   5 - access("VS"."RECORD_ID"="CSD"."RECORD_ID")
   6 - filter("VS"."CAMPUS_VISITING_YN"='Y' AND
              CAST(INTERNAL_FUNCTION("RECORD_TIMESTAMP") AS DATE)>=TO_DATE(' 2020-10-01 00:00:00',
              'syyyy-mm-dd hh24:mi:ss') AND CAST(INTERNAL_FUNCTION("RECORD_TIMESTAMP") AS
              DATE)<=TO_DATE(' 2020-10-31 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
   7 - filter("CSD"."SYMPTOM_TEMPERATURE"='Y')
   9 - filter("CAM"."CAMPUS"='Tampa')
  10 - access("UM"."UNUMBER"="VS"."UNUMBER")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - this is an adaptive plan
   - 2 Sql Plan Directives used for this statement
```

5. Other Topics

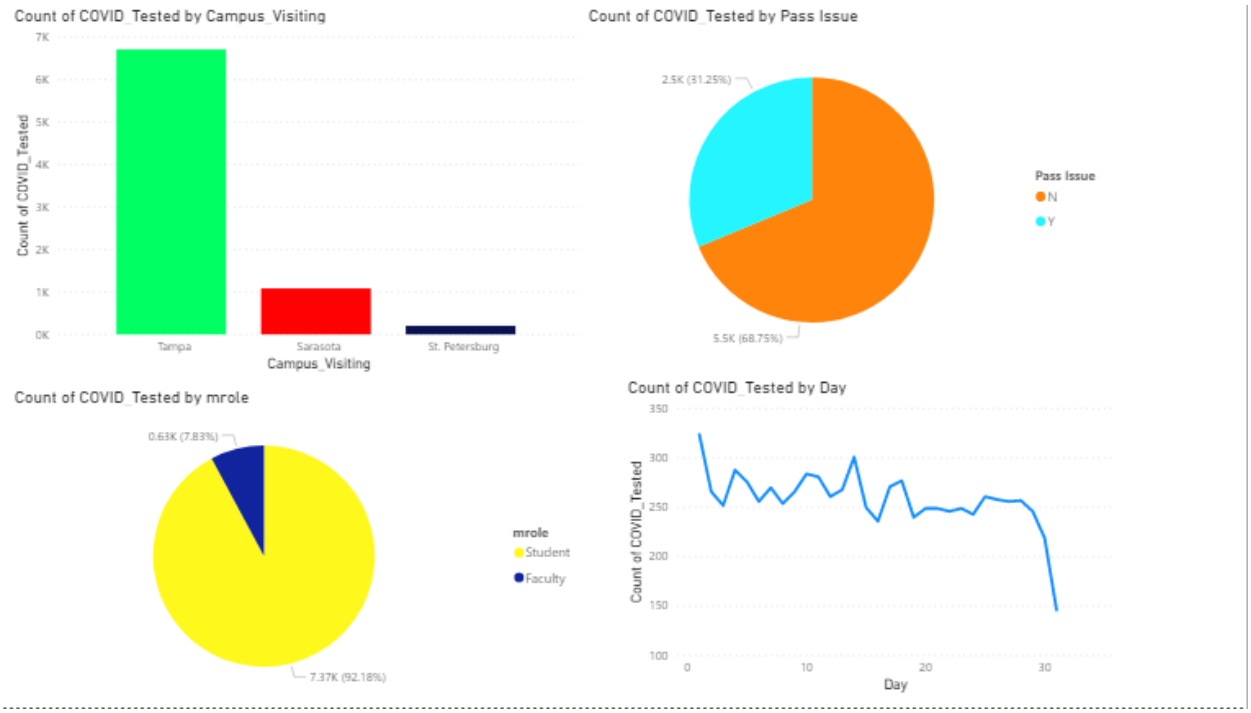
5.1. DBA scripts

- DBA Scripts have been added in section 2.5

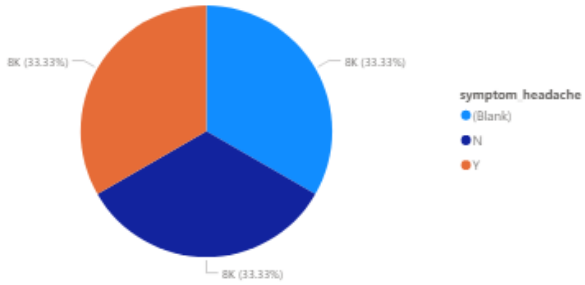
5.2. Data Visualization

As Data visualization tools such as Tableau, Microsoft Power BI help us visualize the data available in different graphical representations which can be used to answer key questions. We aim to create following visualizations for our COVID symptom database (will be added during the final submission):

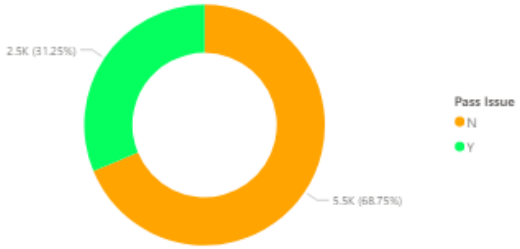
- A line chart with number of cases every day
- A bar graph of number of students who have had symptoms or came into contact with people with virus or number of people who are getting passes with respect to campuses
- Card visualizations to get the numerical data
- Bar graph to represent number of test by testing centers
- A visualization to plot the numbers against each symptom



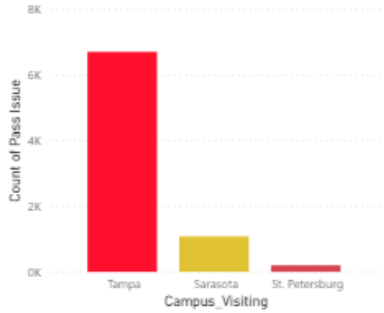
Count of COVID_Tested by symptom_headache



Count of Contact_With_Positive by Pass Issue



Count of Pass Issue by Campus_Visiting



Total number of test locations

187

Count of Test_Location