DBMS PROJECT PART 4

By Mounisha Konduru

1.Queries:

1.Create a query to show the department's and employee's IDs, respectively. Department alias name should be given as dpt Name.

Code:

initiate.py - C:/Users/Mounisha Konduru/Desktop/initiate.py (3.11.0)

File Edit Format Run Options Window Help

```
import psycopg2
import pandas as pd
import warnings
import os
os.system('CLS')
warnings.filterwarnings('ignore')
def initialize():
    connection = psycopg2.connect(
        user = "postgres", #username that you use
        password = "Mouni@99", #password that you use, you don't need to include your password when submiting your code
        host = "localhost",
        port = "5432",
        database = "testing"
    )
    connection.autocommit = True
    return connection

# If you need to add new tables to your database you can use the following function to create the target table
# assuming that conn is a valid, open connection to a Postgres database
def runQuery(conn):
    select_Query = "SELECT employee_id, department_id AS dpt_Name FROM employee_information  ;"
    editions_df = pd.DataFrame(columns = ['employee_id','department_id'])
    with conn.cursor() as cursor:
        cursor.execute(select_Query)
        records = cursor.fetchall()
        for row in records:
            output_df = {'employee_id': row[0], 'department-id': row[1]}
#             print("employee_id = ", row[0], )
#             print("department_id = ", row[1], )
            editions_df = editions_df.append(output_df, ignore_index=True)

        print(editions_df)

def main():
    conn = initialize()
    runQuery(conn)

if __name__ == "__main__":
    main()
```

Output:

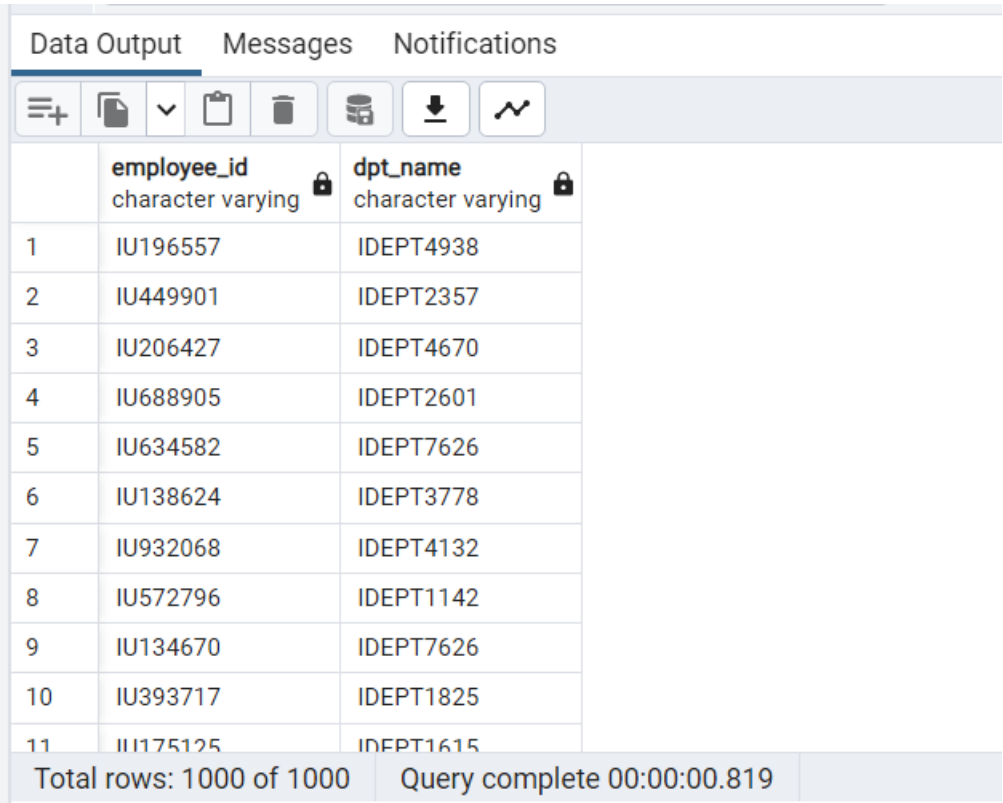
```
============ RESTART: C:/Users/Mounisha Konduru/Desktop/initiate.py ============
    employee_id department_id
0       IU196557      IDEPT4938
1       IU449901      IDEPT2357
2       IU206427      IDEPT4670
3       IU688905      IDEPT2601
4       IU634582      IDEPT7626
..           ...           ...
995     IU531414      IDEPT4055
996     IU227668      IDEPT8598
997     IU528779      IDEPT3062
998     IU874473      IDEPT7783
999     IU800953      IDEPT1677

[1000 rows x 2 columns]
```

Total no.of rows returned 1000

Query:

```
SELECT employee_id, department_id AS dpt_Name
FROM employee_information;
```

Output:



Data Output | Messages | Notifications

	employee_id character varying	dpt_name character varying
1	IU196557	IDEPT4938
2	IU449901	IDEPT2357
3	IU206427	IDEPT4670
4	IU688905	IDEPT2601
5	IU634582	IDEPT7626
6	IU138624	IDEPT3778
7	IU932068	IDEPT4132
8	IU572796	IDEPT1142
9	IU134670	IDEPT7626
10	IU393717	IDEPT1825
11	IU175125	IDEPT1615

Total rows: 1000 of 1000 | Query complete 00:00:00.819

Total no.of rows returned 1000

2. Create a query to show the number of articles published throughout the semester with the name "Sem 5". (count)

Code:

initiate.py - C:/Users/Mounisha Konduru/Desktop/initiate.py (3.11.0)

File Edit Format Run Options Window Help

```
import psycopg2
import pandas as pd
import warnings
import os
os.system('CLS')
warnings.filterwarnings('ignore')
def initialize():
    connection = psycopg2.connect(
        user = "postgres", #username that you use
        password = "Mouni@99", #password that you use, you don't need to include your password when submiting your code
        host = "localhost",
        port = "5432",
        database = "testing"
    )
    connection.autocommit = True
    return connection

# If you need to add new tables to your database you can use the following function to create the target table
# assuming that conn is a valid, open connection to a Postgres database
def runQuery(conn):
    select_Query = "SELECT COUNT(paper_id) FROM student_performance_data WHERE semster_name = 'Sem_5' ;"
    editions_df = pd.DataFrame(columns = ['count'])
    with conn.cursor() as cursor:
        cursor.execute(select_Query)
        records = cursor.fetchall()
        for row in records:
            output_df = {'count': row[0]}
#             print("paper_id = ", row[0], )
            editions_df = editions_df.append(output_df, ignore_index=True)

        print(editions_df)

def main():
    conn = initialize()
    runQuery(conn)

if __name__ == "__main__":
    main()
```

Output:

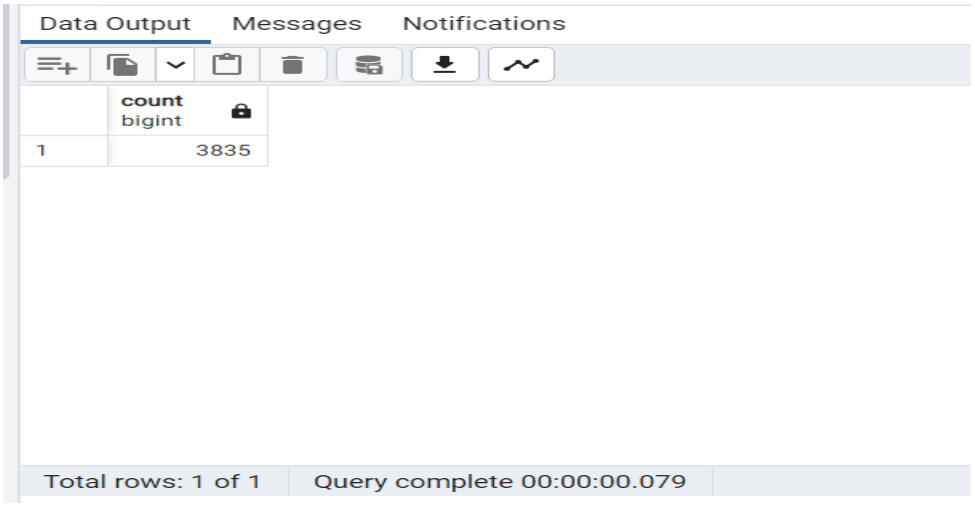
```
   count
0   3835
```

Total no.of rows returned 1

Query:

```
SELECT COUNT(paper_id)

FROM student_performance_data

WHERE semster_name = 'Sem_5';
```

Output:



Total No.of rows returned 1

3. Create a query to show the department name for students with grades of higher than 90 and more than three papers. (IN)

CODE:

```
File  Edit  Format  Run  Options  Window  Help
import psycopg2
import pandas as pd
import warnings
import os
os.system('CLS')
warnings.filterwarnings('ignore')
def initialize():
    connection = psycopg2.connect(
        user = "postgres", #username that you use
        password = "Mouni@99", #password that you use, you don't need to include your password when submiting your code
        host = "localhost",
        port = "5432",
        database = "testing"
    )
    connection.autocommit = True
    return connection

# If you need to add new tables to your database you can use the following function to create the target table
# assuming that conn is a valid, open connection to a Postgres database
def runQuery(conn):
    select_Query = "SELECT DISTINCT department_name FROM department_information WHERE department_id IN (SELECT DISTINCT D.department_id FROM department_performance D join student_performan
    editions_df = pd.DataFrame(columns = ['department-name'])
    with conn.cursor() as cursor:
        cursor.execute(select_Query)
        records = cursor.fetchall()
        for row in records:
            output_df = {'department-name': row[0]}
            editions_df = editions_df.append(output_df, ignore_index=True)

        print(editions_df)

def main():
    conn = initialize()
    runQuery(conn)

if __name__ == "__main__":
    main()
```

OUTPUT:

```
========== RESTART: C:\Users\Mounisha Konduru\Desktop\Python dbms\query 3.py ==
                                    department-name
0                            ABC-EDS Research Academy
1                               Aerospace Engineering
2                    Application Software Centre (ASC)
3                       Biosciences and Bioengineering
4            Center for Learning and Teaching (PPCCLT)
5   Centre for Aerospace Systems Design and Engine...
6                    Centre for Bioengineering (WRCB)
7   Centre for Distance Engineering Education Prog...
8                  Centre for Entrepreneurship (DSCE)
9   Centre for Environmental Science and Engineeri...
10  Centre for Formal Design and Verification of S...
11                   Centre for Policy Studies (CPS)
12  Centre for Research in Nanotechnology and Scie...
13  Centre for Technology Alternatives for Rural A...
14   Centre for Urban Science and Engineering (C-USE)
15  Centre of Studies in Resources Engineering (CSRE)
16                             Chemical Engineering
17                                        Chemistry
18                                Civil Engineering
19                                  Climate Studies
20                             Computer Centre (CC)
21                   Computer Science & Engineering
22                                  Earth Sciences
23                          Educational Technology
24                          Electrical Engineering
25                   Energy Science and Engineering
26                      Humanities & Social Science
27                         Industrial Design Centre
28  Industrial Engineering and Operations Research...
29                                       Mathematics
30                            Mechanical Engineering
31      Metallurgical Engineering & Materials Science
32  National Center of Excellence in Technology fo...
33  National Centre for Aerospace Innovation and R...
34              National Centre for Mathematics (NCM)
35                                           Physics
36                            School of Management
37  Sophisticated Analytical Instrument Facility (...
38                   Systems and Control Engineering
39                      Technology and Design (TCTD)
```

Total no of rows returned 40

QUERY:

```
SELECT DISTINCT department_name
FROM department_information
WHERE department_id IN (SELECT DISTINCT D.department_id
FROM department_performance D
join student_performance_data S ON D.student_id=S.student_id
WHERE S.student_id IN (SELECT student_id
FROM student_performance_data group by student_id, marks
HAVING COUNT(paper_id) > 3 and marks > 90));
```

Output:

Query | Query History | Scratch Pad

```
1  SELECT DISTINCT department_name
2  FROM department_information
3  WHERE department_id IN (SELECT DISTINCT D.department_id
4  FROM department_performance D
5  join student_performance_data S ON D.student_id=S.student_id
6  WHERE S.student_id IN (SELECT student_id
7  FROM student_performance_data group by student_id, marks
8  HAVING COUNT(paper_id) > 3 and marks > 90));
9
```

Data Output | Messages | Notifications

	department_name character varying
1	ABC-EDS Resear...
2	Aerospace Engin...
3	Application Soft...
4	Biosciences and ...
5	Center for Learni...
6	Centre for Aeros...
7	Centre for Bioeng...
8	Centre for Distan...
9	Centre for Entrep...
10	Centre for Enviro...
11	Centre for Forma...

Total rows: 40 of 40 | Query complete 00:00:01.974

Total no.of rows returned 40

4. Create a query that shows the student IDs of those students with more than five papers and a grade of less than 84. (Group by)

Code:

```
import psycopg2
import pandas as pd
import warnings
import os
os.system('CLS')
warnings.filterwarnings('ignore')
def initialize():
    connection = psycopg2.connect(
        user = "postgres", #username that you use
        password = "Mouni@99", #password that you use, you don't need to include your password when submiting your code
        host = "localhost",
        port = "5432",
        database = "testing"
    )
    connection.autocommit = True
    return connection

# If you need to add new tables to your database you can use the following function to create the target table
# assuming that conn is a valid, open connection to a Postgres database
def runQuery(conn):
    select_Query = "SELECT student_id FROM student_performance_data group by student_id, marks HAVING COUNT(paper_id) > 5 and marks < 84 ;"
    editions_df = pd.DataFrame(columns = ['student_id'])
    with conn.cursor() as cursor:
        cursor.execute(select_Query)
        records = cursor.fetchall()
        for row in records:
            output_df = {'student_id': row[0]}
#             print("student_id = ", row[0], )
            editions_df = editions_df.append(output_df, ignore_index=True)

        print(editions_df)

def main():
    conn = initialize()
    runQuery(conn)

if __name__ == "__main__":
    main()
```

Output:

```
============================================================ RESTART: C:/Users/Mounisha Konduru/Desktop/Python dbms/query 4.py ==
    student_id
0  SID20133854
1  SID20136876
2  SID20134240
```

Total no. of rows returned 3

Query:

```
SELECT student_id

FROM student_performance_data

 group by student_id, marks

HAVING COUNT (paper_id) > 5 and marks < 84 ;
```

OUTPUT:

Query | Query History | Scratch Pad

```
1  SELECT student_id
2  FROM student_performance_data
3  group by student_id, marks
4  HAVING COUNT(paper_id) > 5 and marks < 84 ;
5
6
```

Data Output | Messages | Notifications

	student_id character varying
1	SID20133854
2	SID20136876
3	SID20134240

Total no.of rows returned 3

5. Create a query that shows the employee id and department id for employees who work in the oldest department. (Join)

CODE:

```
import psycopg2
import pandas as pd
import warnings
import os
os.system('CLS')
warnings.filterwarnings('ignore')
def initialize():
    connection = psycopg2.connect(
        user = "postgres", #username that you use
        password = "Mouni@99", #password that you use, you don't need to include your password when submiting your code
        host = "localhost",
        port = "5432",
        database = "testing"
    )
    connection.autocommit = True
    return connection

# If you need to add new tables to your database you can use the following function to create the target table
# assuming that conn is a valid, open connection to a Postgres database
def runQuery(conn):
    select_Query = "SELECT E.employee_id, E.department_id FROM employee_information E JOIN department_information D ON E.department_id=D.department_id Where D.doe=(select min(doe)FROM depar
    editions_df = pd.DataFrame(columns = ['employee_id','department_id'])
    with conn.cursor() as cursor:
        cursor.execute(select_Query)
        records = cursor.fetchall()
        for row in records:
            output_df = {'employee_id': row[0],'department_id': row[1]}
            editions_df = editions_df.append(output_df, ignore_index=True)

        print(editions_df)

def main():
    conn = initialize()
    runQuery(conn)

if __name__ == "__main__":
    main()
```

Output:

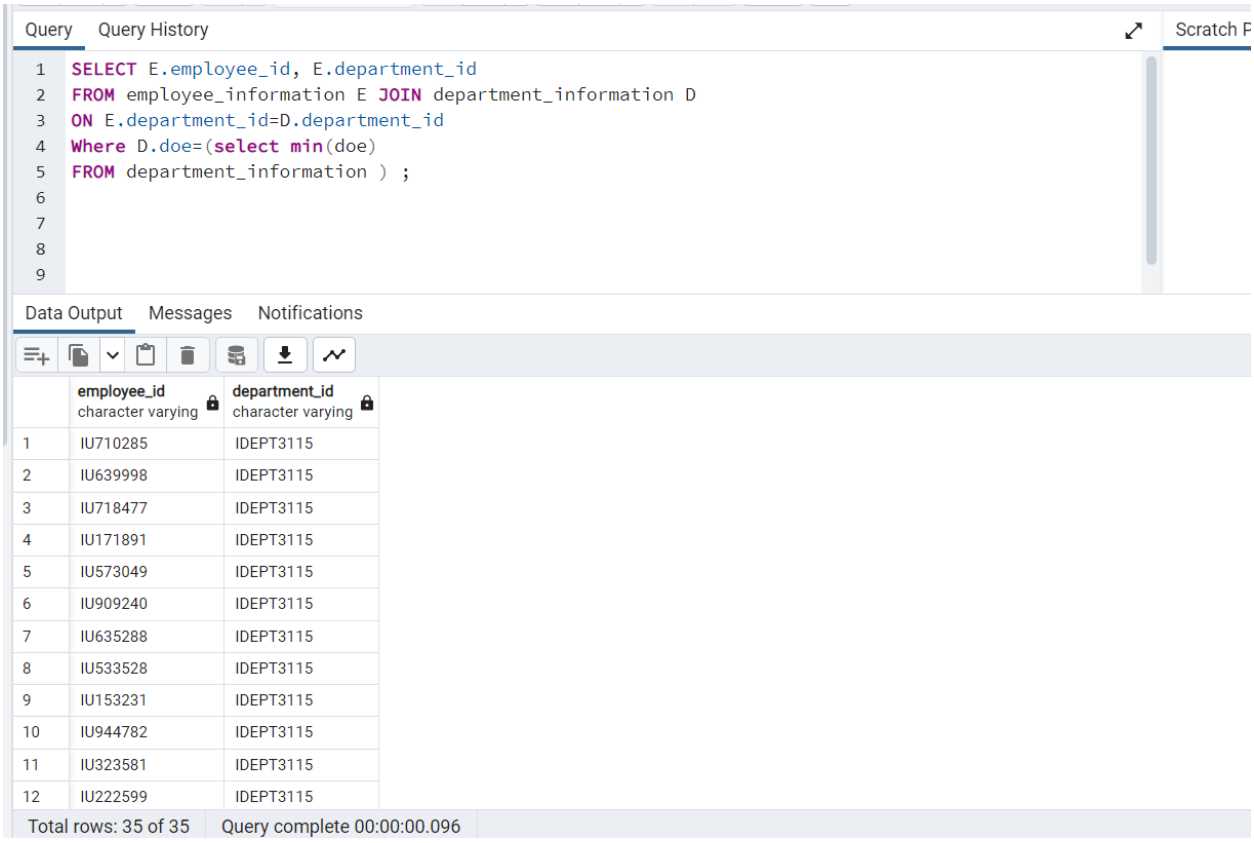
```
======= RESTART: C:\Users\Mounisha Konduru\Desktop\Python dbms\query 5.py ======
   employee_id department_id
0      IU710285     IDEPT3115
1      IU639998     IDEPT3115
2      IU718477     IDEPT3115
3      IU171891     IDEPT3115
4      IU573049     IDEPT3115
5      IU909240     IDEPT3115
6      IU635288     IDEPT3115
7      IU533528     IDEPT3115
8      IU153231     IDEPT3115
9      IU944782     IDEPT3115
10     IU323581     IDEPT3115
11     IU222599     IDEPT3115
12     IU906894     IDEPT3115
13     IU345987     IDEPT3115
14     IU515736     IDEPT3115
15     IU873202     IDEPT3115
16     IU388674     IDEPT3115
17     IU243018     IDEPT3115
18     IU210386     IDEPT3115
19     IU258875     IDEPT3115
20     IU272917     IDEPT3115
21     IU268125     IDEPT3115
22     IU342000     IDEPT3115
23     IU637845     IDEPT3115
24     IU325646     IDEPT3115
25     IU477466     IDEPT3115
26     IU708882     IDEPT3115
27     IU344368     IDEPT3115
28     IU274936     IDEPT3115
29     IU222618     IDEPT3115
30     IU991869     IDEPT3115
31     IU662779     IDEPT3115
32     IU964095     IDEPT3115
33     IU875236     IDEPT3115
34     IU824346     IDEPT3115
```

Total no.of rows returned 35

Query:

```
SELECT E.employee_id, E.department_id

FROM employee_information E JOIN department_information D

ON E.department_id=D.department_id

Where D.doe=(select min(doe)

FROM department_information ) ;
```

OUTPUT:



```
Query  Query History                                                    Scratch P
1  SELECT E.employee_id, E.department_id
2  FROM employee_information E JOIN department_information D
3  ON E.department_id=D.department_id
4  Where D.doe=(select min(doe)
5  FROM department_information ) ;
6
7
8
9
```

Data Output | Messages | Notifications

	employee_id character varying	department_id character varying
1	IU710285	IDEPT3115
2	IU639998	IDEPT3115
3	IU718477	IDEPT3115
4	IU171891	IDEPT3115
5	IU573049	IDEPT3115
6	IU909240	IDEPT3115
7	IU635288	IDEPT3115
8	IU533528	IDEPT3115
9	IU153231	IDEPT3115
10	IU944782	IDEPT3115
11	IU323581	IDEPT3115
12	IU222599	IDEPT3115

Total rows: 35 of 35 Query complete 00:00:00.096

Total no.of rows returned 35

6. Create a query to show student ids for students who are in the more established, Latest department

Code:

```
import psycopg2
import pandas as pd
import warnings
import os
os.system('CLS')
warnings.filterwarnings('ignore')
def initialize():
    connection = psycopg2.connect(
        user = "postgres", #username that you use
        password = "Mouni@99", #password that you use, you don't need to include your password when submiting your code
        host = "localhost",
        port = "5432",
        database = "testing"
    )
    connection.autocommit = True
    return connection

# If you need to add new tables to your database you can use the following function to create the target table
# assuming that conn is a valid, open connection to a Postgres database
def runQuery(conn):
    select_Query = "SELECT S.student_id FROM student_counseling_information S JOIN department_information D ON S.department_admission=D.department_id Where D.doe=( SELECT max(doe)FROM depa
    editions_df = pd.DataFrame(columns = ['student_id'])
    with conn.cursor() as cursor:
        cursor.execute(select_Query)
        records = cursor.fetchall()
        for row in records:
            output_df = {'student_id': row[0]}
#             print("student_id = ", row[0] )
            editions_df = editions_df.append(output_df, ignore_index=True)

        print(editions_df)

def main():
    conn = initialize()
    runQuery(conn)

if __name__ == "__main__":
    main()
```

OUTPUT:

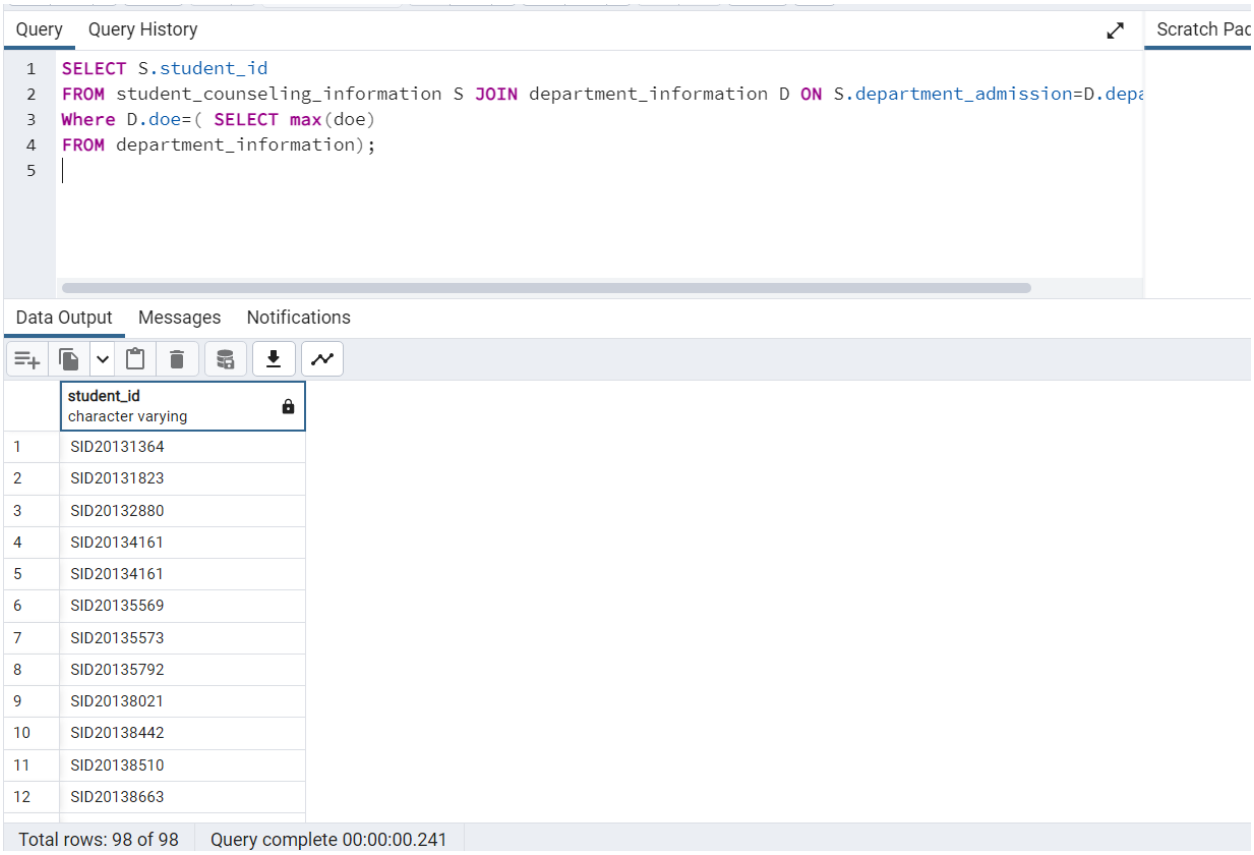
```
========== RESTART: C:\Users\Mounisha Konduru\Desktop\Python dbms
      student_id
0    SID20131364
1    SID20131823
2    SID20132880
3    SID20134161
4    SID20134161
..           ...
93   SID20188319
94   SID20188865
95   SID20189015
96   SID20189443
97   SID20189510

[98 rows x 1 columns]
```

Total no.of rows returned 98.

QUERY:

SELECT S.student_id
FROM student_counseling_information S JOIN department_information D ON S.department_admission=D.department_id
Where D.doe=(SELECT max(doe)
FROM department_information);

OUTPUT:



Total no.of rows returned 98

7. Create a query to show the department name for students who are older than 25. (Join)

Code:

```
import psycopg2
import pandas as pd
import warnings
import os
os.system('CLS')
warnings.filterwarnings('ignore')
def initialize():
    connection = psycopg2.connect(
        user = "postgres", #username that you use
        password = "Mouni@99", #password that you use, you don't need to include your password when submiting your code
        host = "localhost",
        port = "5432",
        database = "testing"
    )
    connection.autocommit = True
    return connection

# If you need to add new tables to your database you can use the following function to create the target table
# assuming that conn is a valid, open connection to a Postgres database
def runQuery(conn):
    select_Query = "SELECT D.department_name FROM student_counseling_information S JOIN department_information D ON S.department_admission=D.department_id Where S.dob<'1996.01.01';"
    editions_df = pd.DataFrame(columns = ['department_name'])
    with conn.cursor() as cursor:
        cursor.execute(select_Query)
        records = cursor.fetchall()
        for row in records:
            output_df = {'department_name': row[0]}
#             print("department_name = ", row[0], )
            editions_df = editions_df.append(output_df, ignore_index=True)

        print(editions_df)

def main():
    conn = initialize()
    runQuery(conn)

if __name__ == "__main__":
    main()
```

Output:

```
========== RESTART: C:/Users/Mounisha Konduru/Desktop/Python dbm
                                    department_name
0    Sophisticated Analytical Instrument Facility (...
1                            Educational Technology
2                      Biosciences and Bioengineering
3    National Center of Excellence in Technology fo...
4                                 Chemical Engineering
..                                                 ...
333              National Centre for Mathematics (NCM)
334                  Application Software Centre (ASC)
335          Center for Learning and Teaching (PPCCLT)
336                           ABC-EDS Research Academy
337                                       Mathematics

[338 rows x 1 columns]
```

Total no.of rows returned 338

Query:

SELECT D.department_name

FROM student_counseling_information S

JOIN department_information D ON S.department_admission=D.department_id

Where S.dob<'1996.01.01';

Output:

Query | Query History | Execute/Refresh F5 | Scratch Pad

```
1  SELECT D.department_name
2  FROM student_counseling_information S JOIN department_information D
3  ON S.department_admission=D.department_id
4  Where S.dob<'1996.01.01' ;
```

Data Output | Messages | Notifications

	department_name character varying
1	Sophisticated An...
2	Educational Tech...
3	Biosciences and ...
4	National Center o...
5	Chemical Engine...
6	Centre for Entrep...
7	Systems and Con...
8	Sophisticated An...
9	Application Soft...
10	Electrical Engine...
11	Chemical Engine...

Total rows: 338 of 338 | Query complete 00:00:00.090

Total no. of rows returned 338

8. Create a query to see how many students share the same grades.

Code:

```
import psycopg2
import pandas as pd
import warnings
import os
os.system('CLS')
warnings.filterwarnings('ignore')
def initialize():
    connection = psycopg2.connect(
        user = "postgres", #username that you use
        password = "Mouni@99", #password that you use, you don't need to include your password when submiting your code
        host = "localhost",
        port = "5432",
        database = "testing"
    )
    connection.autocommit = True
    return connection

# If you need to add new tables to your database you can use the following function to create the target table
# assuming that conn is a valid, open connection to a Postgres database
def runQuery(conn):
    select_Query = "SELECT marks, COUNT(STUDENT_ID) FROM student_performance_data GROUP BY marks;"
    editions_df = pd.DataFrame(columns = ['marks','count'])
    with conn.cursor() as cursor:
        cursor.execute(select_Query)
        records = cursor.fetchall()
        for row in records:
            output_df = {'marks': row[0], 'count': row[1] }
#             print("marks = ", row[0], )
#             print("count = ", row[1], )
            editions_df = editions_df.append(output_df, ignore_index=True)

        print(editions_df)

def main():
    conn = initialize()
    runQuery(conn)

if __name__ == "__main__":
    main()
```

Output:

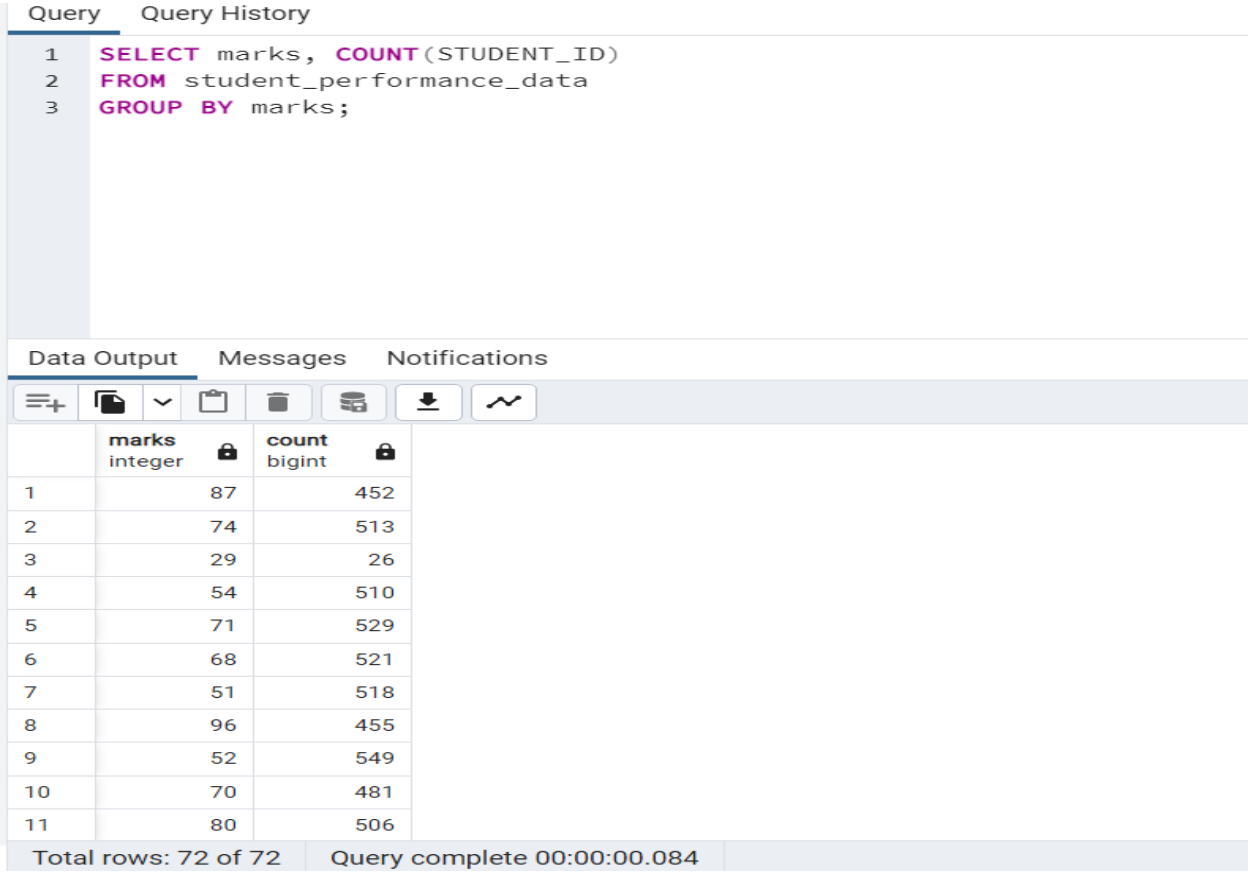
```
=========== RESTART: C:\Users\Mounisha Konduru\Desktc
   marks count
0     87   452
1     74   513
2     29    26
3     54   510
4     71   529
..   ...   ...
67    27    33
68    23    27
69    56   523
70    58   473
71    91   590

[72 rows x 2 columns]
```

Total no.of rows returned 72

Query:

```
SELECT marks, COUNT(STUDENT_ID)
FROM student_performance_data
GROUP BY marks;
```

Output:



Total no.of rows returned 72

9. Create a query to show the number of papers published in the department, along with the department's id, dep id. (Join)

Code:

```
File  Edit  Format  Run  Options  Window  Help
import psycopg2
import pandas as pd
import warnings
import os
os.system('CLS')
warnings.filterwarnings('ignore')
def initialize():
    connection = psycopg2.connect(
        user = "postgres", #username that you use
        password = "Mouni@99", #password that you use, you don't need to include your password when submiting your code
        host = "localhost",
        port = "5432",
        database = "testing"
    )
    connection.autocommit = True
    return connection

# If you need to add new tables to your database you can use the following function to create the target table
# assuming that conn is a valid, open connection to a Postgres database
def runQuery(conn):
    select_Query = "SELECT DISTINCT SC.department_choices AS dep_id, COUNT(paper_id)FROM student_performance_data SP JOIN student_counseling_information SC on SC.student_id= SP.student_id
    editions_df = pd.DataFrame(columns = ['dep_id','count'])
    with conn.cursor() as cursor:
        cursor.execute(select_Query)
        records = cursor.fetchall()
        for row in records:
            output_df = {'dep_id': row[0], 'count': row[1]}
#             print("dep_id = ", row[0], )
#             print("count = ", row[1], )
            editions_df = editions_df.append(output_df, ignore_index=True)

        print(editions_df)

def main():
    conn = initialize()
    runQuery(conn)

if __name__ == "__main__":
    main()
```

Output:

```
========== RESTART: C:\Users\Mounisha Konduru\Desktop\Python dbms\query 9.py ===
        dep_id count
0    IDEPT1096   603
1    IDEPT1142   664
2    IDEPT1367   443
3    IDEPT1378   935
4    IDEPT1388   493
5    IDEPT1423   656
6    IDEPT1533   873
7    IDEPT1537   771
8    IDEPT1615   712
9    IDEPT1677  1099
10   IDEPT1825   714
11   IDEPT1836   878
12   IDEPT2054   771
13   IDEPT2357   657
14   IDEPT2425   771
15   IDEPT2601  1151
16   IDEPT3062   766
17   IDEPT3115  1271
18   IDEPT3778   935
19   IDEPT3868  1046
20   IDEPT4055   711
21   IDEPT4132   714
22   IDEPT4308   720
23   IDEPT4670  1041
24   IDEPT4938   660
25   IDEPT5109   549
26   IDEPT5127   933
27   IDEPT5408   332
28   IDEPT5528  1044
29   IDEPT5564   807
30   IDEPT5881   663
31   IDEPT6347  1037
32   IDEPT7005   549
33   IDEPT7626   662
34   IDEPT7783   985
35   IDEPT8313   872
36   IDEPT8379   927
37   IDEPT8473  1379
38   IDEPT8598   822
39   IDEPT8825   599
```

Total no.of rows returned 40

Query:

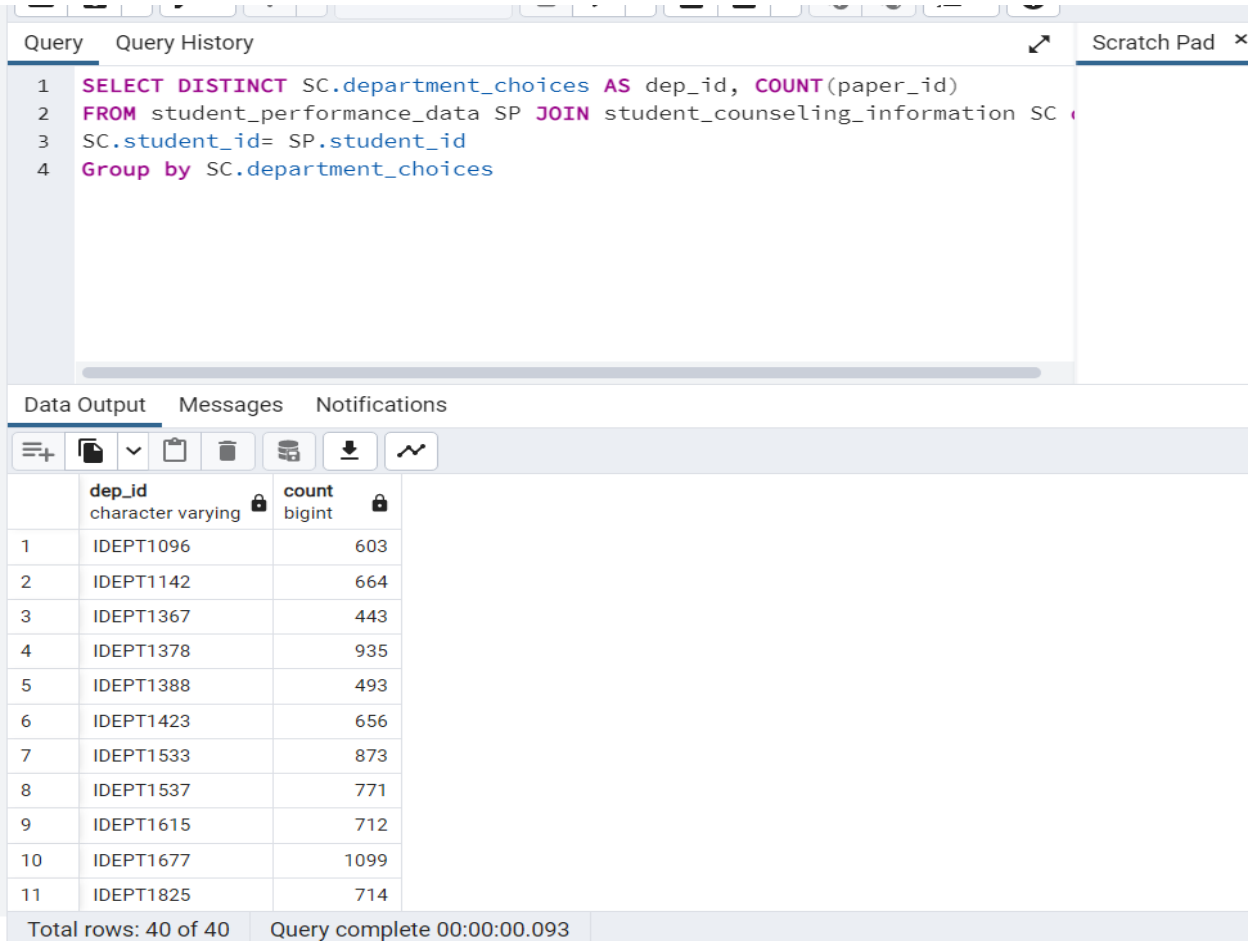
SELECT DISTINCT SC.department_choices AS dep_id, COUNT(paper_id)

FROM student_performance_data SP JOIN student_counseling_information SC on

SC.student_id= SP.student_id

Group by SC.department_choices;

Output:



Total no.of rows returned 40

10. Create a query to show the student IDs from the oldest department's admissions. (Subquery,MIN)

Code:

```
import psycopg2
import pandas as pd
import warnings
import os
os.system('CLS')
warnings.filterwarnings('ignore')
def initialize():
    connection = psycopg2.connect(
        user = "postgres", #username that you use
        password = "Mouni@99", #password that you use, you don't need to include your password when submiting your code
        host = "localhost",
        port = "5432",
        database = "testing"
    )
    connection.autocommit = True
    return connection

# If you need to add new tables to your database you can use the following function to create the target table
# assuming that conn is a valid, open connection to a Postgres database
def runQuery(conn):
    select_Query = "SELECT student_id FROM student_counseling_information WHERE department_admission= (SELECT department_id FROM department_information WHERE doe= (SELECT MIN(doe) FROM depa
    editions_df = pd.DataFrame(columns = ['student_id'])
    with conn.cursor() as cursor:
        cursor.execute(select_Query)
        records = cursor.fetchall()
        for row in records:
            output_df = {'student_id': row[0]}
#             print("student_id = ", row[0], )
            editions_df = editions_df.append(output_df, ignore_index=True)

        print(editions_df)

def main():
    conn = initialize()
    runQuery(conn)

if __name__ == "__main__":
    main()
```

Output:

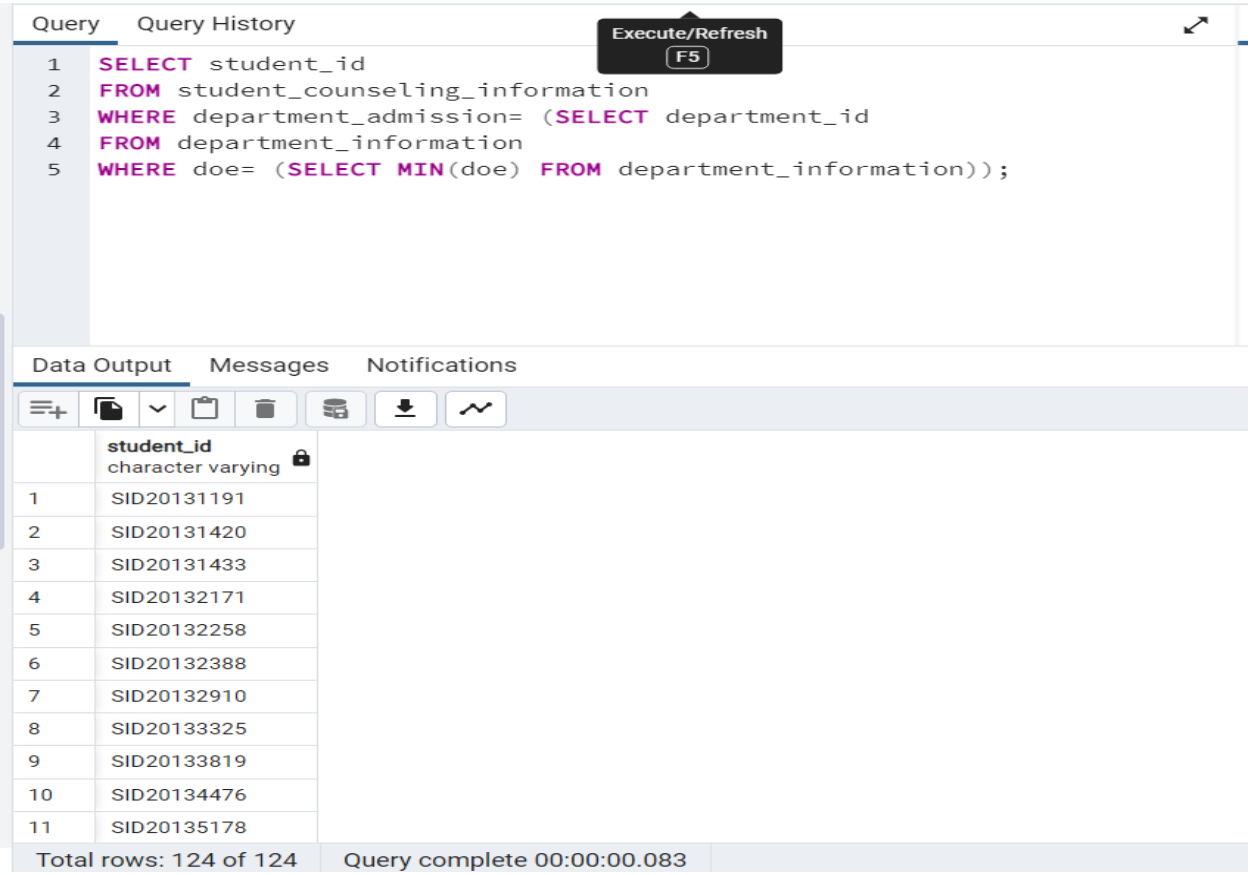
```
========== RESTART: C:/Users/Mounisha Konduru/
      student_id
0    SID20131191
1    SID20131420
2    SID20131433
3    SID20132171
4    SID20132258
..           ...
119  SID20187063
120  SID20187169
121  SID20188203
122  SID20188474
123  SID20189892

[124 rows x 1 columns]
```

Total no.of rows returned 124

Query:

```
SELECT student_id
FROM student_counseling_information
WHERE department_admission= (SELECT department_id
FROM department_information
WHERE doe= (SELECT MIN(doe) FROM department_information)
```

Output:



Total no.of rows returned 124

2. Query performance:

Query 5:

```
EXPLAIN
SELECT E.employee_id, E.department_id
FROM employee_information E JOIN department_information D
ON E.department_id=D.department_id
Where D.doe=(select min(doe)
FROM department_information ) ;
```

Data Output | Messages | Notifications

	QUERY PLAN text
1	Hash Join (cost=3.02..23.93 rows=25 width=19)
2	Hash Cond: ((e.department_id)::text = (d.department_id)::text)
3	InitPlan 1 (returns $0)
4	-> Aggregate (cost=1.50..1.51 rows=1 width=4)
5	-> Seq Scan on department_information (cost=0.00..1.40 rows=40 width=4)
6	-> Seq Scan on employee_information e (cost=0.00..18.00 rows=1000 width=19)
7	-> Hash (cost=1.50..1.50 rows=1 width=32)
8	-> Seq Scan on department_information d (cost=0.00..1.50 rows=1 width=32)
9	Filter: (doe = $0)

Query plan with analyse:

```
EXPLAIN Analyse
SELECT E.employee_id, E.department_id
FROM employee_information E JOIN department_information D
ON E.department_id=D.department_id
Where D.doe=(select min(doe)
FROM department_information ) ;
```

	QUERY PLAN text
1	Hash Join (cost=3.02..23.93 rows=25 width=19) (actual time=0.340..0.576 rows=35 loops=1)
2	Hash Cond: ((e.department_id)::text = (d.department_id)::text)
3	InitPlan 1 (returns $0)
4	-> Aggregate (cost=1.50..1.51 rows=1 width=4) (actual time=0.022..0.023 rows=1 loops=1)
5	-> Seq Scan on department_information (cost=0.00..1.40 rows=40 width=4) (actual time=0.010..0.013 rows=40 loops=1)
6	-> Seq Scan on employee_information e (cost=0.00..18.00 rows=1000 width=19) (actual time=0.087..0.175 rows=1000 loo…
7	-> Hash (cost=1.50..1.50 rows=1 width=32) (actual time=0.153..0.154 rows=1 loops=1)
8	Buckets: 1024 Batches: 1 Memory Usage: 9kB
9	-> Seq Scan on department_information d (cost=0.00..1.50 rows=1 width=32) (actual time=0.104..0.108 rows=1 loops=1)
10	Filter: (doe = $0)
11	Rows Removed by Filter: 39
12	Planning Time: 4.165 ms
13	Execution Time: 1.354 ms

Join Algorithm Used: Hash Join

The estimated cost to run the Query: 23.93

Actual time to run the query: 0.576

Reason:

The query that I had chosen used the Hash Join the reason behind this below**,**

Query:

show work_mem

	work_mem text
1	4MB

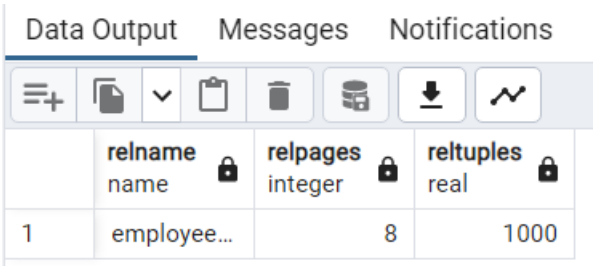
The maximum amount of memory that will be allocated to any query is 4MB

Size of each page is = 8KB

Number of Buffer Pages (BP) = 4MB/8KB = 512 buffer pages

Query:

```
select relname,relpages,reltuples

from pg_class

where relname=' employee_information';
```



Here no of pages in outer relation (M) = 8

The outer relation of the Join is mentioned in the query can thus fit entirely in the buffer memory. Therefore, the database will use the Hash Join in this instance. In this case, the outer relation employee information will be covered by a hash table, and each hash value will have its corresponding tuples associated with it depending on the hash function on the join property. The join property of the inner relation will later be hashed using the same function, and we will search the outer relation's hash table for matches.

Improve the Performance of the Query:

So, if we wanted to enhance the query's performance in this case, we could build a clustered index on the department information's doe field. The data will be in sorter order over the dob with indexes pointing to them after the clustered index on the doe property is created. In order to match the predicate condition of s.doe='min(doe)', the database can use the index scan rather than the sequential scan.

Making an index for other attributes won't help the query run better because they won't speed up the data-scanning process used to filter the doe attribute condition. The index won't match the predicate in that situation.

TO create the Index:

Query:

CREATE INDEX doe_idx ON department_information (doe);

Data Output | Messages | Notifications

```
CREATE INDEX

Query returned successfully in 80 msec.
```

After Creating Index:

Query:

EXPLAIN

SELECT E.employee_id, E.department_id

FROM employee_information E JOIN department_information D

ON E.department_id=D.department_id

Where D.doe=(select min(doe)

FROM department_information)

Data Output | Messages | Notifications

	QUERY PLAN text
1	Hash Join (cost=1.98..22.88 rows=25 width=19)
2	Hash Cond: ((e.department_id)::text = (d.department_id)::text)
3	InitPlan 2 (returns $1)
4	-> Result (cost=0.46..0.47 rows=1 width=4)
5	InitPlan 1 (returns $0)
6	-> Limit (cost=0.14..0.46 rows=1 width=4)
7	-> Index Only Scan using doe_idx on department_information (cost=0.14..12.84 rows=40 width=4)
8	Index Cond: (doe IS NOT NULL)
9	-> Seq Scan on employee_information e (cost=0.00..18.00 rows=1000 width=19)
10	-> Hash (cost=1.50..1.50 rows=1 width=32)
11	-> Seq Scan on department_information d (cost=0.00..1.50 rows=1 width=32)
12	Filter: (doe = $1)

Result:

We can see that there has been a change in the query plan of the query after the index on the department information's doe field was created, using an index scan rather than a sequential scan as it had previously done. As a result, the query's performance improves and runs more efficiently, resulting in a considerable decrease in the anticipated cost to conduct the query.

Before Index,

The estimated cost to run the Query: 23.93.

After Index on dob attribute,

The estimated cost to run the Query: 22.88.

Query 7:

Query Performance:

Query:

```
EXPLAIN
SELECT D.department_name
FROM student_counseling_information S JOIN department_information D
ON S.department_admission=D.department_id
Where S.dob<'1996.01.01' ;
```

	QUERY PLAN text
1	Hash Join (cost=1.90..86.81 rows=331 width=32)
2	Hash Cond: ((s.department_admission)::text = (d.department_id)::text)
3	-> Seq Scan on student_counseling_information s (cost=0.00..83.95 rows=331 width=10)
4	Filter: (dob < '1996-01-01'::date)
5	-> Hash (cost=1.40..1.40 rows=40 width=64)
6	-> Seq Scan on department_information d (cost=0.00..1.40 rows=40 width=64)

Query plan with analyse:

Query:

EXPLAIN Analyse

SELECT D.department_name

FROM student_counseling_information S JOIN department_information D

ON S.department_admission=D.department_id

Where S.dob<'1996.01.01';

	QUERY PLAN text
1	Hash Join (cost=1.90..86.81 rows=331 width=32) (actual time=0.579..2.212 rows=338 ...
2	Hash Cond: ((s.department_admission)::text = (d.department_id)::text)
3	-> Seq Scan on student_counseling_information s (cost=0.00..83.95 rows=331 width=1...
4	Filter: (dob < '1996-01-01'::date)
5	Rows Removed by Filter: 3658
6	-> Hash (cost=1.40..1.40 rows=40 width=64) (actual time=0.540..0.542 rows=40 loops...
7	Buckets: 1024 Batches: 1 Memory Usage: 12kB
8	-> Seq Scan on department_information d (cost=0.00..1.40 rows=40 width=64) (actual t...
9	Planning Time: 0.205 ms
10	Execution Time: 2.260 ms

Join Algorithm Used: Hash Join

The estimated cost to run the Query: 86.81

Actual time to run the query : 2.212

Reason:

The query that I had chosen used the Hash Join the reason behind this below,

Query:

show work_mem

	work_mem text
1	4MB

The maximum amount of memory that will be allocated to any query is 4MB

Size of each page is = 8KB

Number of Buffer Pages (BP) = 4MB/8KB = 512 buffer pages

QUERY:

select relname,relpages,reltuples

from pg_class

where relname='student_counseling_information'

	relname name	relpages integer	reltuples real
1	student_counseling_information	34	3996

Here no of pages in outer relation (M) = 34

The outer relation of the Join in this query can indeed fit entirely in the buffer memory. Therefore, the database will use the Hash Join in this example. Here, a hash table will be constructed over the outer relation student counseling information, with each hash value having its own associated tuples depending on the join attribute's hash function. The join property of the inner relation will later be hashed using the same function, and we will search the outer relation's hash table for matches.

Improve the Performance of the Query:

In this case, we may therefore establish a clustered index on the dob attribute of the student counseling information to enhance the efficiency of the query. The data will be placed in sorted order over the dob with indexes referring to them after the clustered index on the dob attribute is created. Therefore, the database

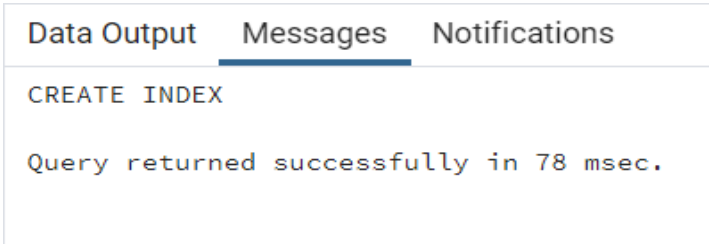
may match the predicate condition of s.dob's "1996.01.01" by using an index scan rather than a sequential scan.

The performance of the query will not be improved by creating indexes on other attributes since they will not speed up the data-scanning process used to filter the dob attribute condition. In that situation, the index won't match the predicate.

TO create the Index:

Query:

CREATE INDEX dob_idx ON student_counseling_information (dob);



After Creating Index:

Query:

EXPLAIN

SELECT D.department_name

FROM student_counseling_information S JOIN department_information D

ON S.department_admission=D.department_id

Where S.dob<'1996.01.01' ;

	QUERY PLAN text
1	Hash Join (cost=15.23..38.67 rows=331 width=218)
2	Hash Cond: ((s.department_admission)::text = (d.department_id)::text)
3	-> Index Scan using dob_idx on student_counseling_information s (cost=0.28..22.83 rows=331 width=10)
4	Index Cond: (dob < '1996-01-01'::date)
5	-> Hash (cost=12.20..12.20 rows=220 width=336)
6	-> Seq Scan on department_information d (cost=0.00..12.20 rows=220 width=336)

Result:

We can see that the query plan has changed once the index was built, with the database now doing an index scan on the student counseling information rather than a sequential scan as it did in the situation before the index was built. This results in the query's performance being enhanced, which lowers the anticipated cost of running the query.

Before Index,

The estimated cost to run the Query: 86.81.

After Index on dob attribute,

The estimated cost to run the Query: 38.67.

3. QUERY PLAN:

Query 5:

Query:

```
SELECT E.employee_id, E.department_id
FROM employee_information E JOIN department_information D
ON E.department_id=D.department_id
Where D.doe>'1991.01.01';
```

Query:

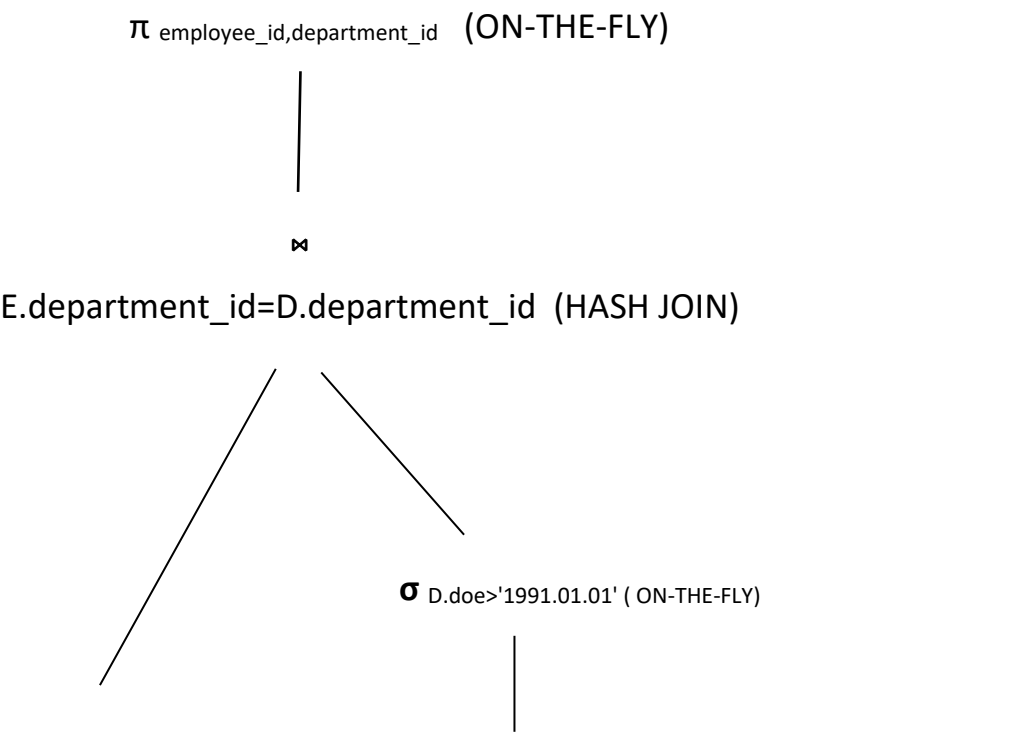
```
Explain
SELECT E.employee_id, E.department_id
FROM employee_information E JOIN department_information D
ON E.department_id=D.department_id
Where D.doe>'1991.01.01';
```

Output:

Data Output | Messages | Notifications

	QUERY PLAN text
1	Hash Join (cost=1.66..22.57 rows=325 width=19)
2	Hash Cond: ((e.department_id)::text = (d.department_id)::text)
3	-> Seq Scan on employee_information e (cost=0.00..18.00 rows=1000 width=19)
4	-> Hash (cost=1.50..1.50 rows=13 width=32)
5	-> Seq Scan on department_information d (cost=0.00..1.50 rows=13 width=32)
6	Filter: (doe > '1991-01-01'::date)

Physical query plan:



The join algorithm chosen by postgres is Hash Join.

Reason:

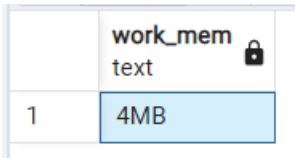
The query that I had chosen used Hash Join

The Hash Join is on the tables Employee_information and Department_information

Here outer relation is Employee_information and the Inner Relation is Department_information

Query:

show work_mem



	work_mem text
1	4MB

The maximum amount of memory that will be allocated to any query is 4MB

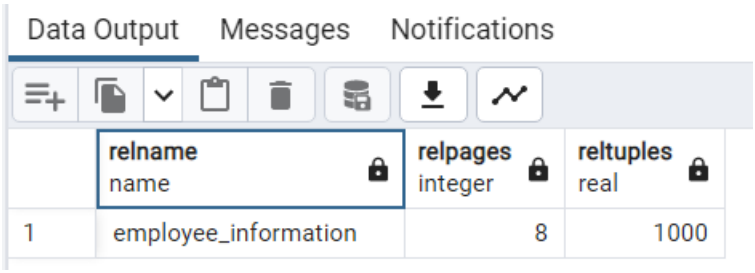
Size of each page is = 8KB

Number of Buffer Pages (BP) = 4MB/8KB = 512 buffer pages

QUERY:

```
select relname,relpages,reltuples

from pg_class

where relname='employee_information'
```

Output:



Here no of pages in outer relation (M) = 8

The outer relation(employee_information) of the Join in this query can indeed fit entirely in the buffer memory. Therefore, the database will use the Hash Join and also one more reason why database uses hash join because it is an equi join over the join attributes in this example. Here, a hash table will be constructed over the outer relation employee_information, with each hash value having its own associated tuples depending on the join attribute's hash function. The join attribute of the inner relation(department_information) will later be hashed using the same function, and we will search the outer relation's hash table for matches.

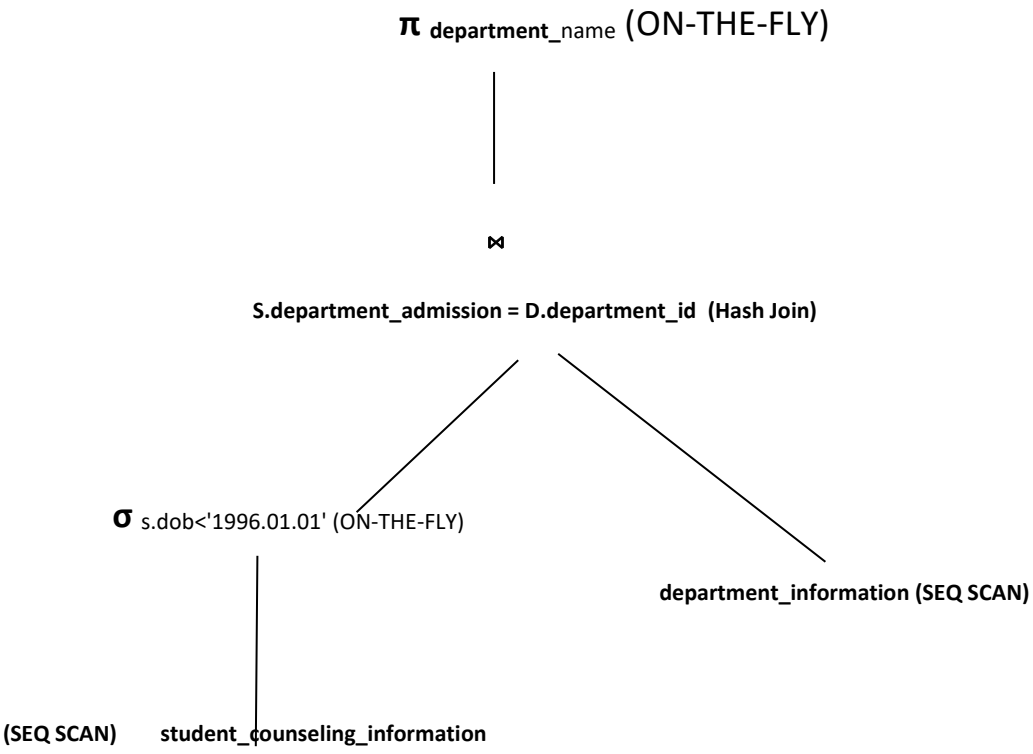
Query: 7

```
SELECT D.department_name
FROM student_counseling_information S
JOIN department_information D ON S.department_admission=D.department_id
Where S.dob<'1996.01.01';
```

Data Output | Messages | Notifications

	QUERY PLAN text
1	Hash Join (cost=1.90..86.81 rows=331 width=32)
2	Hash Cond: ((s.department_admission)::text = (d.department_id)::text)
3	-> Seq Scan on student_counseling_information s (cost=0.00..83.95 rows=331 w...
4	Filter: (dob < '1996-01-01'::date)
5	-> Hash (cost=1.40..1.40 rows=40 width=64)
6	-> Seq Scan on department_information d (cost=0.00..1.40 rows=40 width=64)

Physical Query Plan:



The join algorithm chosen by postgres is Hash Join.

Reason:

The query that I had chosen used Hash Join

The Hash Join is on the tables Employee_information and Department_information

Here outer relation is Employee_information and the Inner Relation is Department_information

Query:

show work_mem

	work_mem text
1	4MB

The maximum amount of memory that will be allocated to any query is 4MB

Size of each page is = 8KB

Number of Buffer Pages (BP) = 4MB/8KB = 512 buffer pages

QUERY:

select relname,relpages,reltuples

from pg_class

where relname='employee_information'

Output:

Data Output | Messages | Notifications

	relname name	relpages integer	reltuples real
1	employee_information	8	1000

Here no of pages in outer relation (M) = 8

The outer relation(employee_information) of the Join in this query can indeed fit entirely in the buffer memory. Therefore, the database will use the Hash Join and also one more reason why database uses hash join because it is an equi join over the join attributes in this example. Here, a hash table will be constructed over the outer relation employee_information, with each hash value having its own associated tuples depending on the join attribute's hash function. The join attribute of the inner relation(department_information) will later be hashed using the same function, and we will search the outer relation's hash table for matches.

4.Visualization:

Excel allows us to represent numerical data visually, making it easier to access and manage. For example, line graphs and bar graphs are only two examples of how we may organize data in Excel to make it more accessible.

By providing it with a visual context through graphs, this type of visualization provides a clear understanding of what the information implies. Making it simpler to spot patterns and trends in huge data sets is the visualization's major objective.

I believed that visualizing data gave us a clear understanding of what it meant by providing it with a visual context through maps or graphs, which made the data easier for the human mind to understand and made it much simpler to spot trends and patterns within big data sets.

Query 9:

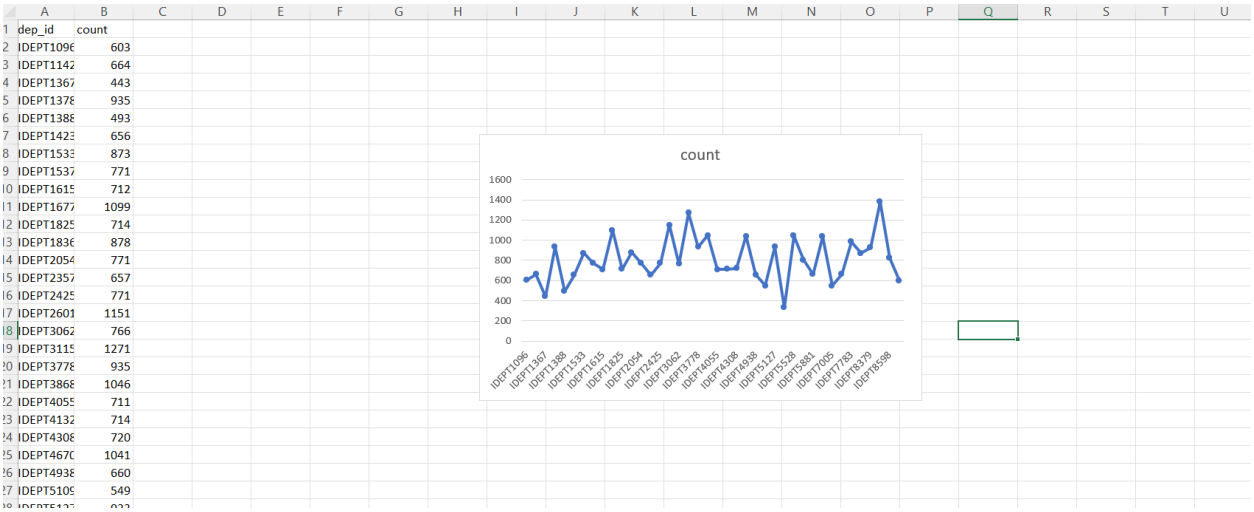
Query:

By looking at the visualization that I have done its providing a plot between the department_id and the count on how many papers got published by the particular department so, by looking at this visualization I can easily have an in-site of which department haspublished the maximum number or minimum number of papers.

SELECT DISTINCT SC.department_choices AS dep_id, COUNT(paper_id)

FROM student_performance_data SP JOIN student_counseling_information SC on

SC.student_id= SP.student_id Group by SC.department_choices;



dep_id	count
IDEPT1096	603
IDEPT1142	664
IDEPT1367	443
IDEPT1378	935
IDEPT1388	493
IDEPT1423	656
IDEPT1533	873
IDEPT1537	771
IDEPT1615	712
IDEPT1677	1099
IDEPT1825	714
IDEPT1836	878
IDEPT2054	771
IDEPT2357	657
IDEPT2425	771
IDEPT2601	1151
IDEPT3062	766
IDEPT3115	1271
IDEPT3778	935
IDEPT3868	1046
IDEPT4055	711
IDEPT4132	714
IDEPT4308	720
IDEPT4670	1041
IDEPT4938	660
IDEPT5109	549

Query 8:

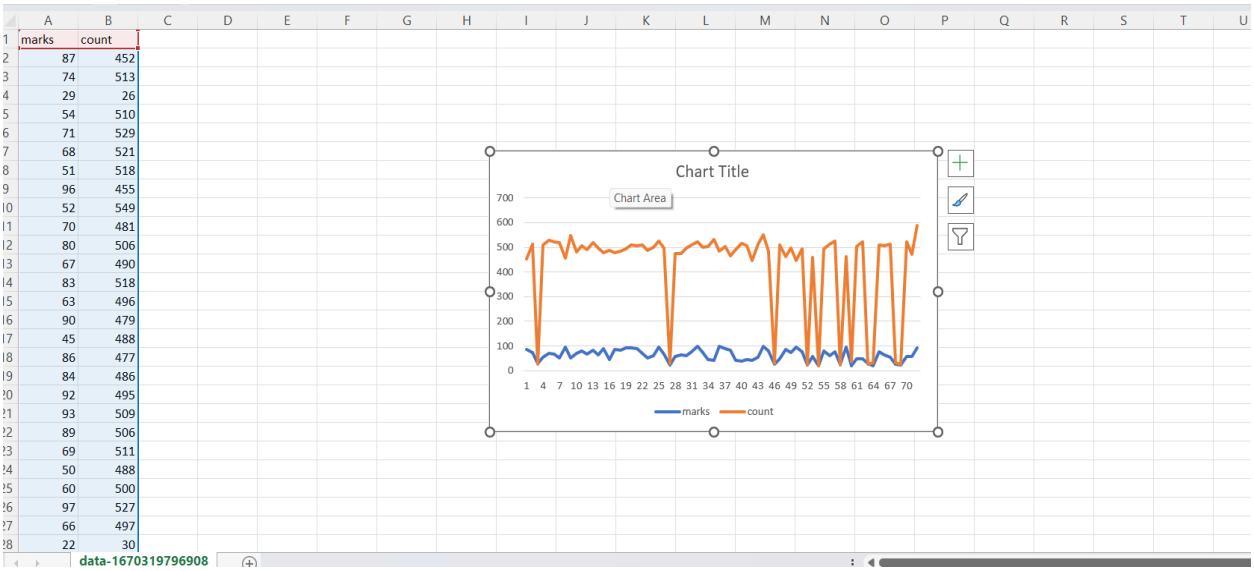
Query:

By looking at this visualization one can find out how the marks are distributed among the students and can know about the total performance of all the departments.

SELECT marks, COUNT(STUDENT_ID)

FROM student_performance_data

GROUP BY marks;

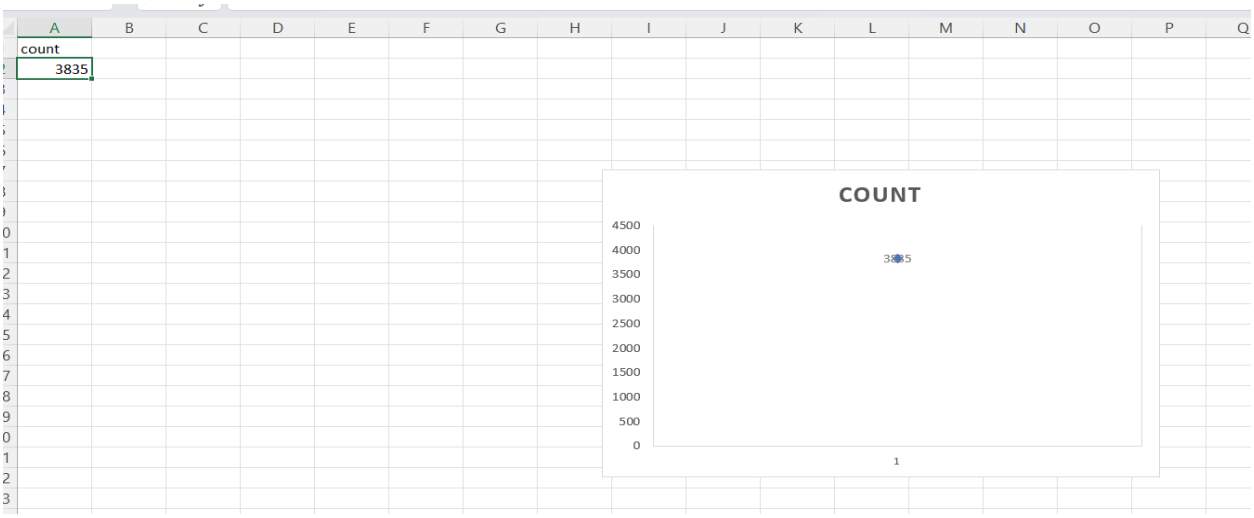


Query 2:

Query:

By looking at this visualization, we can actually know the count of the articles published in the sem 5. It is a simple visualization consisting of only 1 point with x attribute as semester name and y attribute as count.

```
SELECT COUNT (paper_id)

FROM student_performance_data

WHERE semster_name = 'Sem_5'
```



5. Presentation:

Presentation link:

https://drive.google.com/file/d/1m3MncW_H2DX9OUU0hePt3TGDs76LRoO4/view?usp=share_link