CS586 Introduction to Database Management

Apartment Rental Database

By: Hanesh Koganti

Finalize the dataset(s) that you proposed in Part I

- I have finalized to use 7 datasets for populating the relations that I have created in my Apartment Rental Database.

- I developed these 7 datasets from the raw data that I have collected from the Kaggle website. The raw data consist of the student apartment rental data of a college in India for the year 2021-2022, and the raw data is made of four individual csv files that contains null values, duplicates, and redundant data.

- So, before creating the database and populating the relations, I have considered cleaning and normalizing the data using the python pandas' tools which is a high-level data manipulation tool, so that it would be easy to create and work on the apartment rental database efficiently.

Describe how you are normalizing the original dataset if you create multiple tables.

The original raw data set that I got consists of 4 csv files where each of them consist of data about,

1. Tenant Data.csv
2. Apartment Rental Data.csv
3. Bill Payment Details.csv
4. Maintenance Details.csv

The above datasets contain multiple columns in them which consist of redundant data and null values associated with them.

For example, if we consider,

- The Apartment Rental Data.csv consists of much redundant data and null values for the information of the rooms that are not occupied by anyone.

- The Bill Payment Details.csv consists of all the details of the tenant along with the bill details which are redundant as that data is already available in TenantData.csv

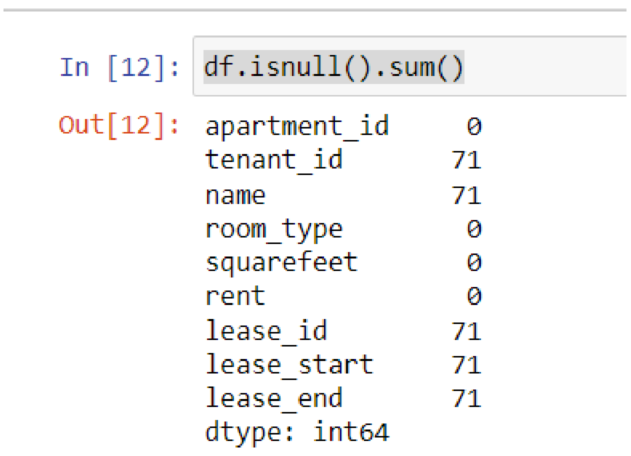
So, to remove this redundancy from the relations I split those four csv files into seven csv files that relate to each other with some sort of relationship among the attributes which are common in both the datasets. I have presented the common steps involved in this process down below,

I have used pandas' tools form python to do all this process and the process looks like this,

I. At first, I have Imported the apartmentdata.csv as dataframe named df using the below python command.

   - df=pd.read_csv(r'C:\apartment_rental_data\apartment.csv')

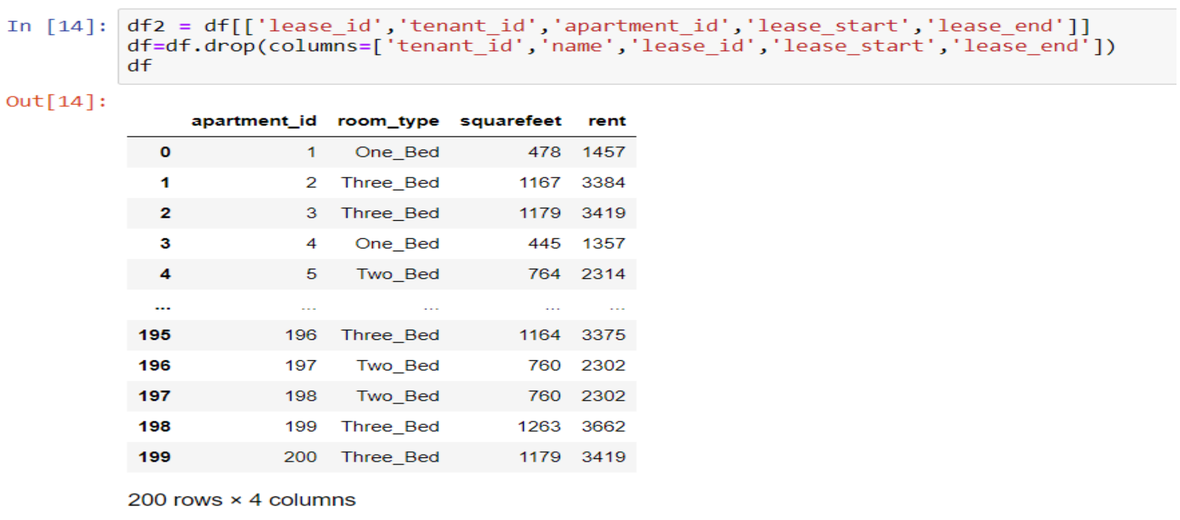
II. Then I had a look over the total no of null values present in each column of the apartment data by using the following command.
df.isnull().sum()



```
In [12]: df.isnull().sum()

Out[12]: apartment_id     0
         tenant_id       71
         name            71
         room_type        0
         squarefeet       0
         rent             0
         lease_id        71
         lease_start     71
         lease_end       71
         dtype: int64
```

III. So, to remove the null value from the raw apartment_rental_data.csv which are present due to the apartment rooms that are not occupied by anyone, I had divided this data frame into two different tables one containing only the information regarding the apartments rooms which doesn't contain any null values and other containing the leasing information with both tenant_id and apartment_id acting as foreign key attributes to their respective tables.

IV. So, by using the below commands we will get a new data frame that contains information regarding the leasing without containing any null values.



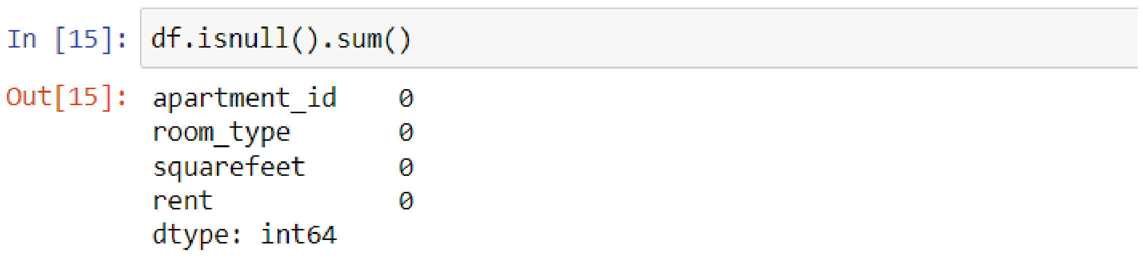
```
In [14]: df2 = df[['lease_id','tenant_id','apartment_id','lease_start','lease_end']]
         df=df.drop(columns=['tenant_id','name','lease_id','lease_start','lease_end'])
         df
```

Out[14]:

	apartment_id	room_type	squarefeet	rent
0	1	One_Bed	478	1457
1	2	Three_Bed	1167	3384
2	3	Three_Bed	1179	3419
3	4	One_Bed	445	1357
4	5	Two_Bed	764	2314
...	...	...	...	...
195	196	Three_Bed	1164	3375
196	197	Two_Bed	760	2302
197	198	Two_Bed	760	2302
198	199	Three_Bed	1263	3662
199	200	Three_Bed	1179	3419

200 rows × 4 columns

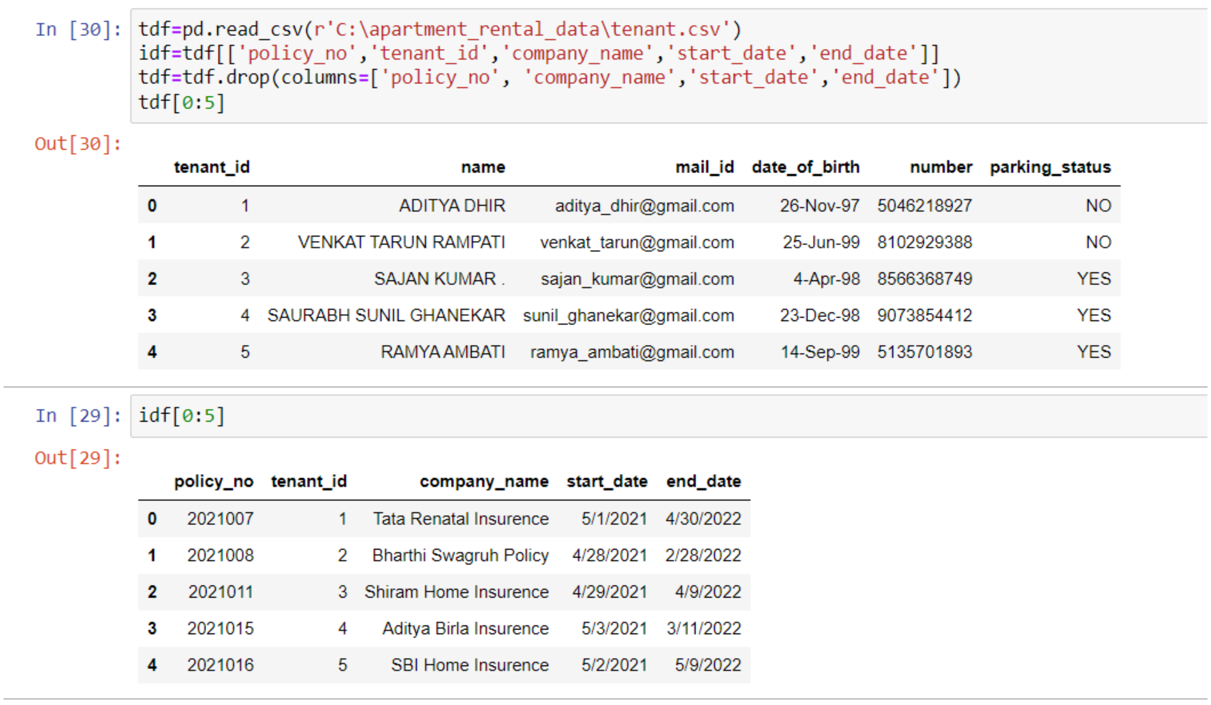
v. So, now if we check for the null values in the normalized apartment.csv table which will turn out to be zero.



```
In [15]: df.isnull().sum()

Out[15]: apartment_id    0
         room_type       0
         squarefeet      0
         rent            0
         dtype: int64
```

vi. We will follow a similar process over all the csv files that we have downloaded from Kaggle.

Vii. From the tenant table we will get two different data frames, one containing tenant information and the other one containing the insurance information where the tenant_id is acting as a foreign key between both the tables.



```
In [30]: tdf=pd.read_csv(r'C:\apartment_rental_data\tenant.csv')
         idf=tdf[['policy_no','tenant_id','company_name','start_date','end_date']]
         tdf=tdf.drop(columns=['policy_no', 'company_name','start_date','end_date'])
         tdf[0:5]
```

Out[30]:

	tenant_id	name	mail_id	date_of_birth	number	parking_status
0	1	ADITYA DHIR	aditya_dhir@gmail.com	26-Nov-97	5046218927	NO
1	2	VENKAT TARUN RAMPATI	venkat_tarun@gmail.com	25-Jun-99	8102929388	NO
2	3	SAJAN KUMAR .	sajan_kumar@gmail.com	4-Apr-98	8566368749	YES
3	4	SAURABH SUNIL GHANEKAR	sunil_ghanekar@gmail.com	23-Dec-98	9073854412	YES
4	5	RAMYA AMBATI	ramya_ambati@gmail.com	14-Sep-99	5135701893	YES

```
In [29]: idf[0:5]
```

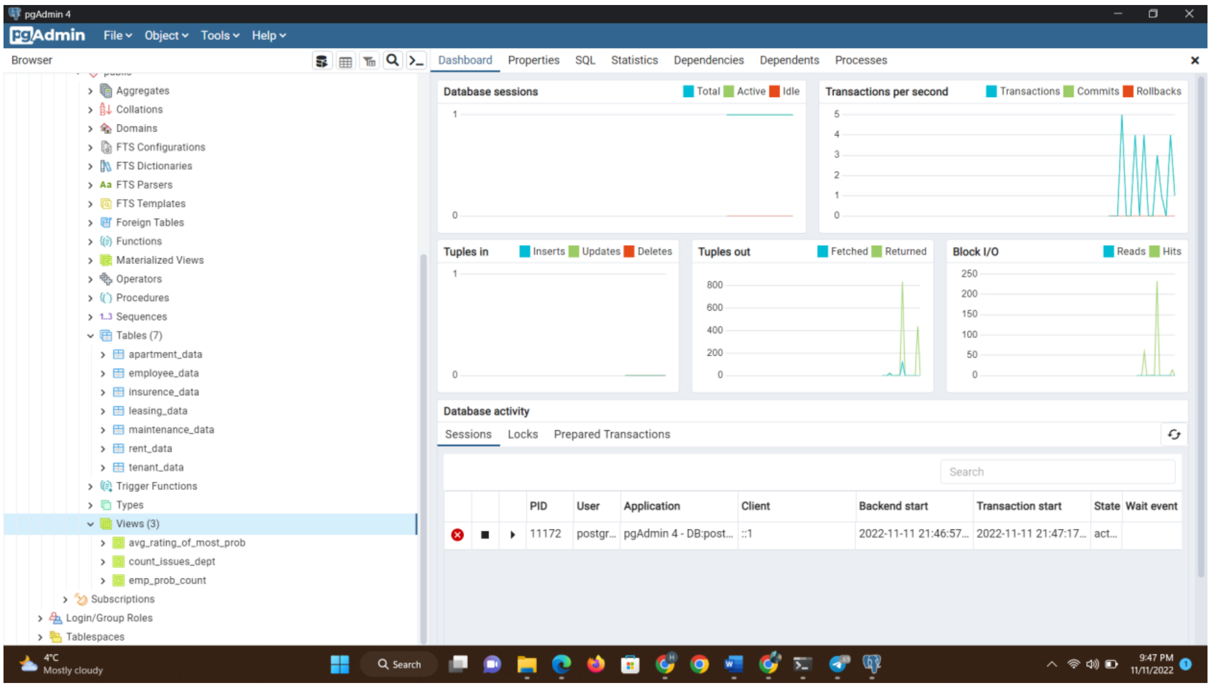
Out[29]:

	policy_no	tenant_id	company_name	start_date	end_date
0	2021007	1	Tata Renatal Insurence	5/1/2021	4/30/2022
1	2021008	2	Bharthi Swagruh Policy	4/28/2021	2/28/2022
2	2021011	3	Shiram Home Insurence	4/29/2021	4/9/2022
3	2021015	4	Aditya Birla Insurence	5/3/2021	3/11/2022
4	2021016	5	SBI Home Insurence	5/2/2021	5/9/2022

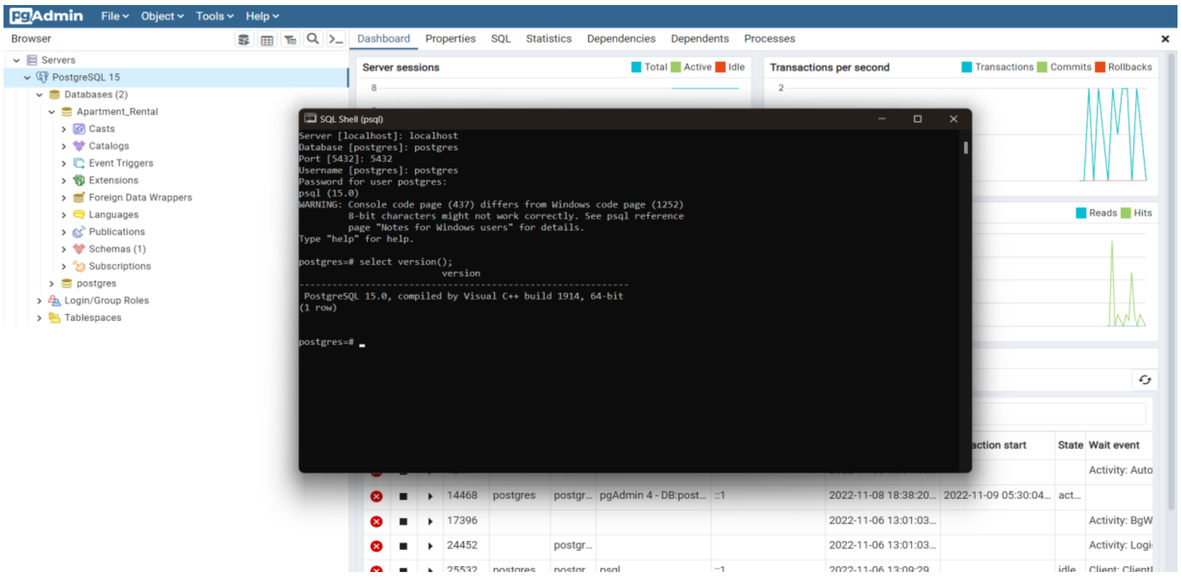
Viii. Similarly, from the bill table we will get one csv file by dropping the all-tenant details which are redundant apart from the tenant_id, which will act as a foreign key

ix. Finally from the maintenance table we will extract the employee details separately, by which the data will become less redundant, so that it can be stored and retrieved effectively from the database.

1. **Show that you have successfully installed Postgres and you can use pgAdmin or psql on your local machine or virtual machine (you can provide a screenshot)**

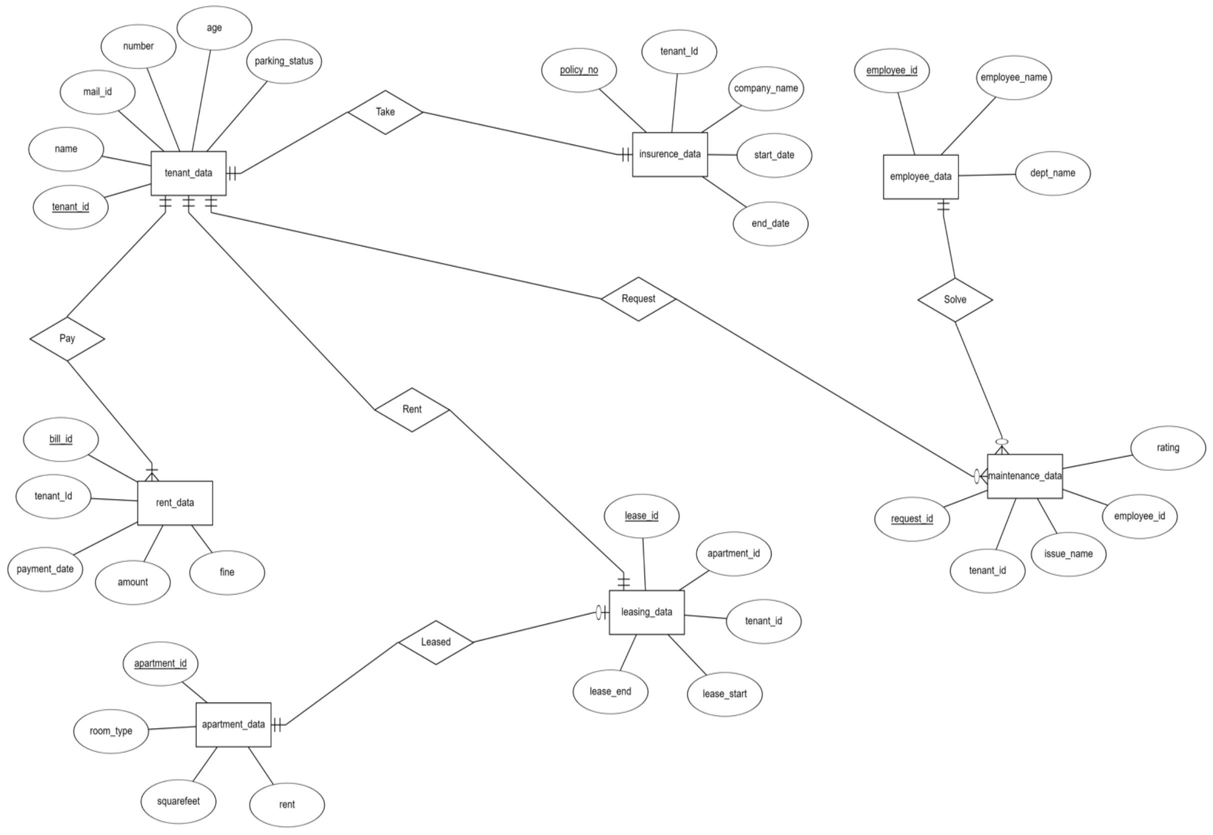


			PID	User	Application	Client	Backend start	Transaction start	State	Wait event
			11172	postgr...	pgAdmin 4 - DB:post...	::1	2022-11-11 21:46:57...	2022-11-11 21:47:17...	act...	



```
Server [localhost]: localhost
Database [postgres]: postgres
Port [5432]: 5432
Username [postgres]: postgres
Password for user postgres:
psql (15.0)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

postgres=# select version();
                          version
------------------------------------------------------------
 PostgreSQL 15.0, compiled by Visual C++ build 1914, 64-bit
(1 row)


postgres=# _
```

2. ER diagram



3. Database schema designed with your designed tables, attribute declaration, primary/foreign keys, views, or temporary tables etc. (You should demonstrate a variety of schema and SQL features such as a variety of data types, keys, foreign keys, different cardinalities).

Tenant Table:

tenant_data(<u>tenant_id</u> INT, name VARCHAR, mail_id VARCHAR, number VARCHAR[10], age INT, parking_status VARCHAR)

tenant_id is a primary key

Apartment Table:

apartment_data(<u>apartment_id</u> INT, room_type VARCHAR, squarefeet INT, rent INT)

apartment_id is a primary key

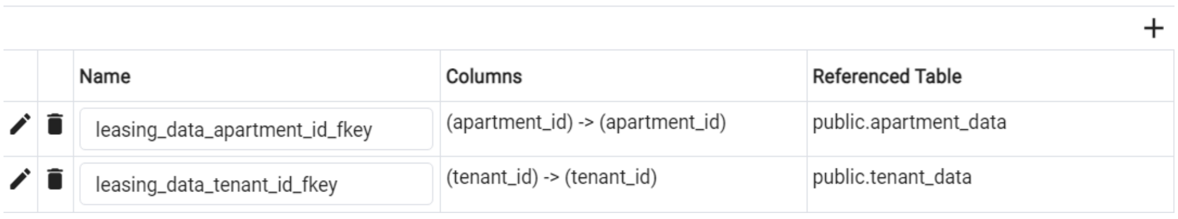
Leasing Table:

leasing_data(<u>lease_id</u> INT, tenant_id INT , apartment_id INT, , lease_start DATE, lease_end DATE)

lease_id is a primary key

tenant_id is a foreign key referencing tenant_data(tenant_id)

apartment_id is a foreign key referencing apartment_data(apartment_id)

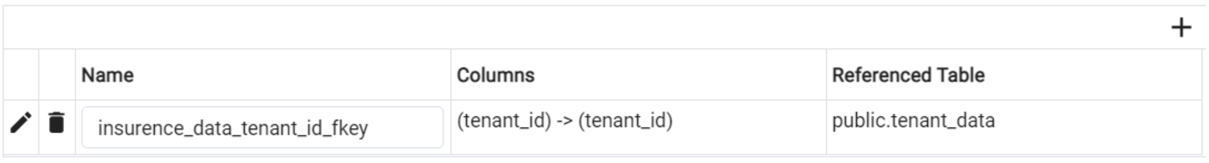


Name	Columns	Referenced Table
leasing_data_apartment_id_fkey	(apartment_id) -> (apartment_id)	public.apartment_data
leasing_data_tenant_id_fkey	(tenant_id) -> (tenant_id)	public.tenant_data

Insurance Table:

insurance_data(<u>policy_no</u> INT, tenant_id INT, company_name VARCHAR, start_date DATE, end_date DATE)

tenant_id is a foreign key referencing tenant_data(tenant_id)



		Name	Columns	Referenced Table
✎	🗑	insurence_data_tenant_id_fkey	(tenant_id) -> (tenant_id)	public.tenant_data

employee Table:

employee_data(<u>employee_id</u> INT, employee_name VARCHAR, dept_name VARCHAR)

employee_id is a primary key

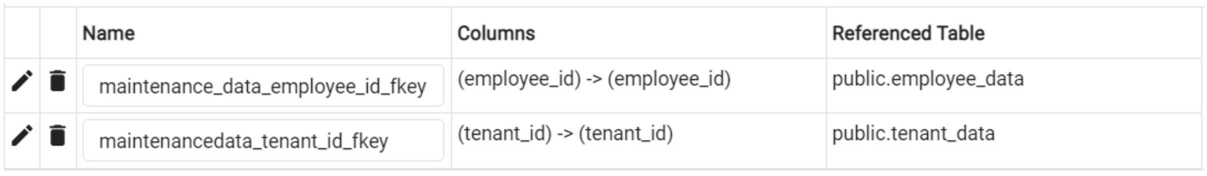
maintenance Table:

maintenance_data(<u>request_id</u> INT, tenant_id INT, issue_name VARCHAR, employee_id INT, rating INT)

request_id is a primary key

tenant_id is a foreign key referencing tenant_data(tenant_id)

employee_id is a foreign key referencing employee_data(employee_id)



		Name	Columns	Referenced Table
		maintenance_data_employee_id_fkey	(employee_id) -> (employee_id)	public.employee_data
		maintenancedata_tenant_id_fkey	(tenant_id) -> (tenant_id)	public.tenant_data

4.Data loading process and data preprocessing and cleaning. You should have effective data entry and avoid large amounts of manual data entry. Please describe what you have done to clean your data fully, what you have considered in cleaning, and how you have chosen to load and import your data in your tables.

There are two places where I have loaded the data that I have,

- ➢ At first, I loaded the raw data that I downloaded from Kaggle into Jupiter notebook as data frames and then I have worked on that data to remove the redundant data from these tables by normalizing them as mentioned above.
- ➢ Then after data preprocessing and cleaning I loaded this data to the relationships that I had created in my pgAdmin database.

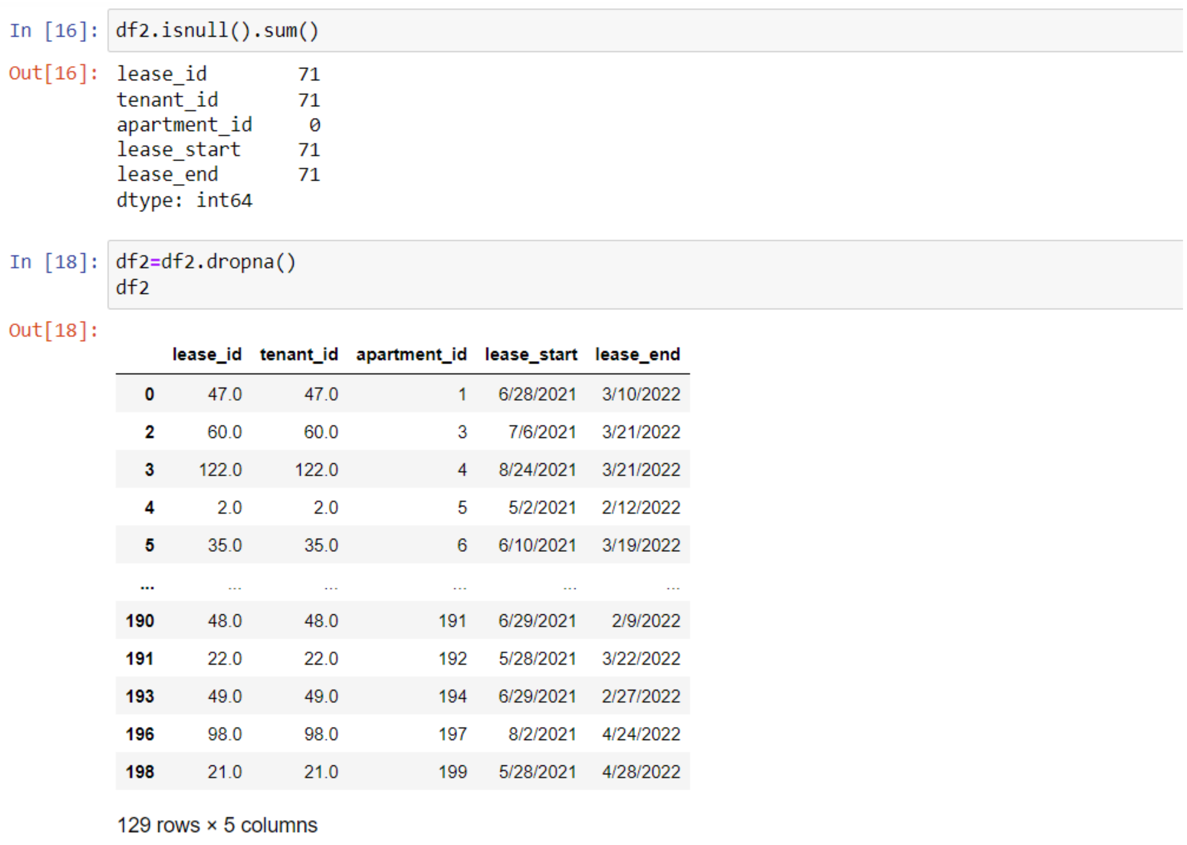
- ❖ So, to load the data into the Jupiter notebook I use the read_csv() function in r mode which is a built in function of pandas, a high level data manipulation tool.

  - df = pd.read_csv(r'C:\DBMS PROJECT\UniversityTenantDetails.csv')

- ❖ After loading all the seven data sets into separate data frames in the similar way as stated above, I had started working on the data preprocessing and data cleaning which is much required as it will ensure the integrity of the data and also make sure that the data is stored and retrieved effectively.

❖ After loading the data into the data frames then I look for the null values in the data by using df.isnull().sum() command and then dropped them using the df.dropna() command

- For example, in the leasing department data frame which I had created from the raw apartment rental data there are null values which have to be dropped by using the command as mentioned above.



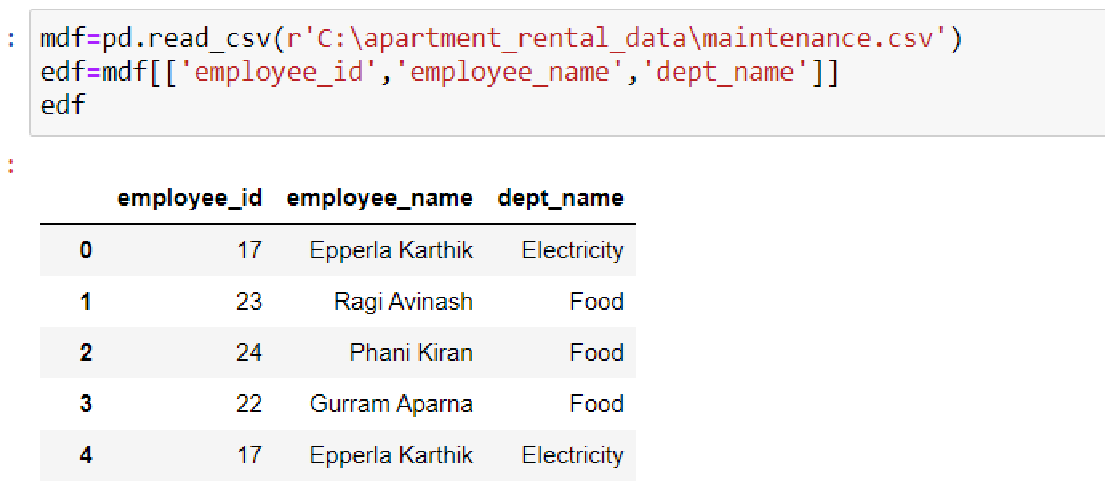
```
In [16]: df2.isnull().sum()

Out[16]: lease_id        71
         tenant_id       71
         apartment_id     0
         lease_start     71
         lease_end       71
         dtype: int64

In [18]: df2=df2.dropna()
         df2

Out[18]:
```

	lease_id	tenant_id	apartment_id	lease_start	lease_end
0	47.0	47.0	1	6/28/2021	3/10/2022
2	60.0	60.0	3	7/6/2021	3/21/2022
3	122.0	122.0	4	8/24/2021	3/21/2022
4	2.0	2.0	5	5/2/2021	2/12/2022
5	35.0	35.0	6	6/10/2021	3/19/2022
...	...	...	...	...	...
190	48.0	48.0	191	6/29/2021	2/9/2022
191	22.0	22.0	192	5/28/2021	3/22/2022
193	49.0	49.0	194	6/29/2021	2/27/2022
196	98.0	98.0	197	8/2/2021	4/24/2022
198	21.0	21.0	199	5/28/2021	4/28/2022

129 rows × 5 columns

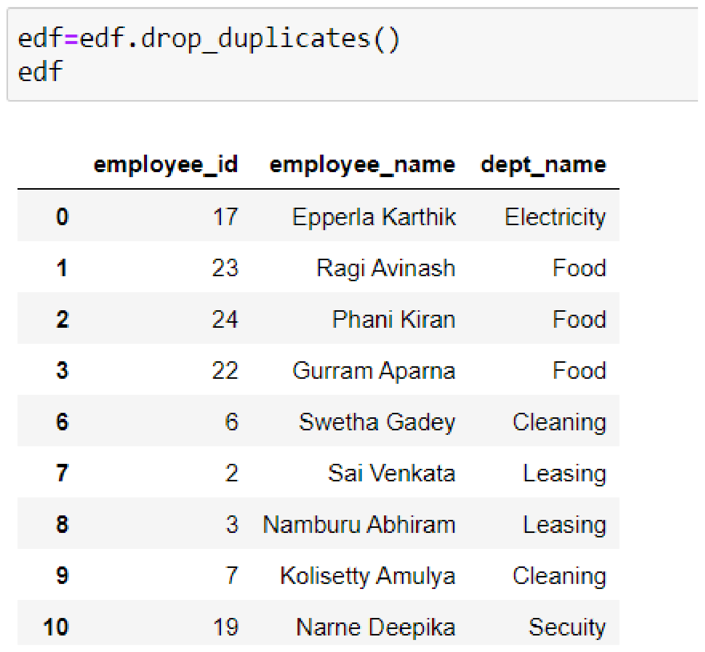
❖ After dropping the null values then I dropped the duplicate values that I found using the command duplicate=df[df.duplicated()] and then I used df.drop_duplicates() function for this purpose.

- For example, in the employee details dataframe that I have created from the raw maintenance data.csv, we can find the duplicate employee data which we look over and can drop using the above commands.



```
mdf=pd.read_csv(r'C:\apartment_rental_data\maintenance.csv')
edf=mdf[['employee_id','employee_name','dept_name']]
edf
```

	employee_id	employee_name	dept_name
0	17	Epperla Karthik	Electricity
1	23	Ragi Avinash	Food
2	24	Phani Kiran	Food
3	22	Gurram Aparna	Food
4	17	Epperla Karthik	Electricity



```
edf=edf.drop_duplicates()
edf
```

	employee_id	employee_name	dept_name
0	17	Epperla Karthik	Electricity
1	23	Ragi Avinash	Food
2	24	Phani Kiran	Food
3	22	Gurram Aparna	Food
6	6	Swetha Gadey	Cleaning
7	2	Sai Venkata	Leasing
8	3	Namburu Abhiram	Leasing
9	7	Kolisetty Amulya	Cleaning
10	19	Narne Deepika	Secuity

❖ Later I have dropped the columns that I don’t need such as ‘address’ in the tenant_data using the command,

- df=df.drop(columns=['address']);

❖ I also did changes to the format of the dates to the YYYY-MM-DD in leasing table, insurance table and the rent payment table by using the following command,

- df['coloumn_name'] = pd.to_datetime(df.coloumn_name)

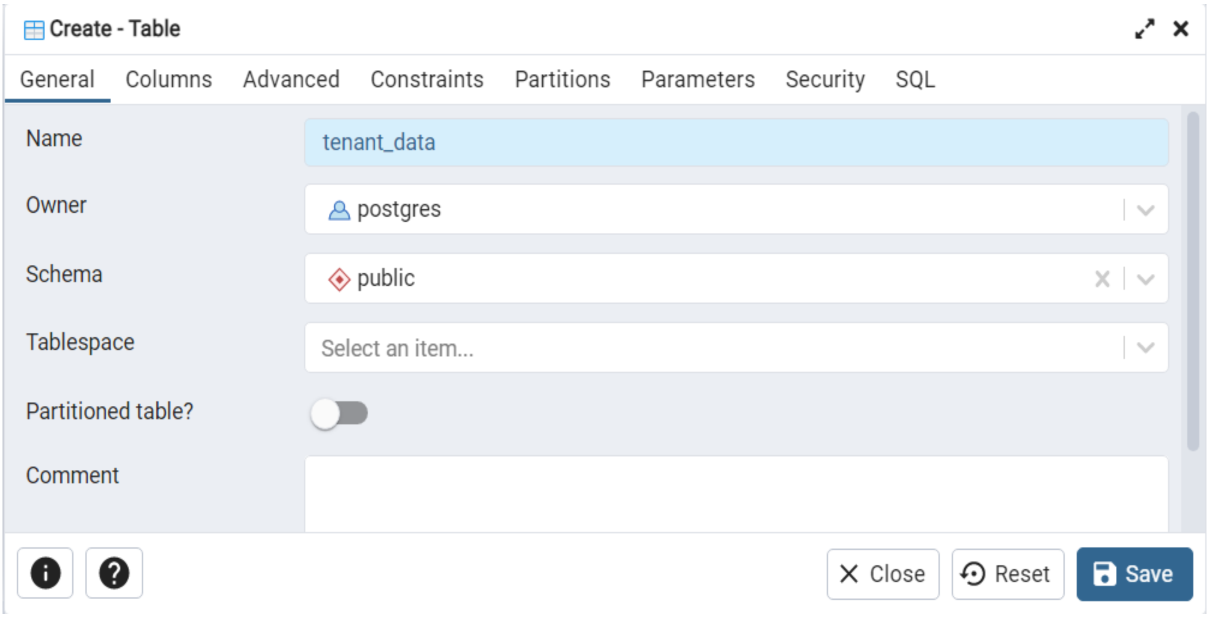
❖ I had performed the same process over all the tables that I got after normalization, so to ensure that they are free from null values and duplicated values.

Next,

To load the data into the PgAdmin, I have created the tables using the GUI interface provided in the PgAdmin database. I have attached the screenshots of the steps involved in the process.

Step 1:

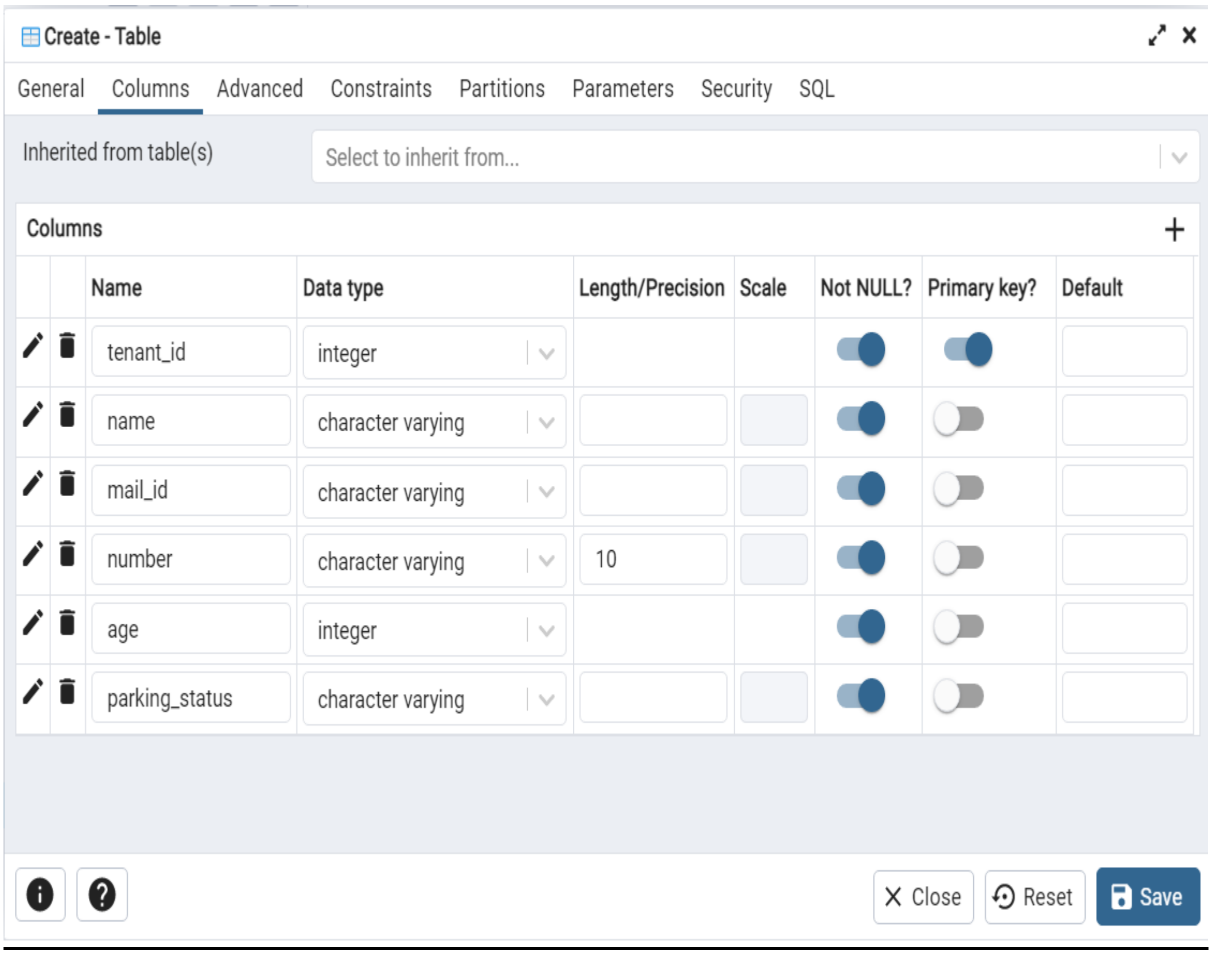
Defining Table Name:


Create - Table

Name	tenant_data
Owner	postgres
Schema	public
Tablespace	Select an item...
Partitioned table?	
Comment	

Step -2:

Defining the attributes of the table along with constrains:



Create - Table

General | Columns | Advanced | Constraints | Partitions | Parameters | Security | SQL

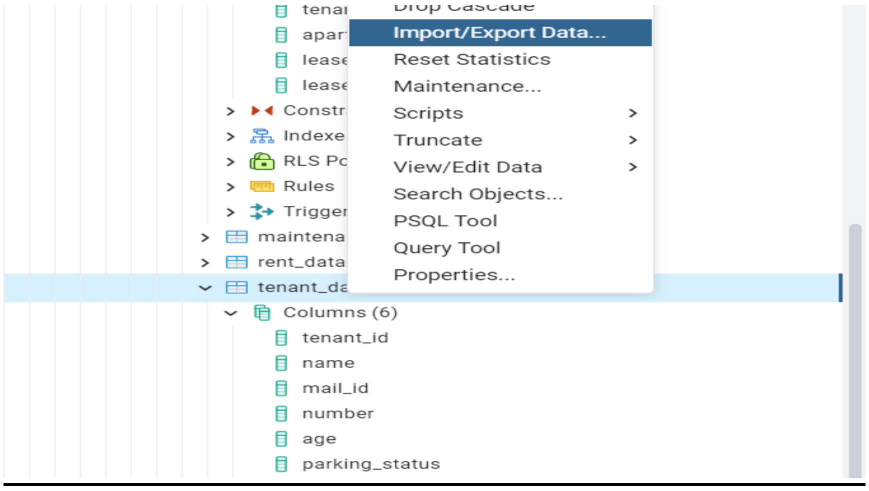
Inherited from table(s): Select to inherit from...

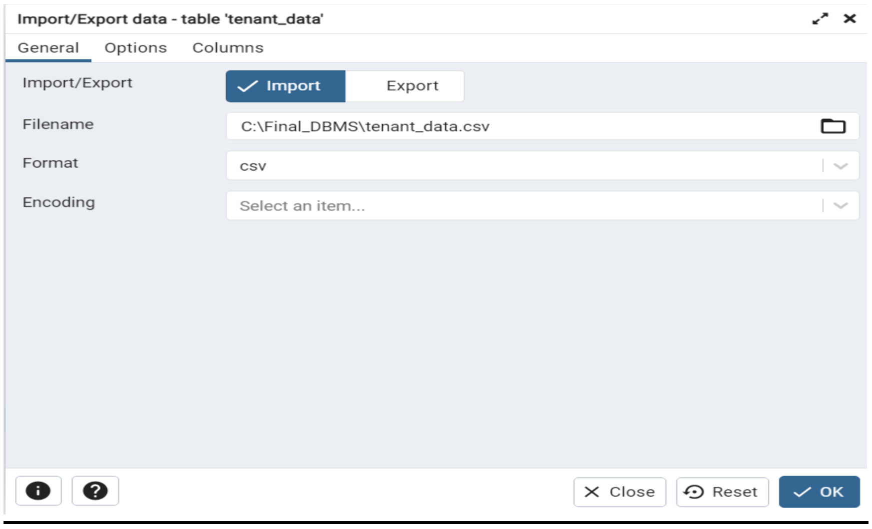
Columns

Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default
tenant_id	integer			✓	✓	
name	character varying			✓		
mail_id	character varying			✓		
number	character varying	10		✓		
age	integer			✓		
parking_status	character varying			✓		

Step-3:

Importing the CSV File:




Import/Export data - table 'tenant_data'

General Options Columns

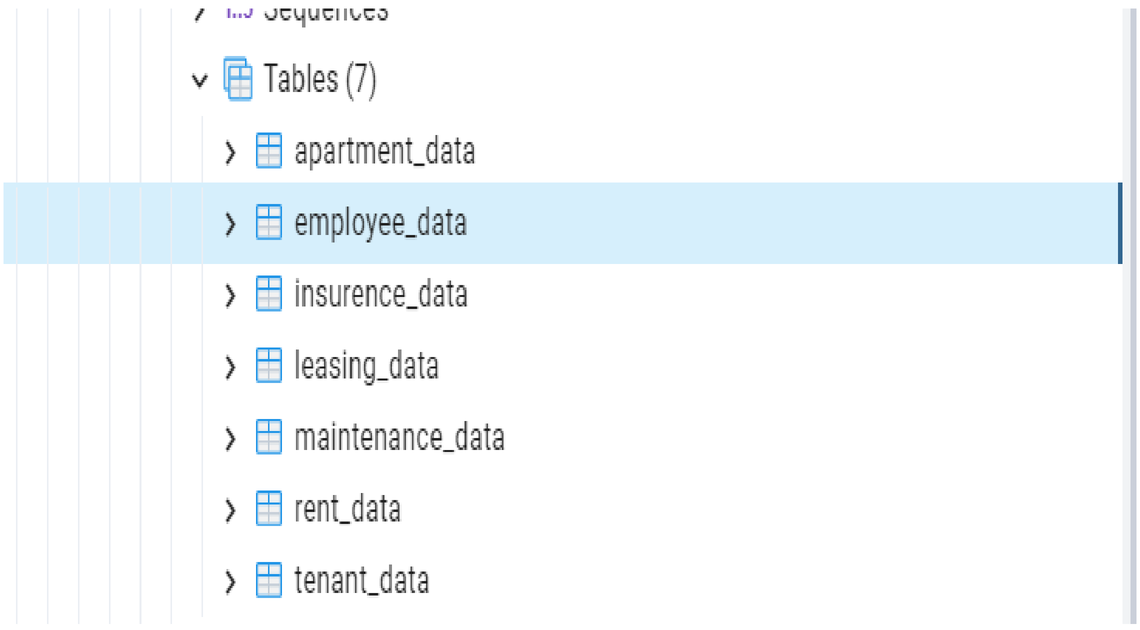
Import/Export	[✓ Import] Export
Filename	C:\Final_DBMS\tenant_data.csv
Format	csv
Encoding	Select an item...

5.Screenshots of the first five rows of all the populated tables. Your screenshots need to show sufficiently show your working environment as well.

Ans:

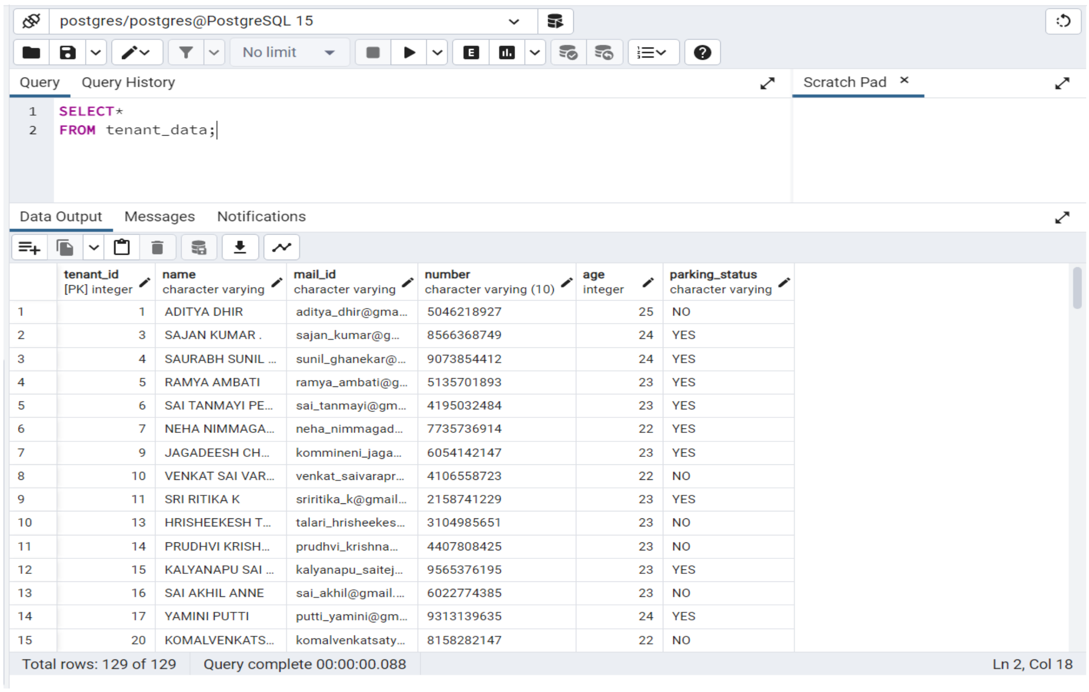
- All the relations in my database.

<u>All the relations:</u>



Sequences
Tables (7)
apartment_data
employee_data
insurence_data
leasing_data
maintenance_data
rent_data
tenant_data

tenant_data:

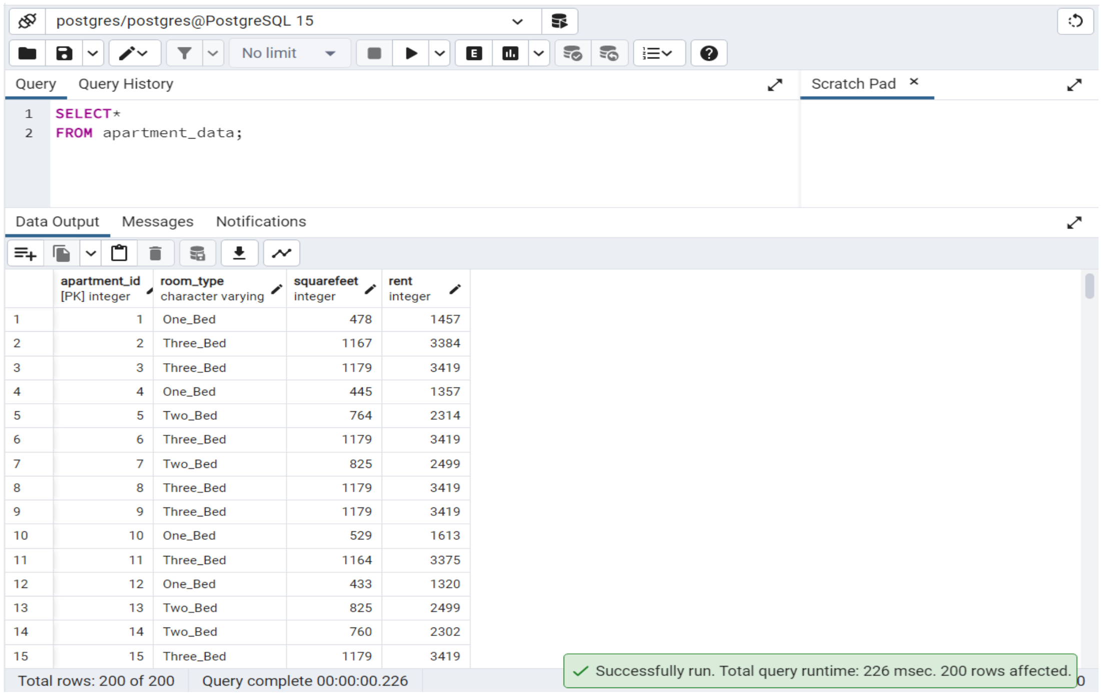


```
SELECT*
FROM tenant_data;
```

	tenant_id [PK] integer	name character varying	mail_id character varying	number character varying (10)	age integer	parking_status character varying
1	1	ADITYA DHIR	aditya_dhir@gma...	5046218927	25	NO
2	3	SAJAN KUMAR .	sajan_kumar@g...	8566368749	24	YES
3	4	SAURABH SUNIL ...	sunil_ghanekar@...	9073854412	24	YES
4	5	RAMYA AMBATI	ramya_ambati@g...	5135701893	23	YES
5	6	SAI TANMAYI PE...	sai_tanmayi@gm...	4195032484	23	YES
6	7	NEHA NIMMAGA...	neha_nimmagad...	7735736914	22	YES
7	9	JAGADEESH CH...	kommineni_jaga...	6054142147	23	YES
8	10	VENKAT SAI VAR...	venkat_saivarapr...	4106558723	22	NO
9	11	SRI RITIKA K	sriritika_k@gmail...	2158741229	23	YES
10	13	HRISHEEKESH T...	talari_hrisheekes...	3104985651	23	NO
11	14	PRUDHVI KRISH...	prudhvi_krishna...	4407808425	23	NO
12	15	KALYANAPU SAI ...	kalyanapu_saitej...	9565376195	23	YES
13	16	SAI AKHIL ANNE	sai_akhil@gmail....	6022774385	23	NO
14	17	YAMINI PUTTI	putti_yamini@gm...	9313139635	24	YES
15	20	KOMALVENKATS...	komalvenkatsaty...	8158282147	22	NO

Total rows: 129 of 129   Query complete 00:00:00.088   Ln 2, Col 18

apartment_data:



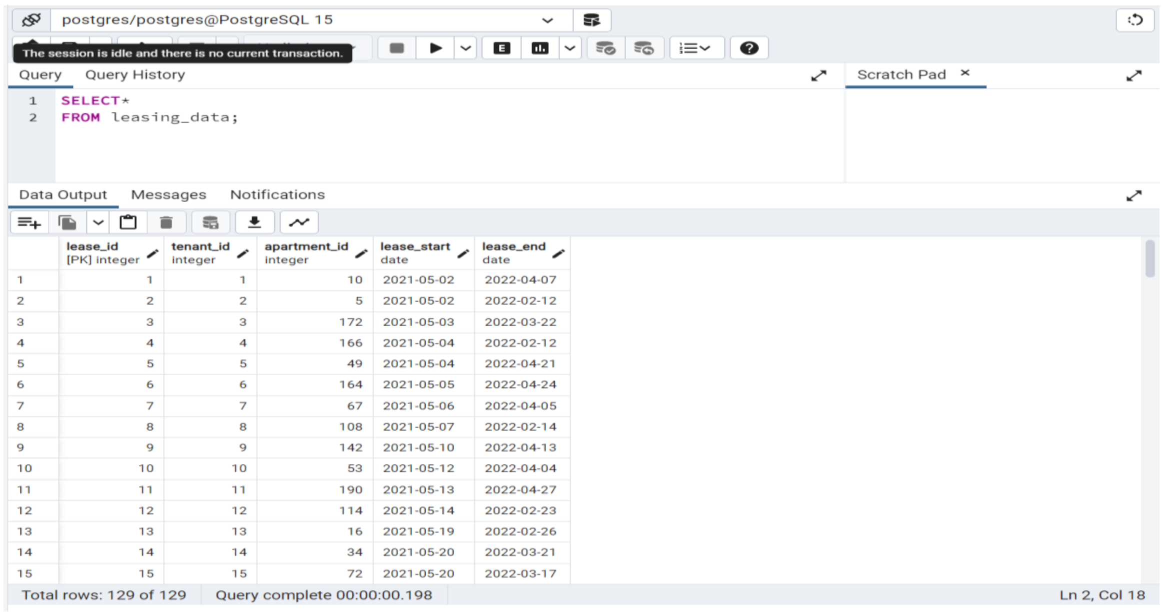
```
SELECT*
FROM apartment_data;
```

	apartment_id [PK] integer	room_type character varying	squarefeet integer	rent integer
1	1	One_Bed	478	1457
2	2	Three_Bed	1167	3384
3	3	Three_Bed	1179	3419
4	4	One_Bed	445	1357
5	5	Two_Bed	764	2314
6	6	Three_Bed	1179	3419
7	7	Two_Bed	825	2499
8	8	Three_Bed	1179	3419
9	9	Three_Bed	1179	3419
10	10	One_Bed	529	1613
11	11	Three_Bed	1164	3375
12	12	One_Bed	433	1320
13	13	Two_Bed	825	2499
14	14	Two_Bed	760	2302
15	15	Three_Bed	1179	3419

Total rows: 200 of 200   Query complete 00:00:00.226

✓ Successfully run. Total query runtime: 226 msec. 200 rows affected.

leasing_data:



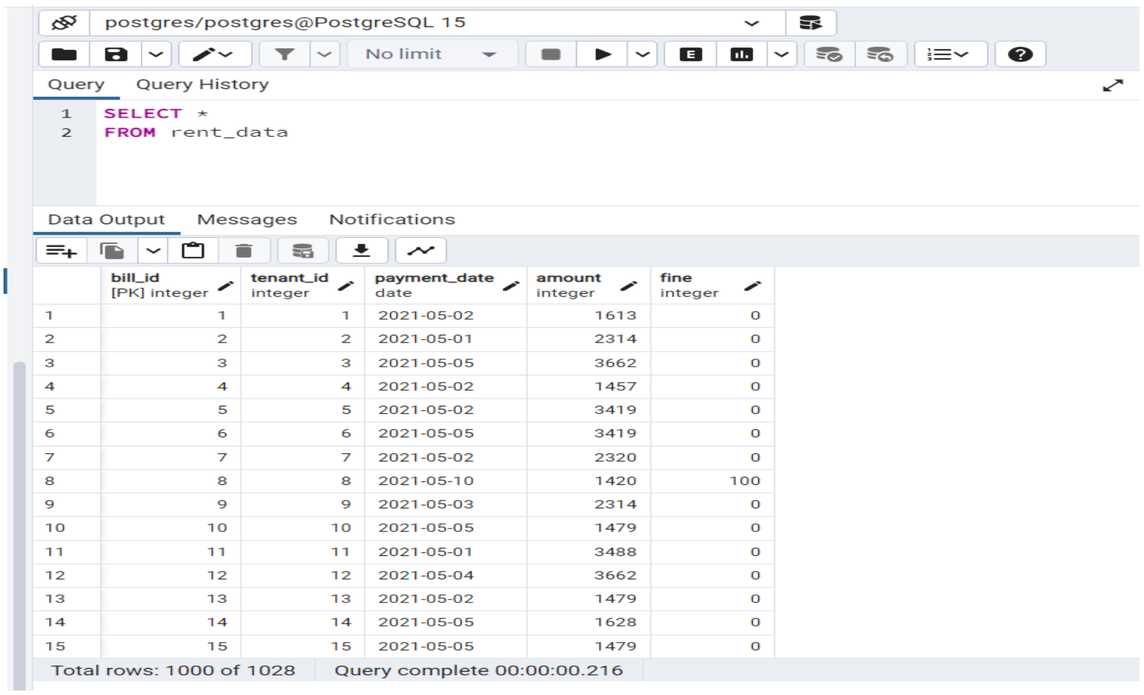
postgres/postgres@PostgreSQL 15
The session is idle and there is no current transaction.
Query
Query History
Scratch Pad
Data Output
Messages
Notifications

```sql
SELECT*
FROM leasing_data;
```

	lease_id [PK] integer	tenant_id integer	apartment_id integer	lease_start date	lease_end date
1	1	1	10	2021-05-02	2022-04-07
2	2	2	5	2021-05-02	2022-02-12
3	3	3	172	2021-05-03	2022-03-22
4	4	4	166	2021-05-04	2022-02-12
5	5	5	49	2021-05-04	2022-04-21
6	6	6	164	2021-05-05	2022-04-24
7	7	7	67	2021-05-06	2022-04-05
8	8	8	108	2021-05-07	2022-02-14
9	9	9	142	2021-05-10	2022-04-13
10	10	10	53	2021-05-12	2022-04-04
11	11	11	190	2021-05-13	2022-04-27
12	12	12	114	2021-05-14	2022-02-23
13	13	13	16	2021-05-19	2022-02-26
14	14	14	34	2021-05-20	2022-03-21
15	15	15	72	2021-05-20	2022-03-17

Total rows: 129 of 129   Query complete 00:00:00.198   Ln 2, Col 18

rent_data:

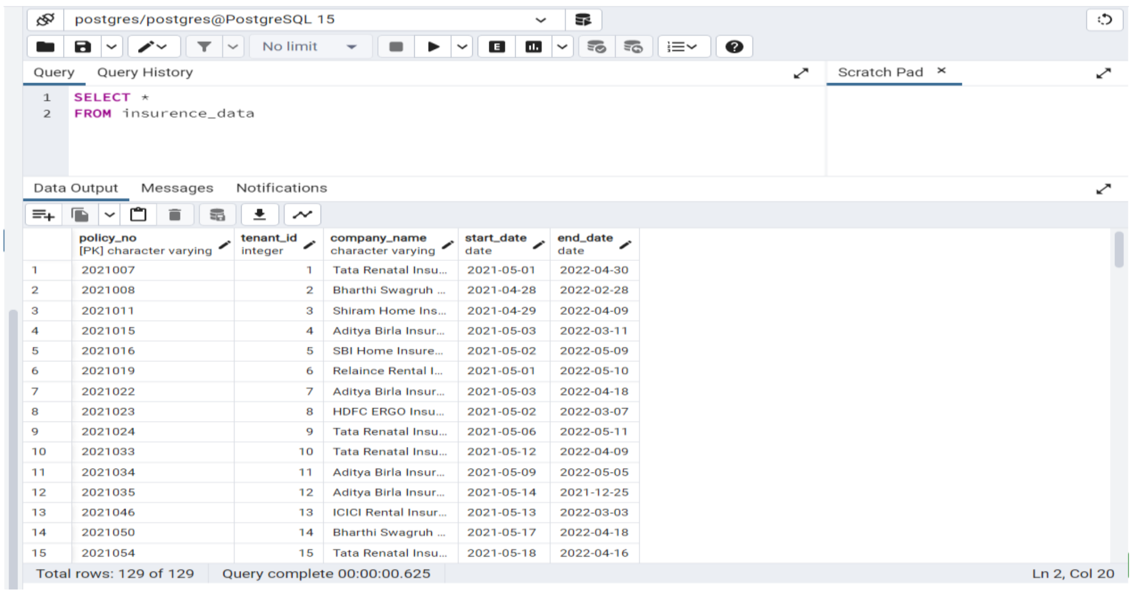


```sql
SELECT *
FROM rent_data
```

	bill_id [PK] integer	tenant_id integer	payment_date date	amount integer	fine integer
1	1	1	2021-05-02	1613	0
2	2	2	2021-05-01	2314	0
3	3	3	2021-05-05	3662	0
4	4	4	2021-05-02	1457	0
5	5	5	2021-05-02	3419	0
6	6	6	2021-05-05	3419	0
7	7	7	2021-05-02	2320	0
8	8	8	2021-05-10	1420	100
9	9	9	2021-05-03	2314	0
10	10	10	2021-05-05	1479	0
11	11	11	2021-05-01	3488	0
12	12	12	2021-05-04	3662	0
13	13	13	2021-05-02	1479	0
14	14	14	2021-05-05	1628	0
15	15	15	2021-05-05	1479	0

Total rows: 1000 of 1028   Query complete 00:00:00.216

insurance_data:



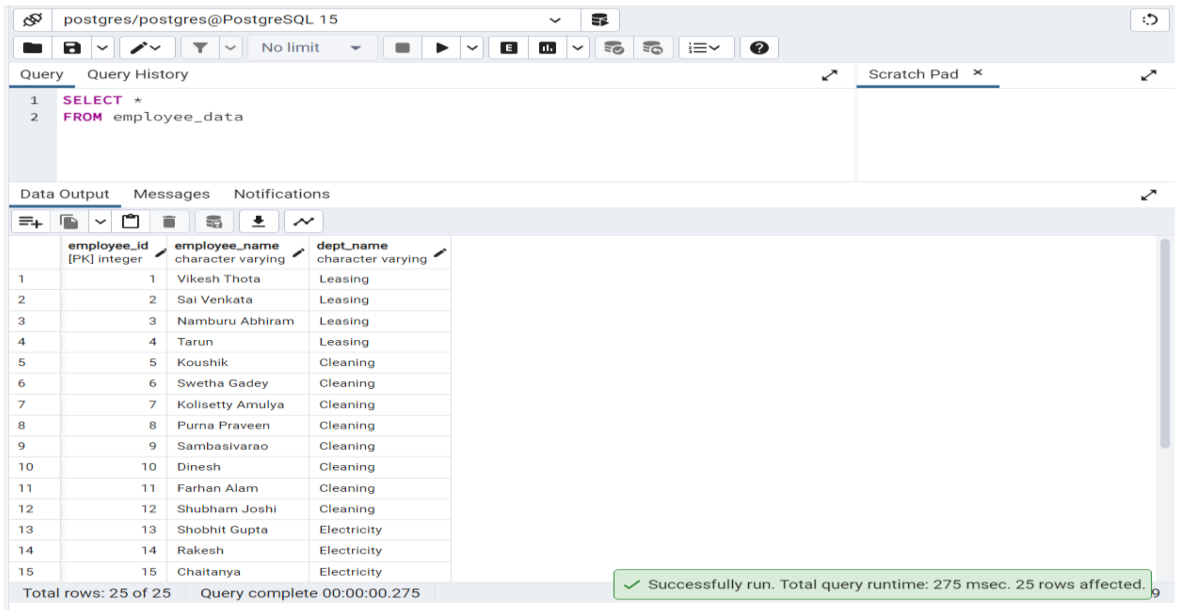
postgres/postgres@PostgreSQL 15

```
SELECT *
FROM insurence_data
```

	policy_no [PK] character varying	tenant_id integer	company_name character varying	start_date date	end_date date
1	2021007	1	Tata Renatal Insu...	2021-05-01	2022-04-30
2	2021008	2	Bharthi Swagruh ...	2021-04-28	2022-02-28
3	2021011	3	Shiram Home Ins...	2021-04-29	2022-04-09
4	2021015	4	Aditya Birla Insur...	2021-05-03	2022-03-11
5	2021016	5	SBI Home Insure...	2021-05-02	2022-05-09
6	2021019	6	Relaince Rental I...	2021-05-01	2022-05-10
7	2021022	7	Aditya Birla Insur...	2021-05-03	2022-04-18
8	2021023	8	HDFC ERGO Insu...	2021-05-02	2022-03-07
9	2021024	9	Tata Renatal Insu...	2021-05-06	2022-05-11
10	2021033	10	Tata Renatal Insu...	2021-05-12	2022-04-09
11	2021034	11	Aditya Birla Insur...	2021-05-09	2022-05-05
12	2021035	12	Aditya Birla Insur...	2021-05-14	2021-12-25
13	2021046	13	ICICI Rental Insur...	2021-05-13	2022-03-03
14	2021050	14	Bharthi Swagruh ...	2021-05-17	2022-04-18
15	2021054	15	Tata Renatal Insu...	2021-05-18	2022-04-16

Total rows: 129 of 129 Query complete 00:00:00.625 Ln 2, Col 20

employee_data:



postgres/postgres@PostgreSQL 15

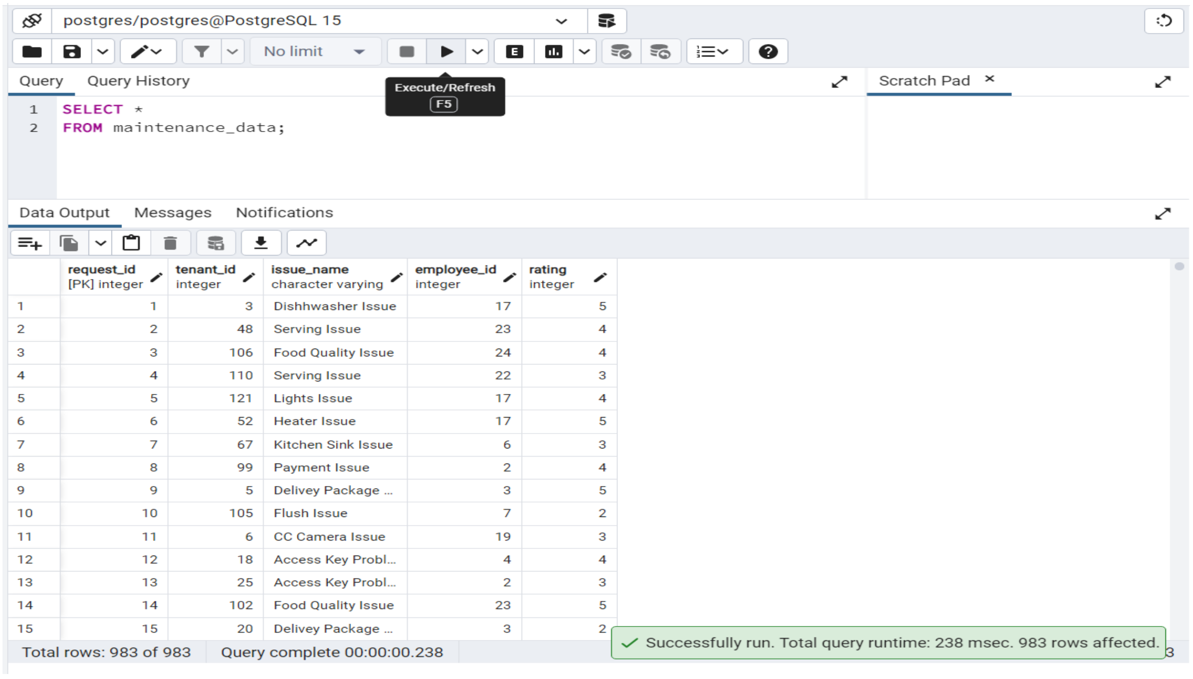
```
SELECT *
FROM employee_data
```

	employee_id [PK] integer	employee_name character varying	dept_name character varying
1	1	Vikesh Thota	Leasing
2	2	Sai Venkata	Leasing
3	3	Namburu Abhiram	Leasing
4	4	Tarun	Leasing
5	5	Koushik	Cleaning
6	6	Swetha Gadey	Cleaning
7	7	Kolisetty Amulya	Cleaning
8	8	Purna Praveen	Cleaning
9	9	Sambasivarao	Cleaning
10	10	Dinesh	Cleaning
11	11	Farhan Alam	Cleaning
12	12	Shubham Joshi	Cleaning
13	13	Shobhit Gupta	Electricity
14	14	Rakesh	Electricity
15	15	Chaitanya	Electricity

Total rows: 25 of 25 Query complete 00:00:00.275

✓ Successfully run. Total query runtime: 275 msec. 25 rows affected.

maintenance_data:



```
SELECT *
FROM maintenance_data;
```

	request_id [PK] integer	tenant_id integer	issue_name character varying	employee_id integer	rating integer
1	1	3	Dishhwasher Issue	17	5
2	2	48	Serving Issue	23	4
3	3	106	Food Quality Issue	24	4
4	4	110	Serving Issue	22	3
5	5	121	Lights Issue	17	4
6	6	52	Heater Issue	17	5
7	7	67	Kitchen Sink Issue	6	3
8	8	99	Payment Issue	2	4
9	9	5	Delivey Package ...	3	5
10	10	105	Flush Issue	7	2
11	11	6	CC Camera Issue	19	3
12	12	18	Access Key Probl...	4	4
13	13	25	Access Key Probl...	2	3
14	14	102	Food Quality Issue	23	5
15	15	20	Delivey Package ...	3	2

Total rows: 983 of 983 Query complete 00:00:00.238

✓ Successfully run. Total query runtime: 238 msec. 983 rows affected.

5. **Your 10 English questions and 10 query execution of those questions with screenshots of the first five rows of output and the total number of rows in the result. Design a variety of complex questions, as mentioned earlier.**

   Any modification that was necessary to your proposal in part I.

1. **List the apartments that is having lowest cost per square feet?**

Ans:

select apartment_id

from((select apartment_id, (rent/squarefeet) as low

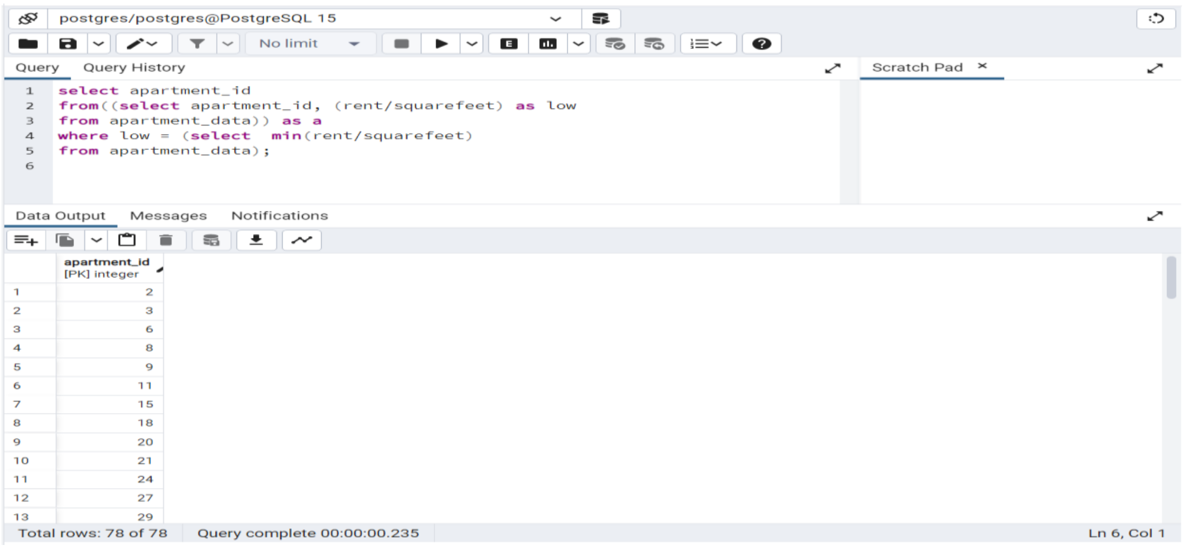
from apartment_data)) as a

where low = (select min(rent/squarefeet)

from apartment_data);

<u>OUTPUT:</u>

No of rows returned: 78



```
select apartment_id
from((select apartment_id, (rent/squarefeet) as low
from apartment_data)) as a
where low = (select  min(rent/squarefeet)
from apartment_data);
```

```
    apartment_id
    [PK] integer
1              2
2              3
3              6
4              8
5              9
6             11
7             15
8             18
9             20
10            21
11            24
12            27
13            29
Total rows: 78 of 78    Query complete 00:00:00.235    Ln 6, Col 1
```

2. List the apartments which are not occupied by any customers?

select *

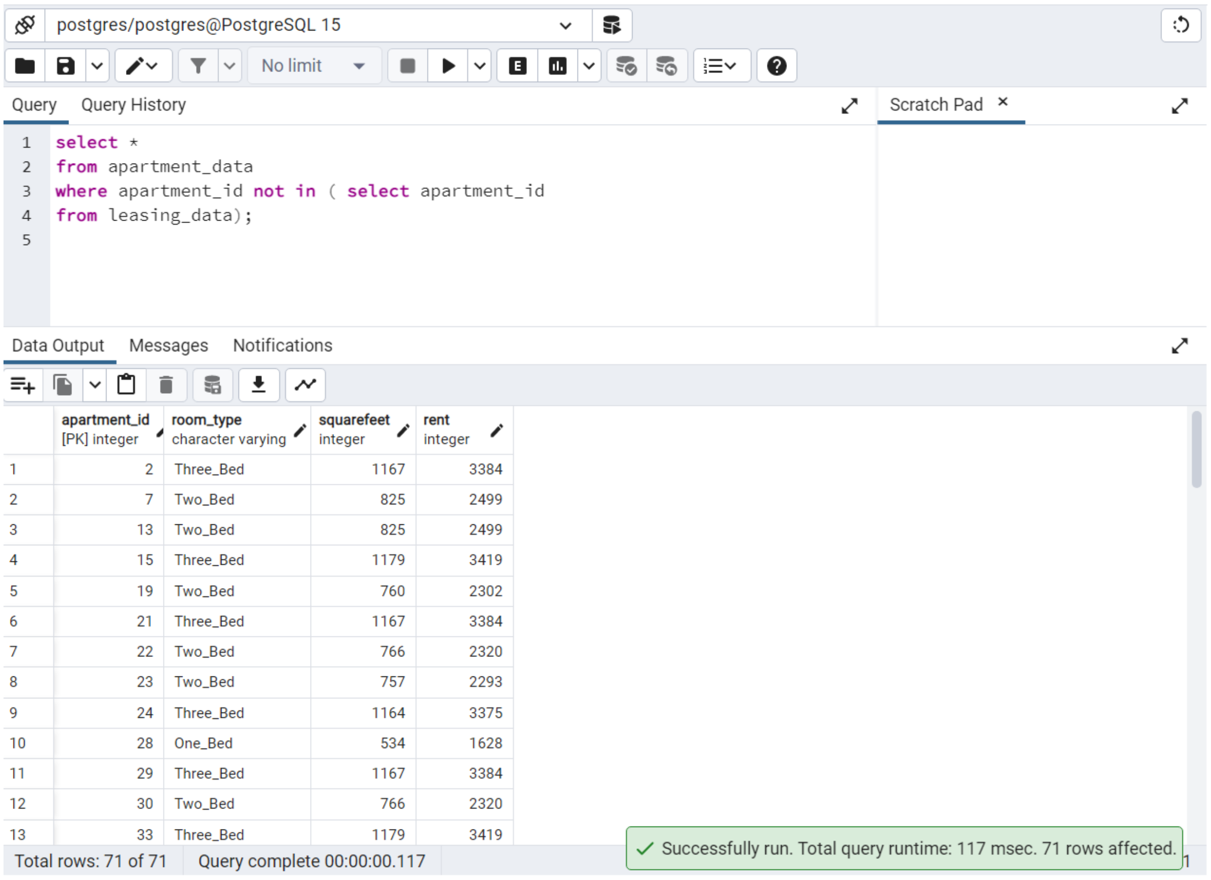
from apartment_data

where apartment_id not in (select apartment_id

from leasing_data);

OUTPUT:

No of rows returned: 71



postgres/postgres@PostgreSQL 15

No limit

Query | Query History | Scratch Pad

```
select *
from apartment_data
where apartment_id not in ( select apartment_id
from leasing_data);
```

Data Output | Messages | Notifications

	apartment_id [PK] integer	room_type character varying	squarefeet integer	rent integer
1	2	Three_Bed	1167	3384
2	7	Two_Bed	825	2499
3	13	Two_Bed	825	2499
4	15	Three_Bed	1179	3419
5	19	Two_Bed	760	2302
6	21	Three_Bed	1167	3384
7	22	Two_Bed	766	2320
8	23	Two_Bed	757	2293
9	24	Three_Bed	1164	3375
10	28	One_Bed	534	1628
11	29	Three_Bed	1167	3384
12	30	Two_Bed	766	2320
13	33	Three_Bed	1179	3419

Total rows: 71 of 71 Query complete 00:00:00.117

✓ Successfully run. Total query runtime: 117 msec. 71 rows affected.

3.List all the Tenants who are senior citizens? (OLD QUESTION)

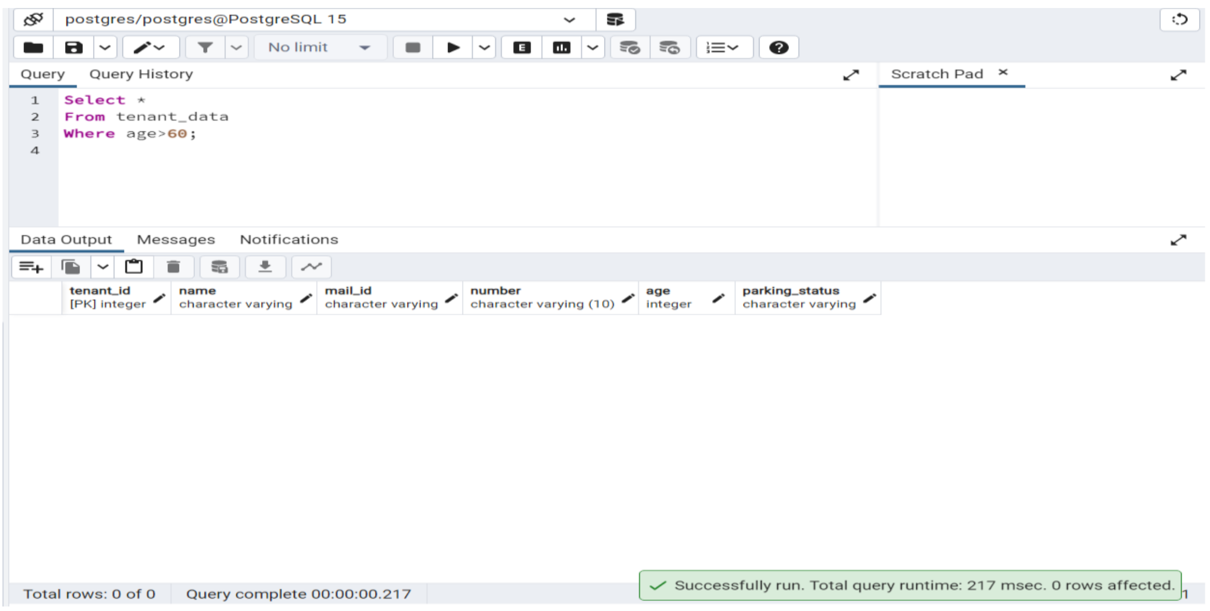
(I changed the Question)

For old question:

```
Select *
From tenant_data
Where age>60;
```

Output:

Zero rows returned



postgres/postgres@PostgreSQL 15

No limit

Query Query History Scratch Pad

```
Select *
From tenant_data
Where age>60;
```

Data Output Messages Notifications

```
tenant_id      name               mail_id            number                  age       parking_status
[PK] integer   character varying  character varying  character varying (10)  integer   character varying
```

Total rows: 0 of 0   Query complete 00:00:00.217

Successfully run. Total query runtime: 217 msec. 0 rows affected.

➔ But as the number of rows returned is Zero and the query is very simple, I wanted to change this question.

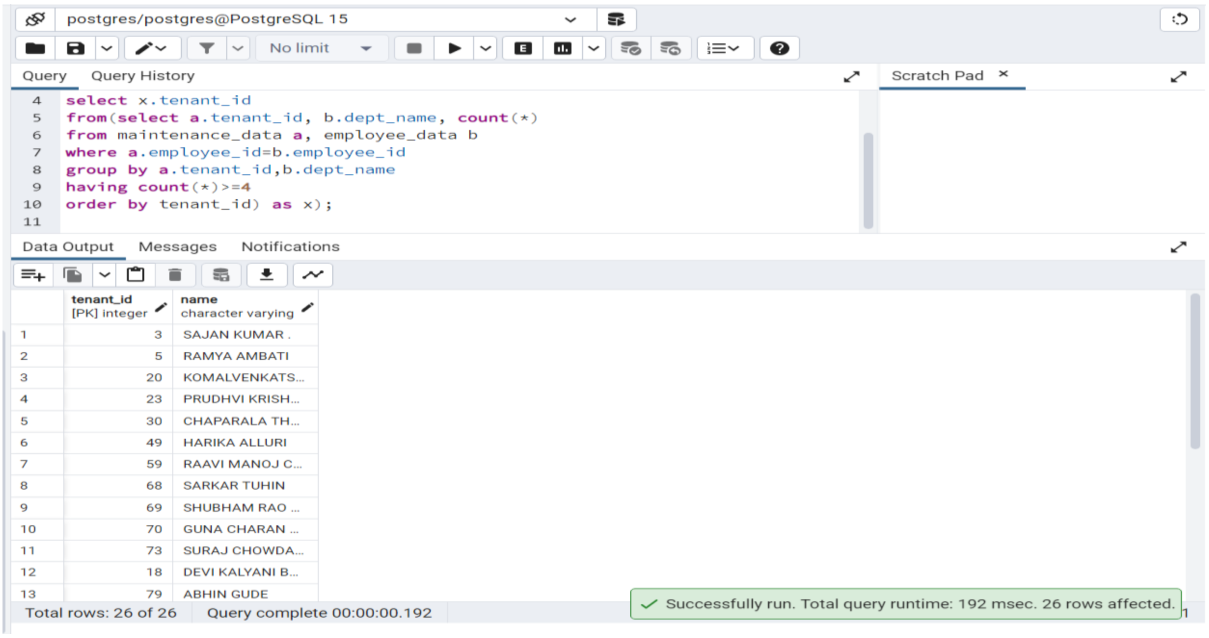
3. Print the names of the tenants who have faced at least four problems in any department. (Changed)

Ans:

```
select x.tenant_id
from(select a.tenant_id, b.dept_name, count(*)
from maintenance_data a, employee_data b
where a.employee_id=b.employee_id
group by a.tenant_id,b.dept_name
having count(*)>=4
order by tenant_id) as x);
```

<u>OUTPUT:</u>

No of returned is 26



postgres/postgres@PostgreSQL 15

No limit

Query | Query History | Scratch Pad

```
select x.tenant_id
from(select a.tenant_id, b.dept_name, count(*)
from maintenance_data a, employee_data b
where a.employee_id=b.employee_id
group by a.tenant_id,b.dept_name
having count(*)>=4
order by tenant_id) as x);
```

Data Output | Messages | Notifications

	tenant_id [PK] integer	name character varying
1	3	SAJAN KUMAR .
2	5	RAMYA AMBATI
3	20	KOMALVENKATS...
4	23	PRUDHVI KRISH...
5	30	CHAPARALA TH...
6	49	HARIKA ALLURI
7	59	RAAVI MANOJ C...
8	68	SARKAR TUHIN
9	69	SHUBHAM RAO ...
10	70	GUNA CHARAN ...
11	73	SURAJ CHOWDA...
12	18	DEVI KALYANI B...
13	79	ABHIN GUDE

Total rows: 26 of 26 Query complete 00:00:00.192

✓ Successfully run. Total query runtime: 192 msec. 26 rows affected.

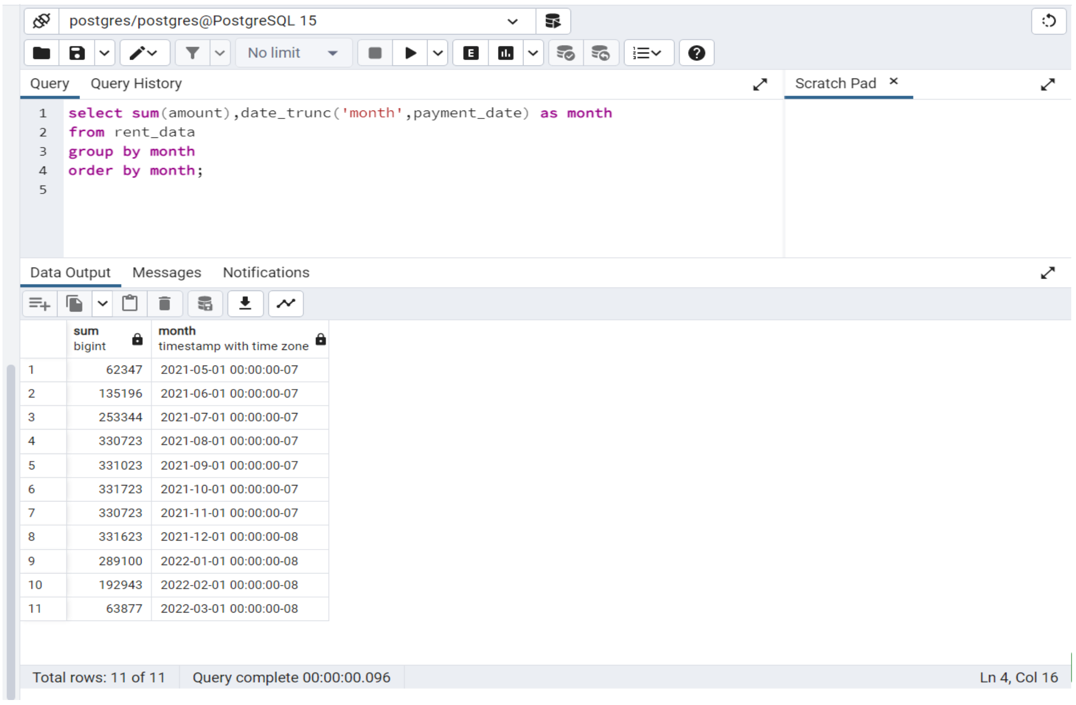
4. Return the total rental revenue generated per month by the apartment

Ans:

```
select sum(amount),date_trunc('month',payment_date) as month
from rent_data
group by month
order by month;
```

OUTPUT:

No of rows returned is 11



```
select sum(amount),date_trunc('month',payment_date) as month
from rent_data
group by month
order by month;
```

	sum bigint	month timestamp with time zone
1	62347	2021-05-01 00:00:00-07
2	135196	2021-06-01 00:00:00-07
3	253344	2021-07-01 00:00:00-07
4	330723	2021-08-01 00:00:00-07
5	331023	2021-09-01 00:00:00-07
6	331723	2021-10-01 00:00:00-07
7	330723	2021-11-01 00:00:00-07
8	331623	2021-12-01 00:00:00-08
9	289100	2022-01-01 00:00:00-08
10	192943	2022-02-01 00:00:00-08
11	63877	2022-03-01 00:00:00-08

Total rows: 11 of 11 Query complete 00:00:00.096 Ln 4, Col 16

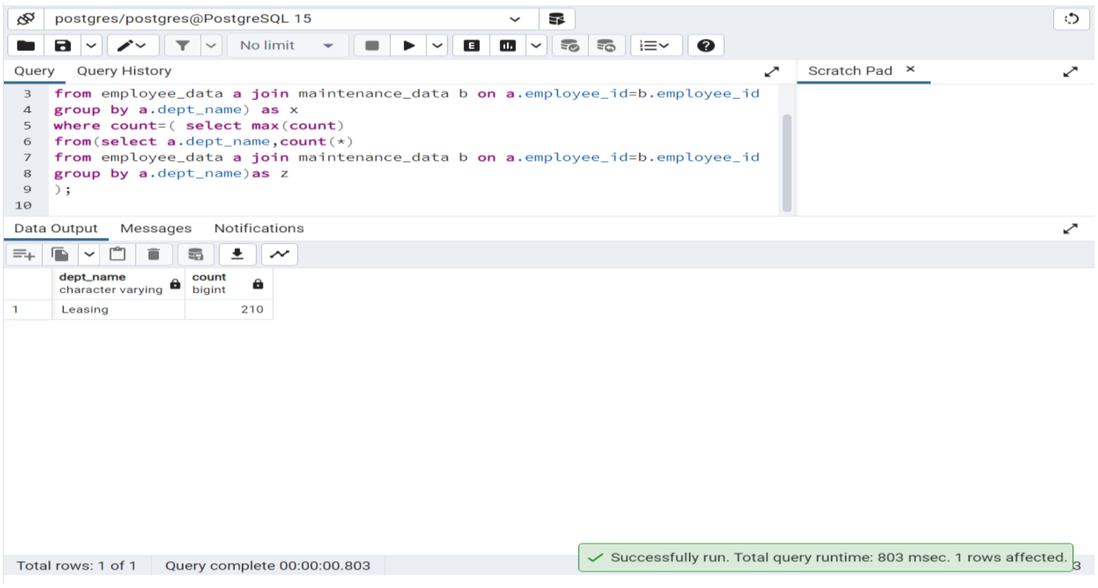
5. Which department has solved the greater number of problems of the customers?

Ans:

```
select*
from(select a.dept_name,count(*)
from employee_data a join maintenance_data b on a.employee_id=b.employee_id
group by a.dept_name) as x
where count=( select max(count)
from(select a.dept_name,count(*)
from employee_data a join maintenance_data b on a.employee_id=b.employee_id
group by a.dept_name)as z
);
```

Output:

No of rows returned 1



postgres/postgres@PostgreSQL 15

No limit

Query | Query History | Scratch Pad

```
from employee_data a join maintenance_data b on a.employee_id=b.employee_id
group by a.dept_name) as x
where count=( select max(count)
from(select a.dept_name,count(*)
from employee_data a join maintenance_data b on a.employee_id=b.employee_id
group by a.dept_name)as z
);
```

Data Output | Messages | Notifications

	dept_name character varying	count bigint
1	Leasing	210

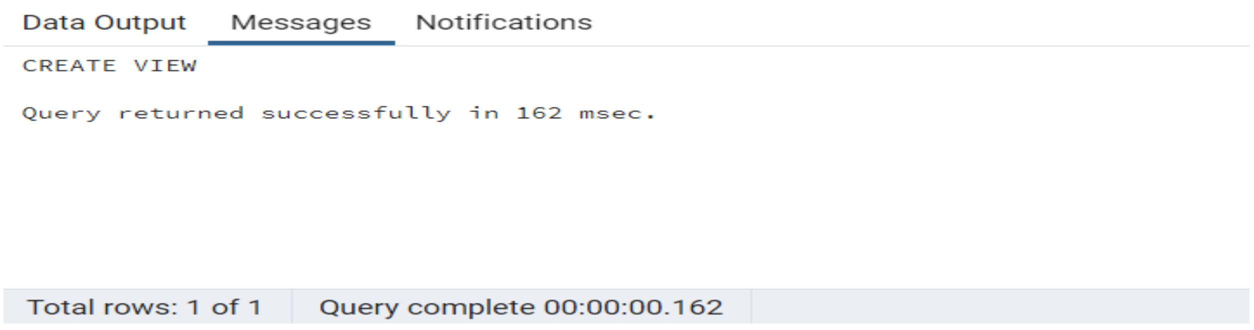
Total rows: 1 of 1 Query complete 00:00:00.803

✓ Successfully run. Total query runtime: 803 msec. 1 rows affected.

6. **Return the employee who solved more problems and has highest average rating?**

Ans:

```
Create view emp_prob_count AS
select employee_id,count(*)
from  maintenance_data
group by employee_id;
```

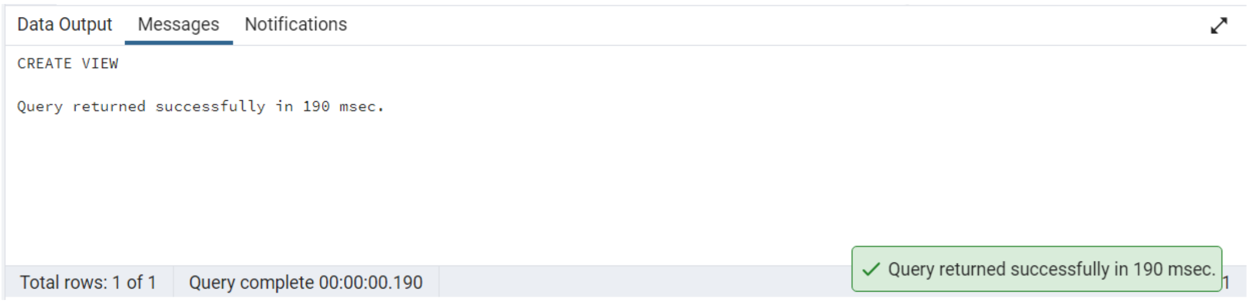


Data Output Messages Notifications

```
CREATE VIEW

Query returned successfully in 162 msec.
```

Total rows: 1 of 1 Query complete 00:00:00.162

```
Create view avg_rating_of_most_prob as
select employee_id, avg(rating)
from (select *
from maintenance_data
where employee_id in (select employee_id
from emp_prob_count
where count=(select max(count)
from emp_prob_count))) as x
group by employee_id;
```



```
CREATE VIEW

Query returned successfully in 190 msec.
```

Total rows: 1 of 1 Query complete 00:00:00.190

Query returned successfully in 190 msec.

select employee_id, avg as High_max_rating

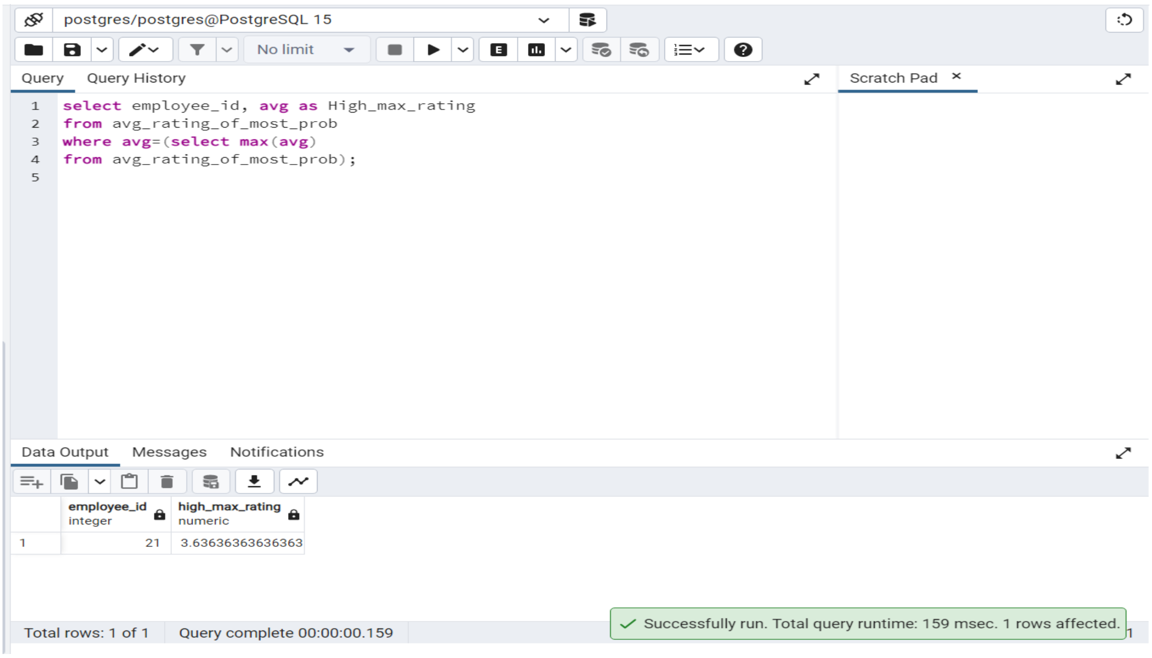
from avg_rating_of_most_prob

where avg=(select max(avg)

from avg_rating_of_most_prob);

OUTPUT:

No of rows returned 1



```
select employee_id, avg as High_max_rating
from avg_rating_of_most_prob
where avg=(select max(avg)
from avg_rating_of_most_prob);
```

	employee_id integer	high_max_rating numeric
1	21	3.63636363636363

Total rows: 1 of 1 Query complete 00:00:00.159

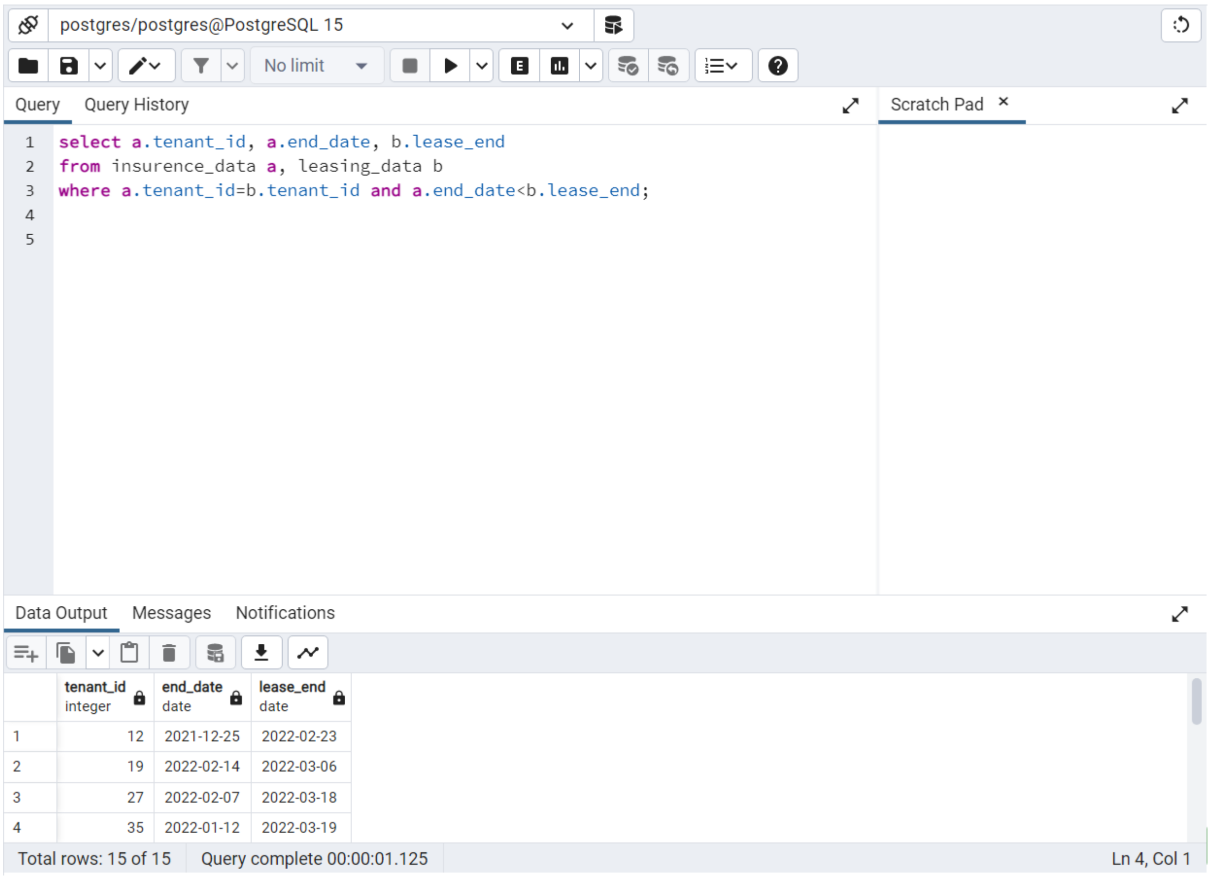
Successfully run. Total query runtime: 159 msec. 1 rows affected.

7. List the names of the customers whose insurance might expire before the lease end date.

```
select a.tenant_id, a.end_date, b.lease_end
from insurence_data a, leasing_data b
where a.tenant_id=b.tenant_id and a.end_date<b.lease_end;
```

<u>OUTPUT:</u>

No of rows returned 15



postgres/postgres@PostgreSQL 15

```
select a.tenant_id, a.end_date, b.lease_end
from insurence_data a, leasing_data b
where a.tenant_id=b.tenant_id and a.end_date<b.lease_end;
```

	tenant_id integer	end_date date	lease_end date
1	12	2021-12-25	2022-02-23
2	19	2022-02-14	2022-03-06
3	27	2022-02-07	2022-03-18
4	35	2022-01-12	2022-03-19

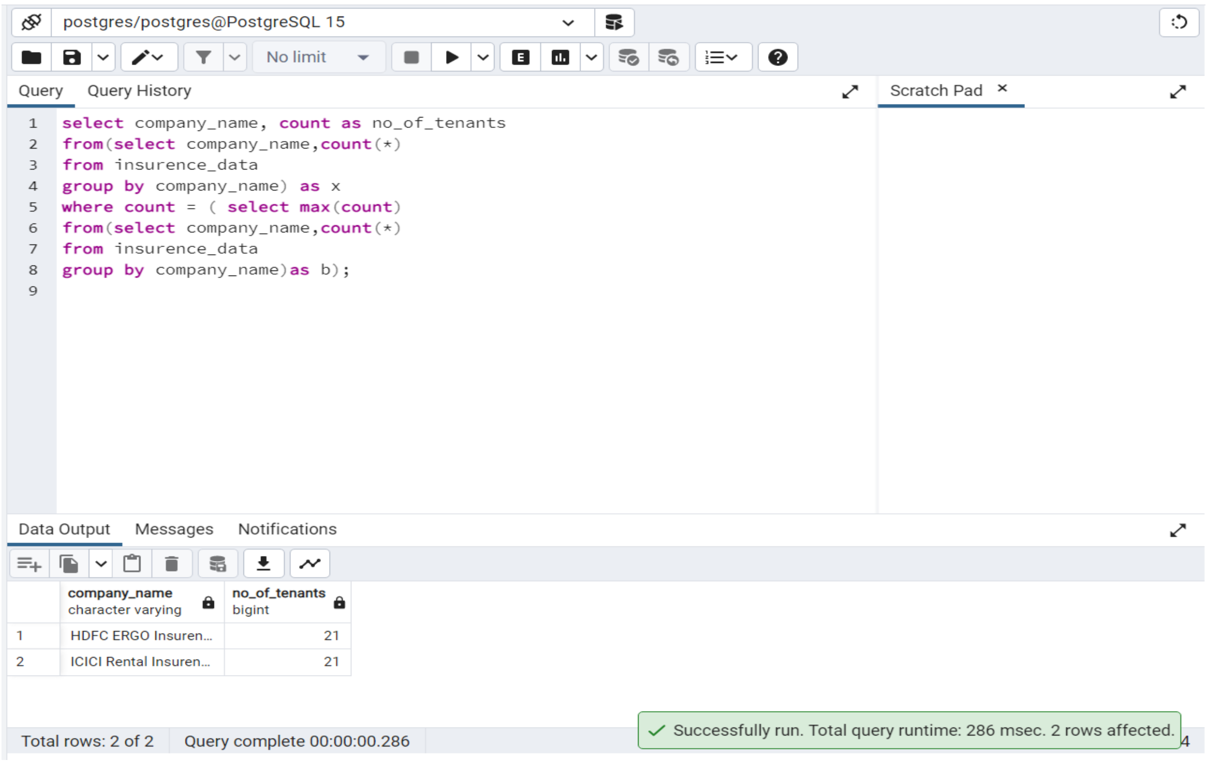
Total rows: 15 of 15   Query complete 00:00:01.125   Ln 4, Col 1

8.Which insurance company has the greater number of users registered.

```
select company_name, count as no_of_tenants
from(select company_name,count(*)
from insurence_data
group by company_name) as x
where count = ( select max(count)
from(select company_name,count(*)
from insurence_data
group by company_name)as b);
```

<u>Output:</u>

No of rows returned 2



postgres/postgres@PostgreSQL 15

```
select company_name, count as no_of_tenants
from(select company_name,count(*)
from insurence_data
group by company_name) as x
where count = ( select max(count)
from(select company_name,count(*)
from insurence_data
group by company_name)as b);
```

	company_name character varying	no_of_tenants bigint
1	HDFC ERGO Insuren...	21
2	ICICI Rental Insuren...	21

Total rows: 2 of 2 Query complete 00:00:00.286

Successfully run. Total query runtime: 286 msec. 2 rows affected.

9.

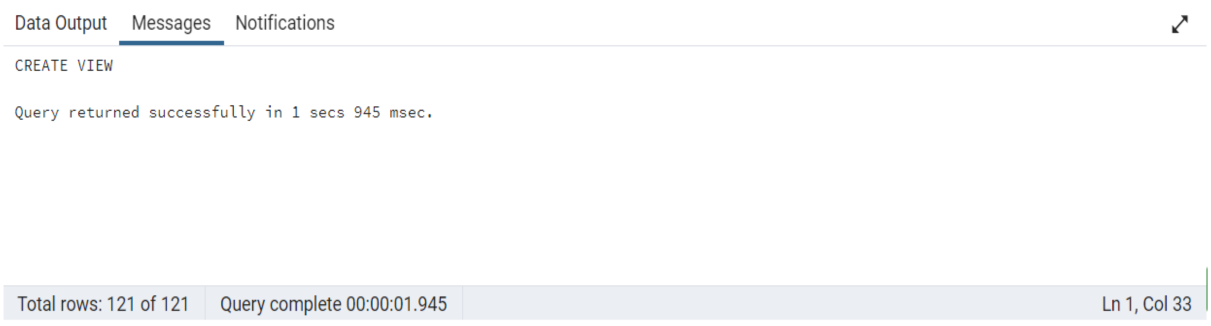
I. **Which maintenance problem has been faced by most of the customer in each department? (OLD QUESTION)**

II. **Which maintenance problem has been faced by most of the customers in all the department? (NEW QUESTION)**

Ans:

I had changed the question from each department to all the departments as I need to use the concept of partitions to divide the relationship over the department column, which is not yet covered, and I am not familiar with at present.

```
Create view count_issues_dept as

SELECT distinct a.dept_name,issue_name,COUNT(issue_name)

FROM employee_data a, maintenance_data b

WHERE a.employee_id = b.employee_id

GROUP BY dept_name,issue_name

order by dept_name, count desc;
```



```
CREATE VIEW

Query returned successfully in 1 secs 945 msec.
```

```
Total rows: 121 of 121    Query complete 00:00:01.945    Ln 1, Col 33
```

Select *

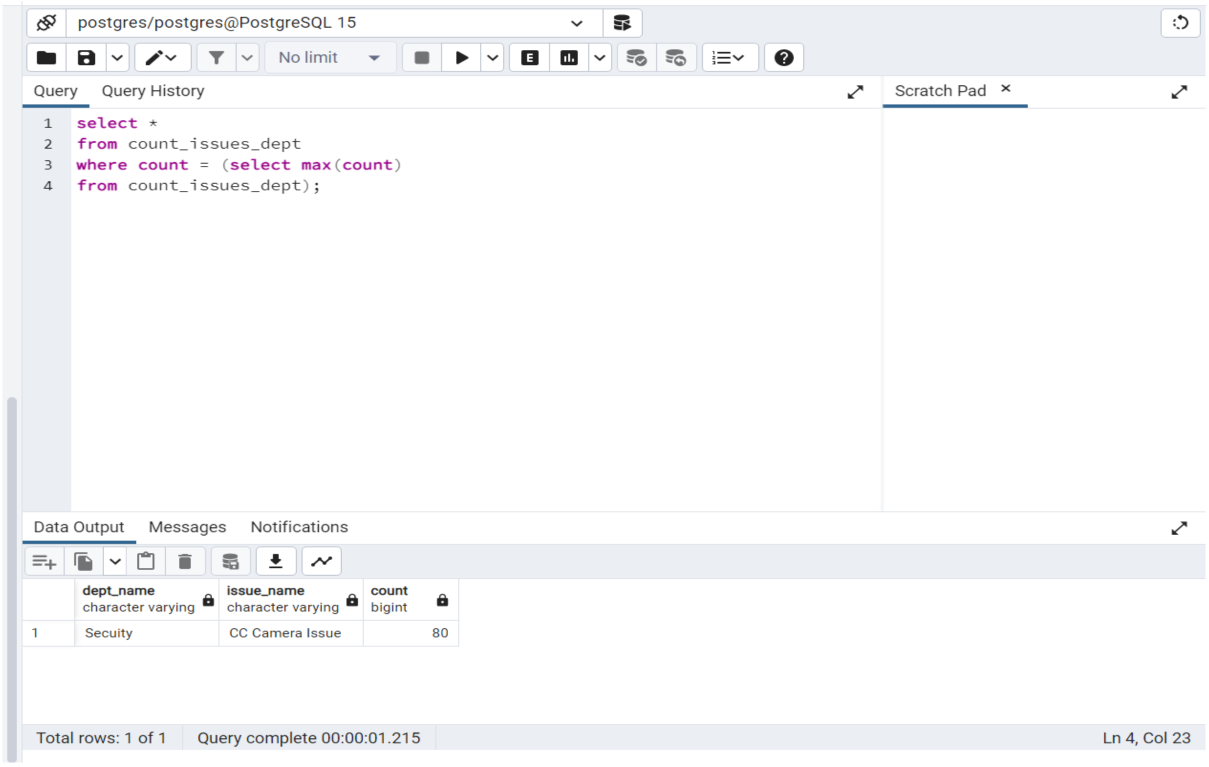
from count_issues_dept

where count = (select max(count)

from count_issues_dept);

OUTPUT:

No of rows returned 1



```
select *
from count_issues_dept
where count = (select max(count)
from count_issues_dept);
```

	dept_name character varying	issue_name character varying	count bigint
1	Secuity	CC Camera Issue	80

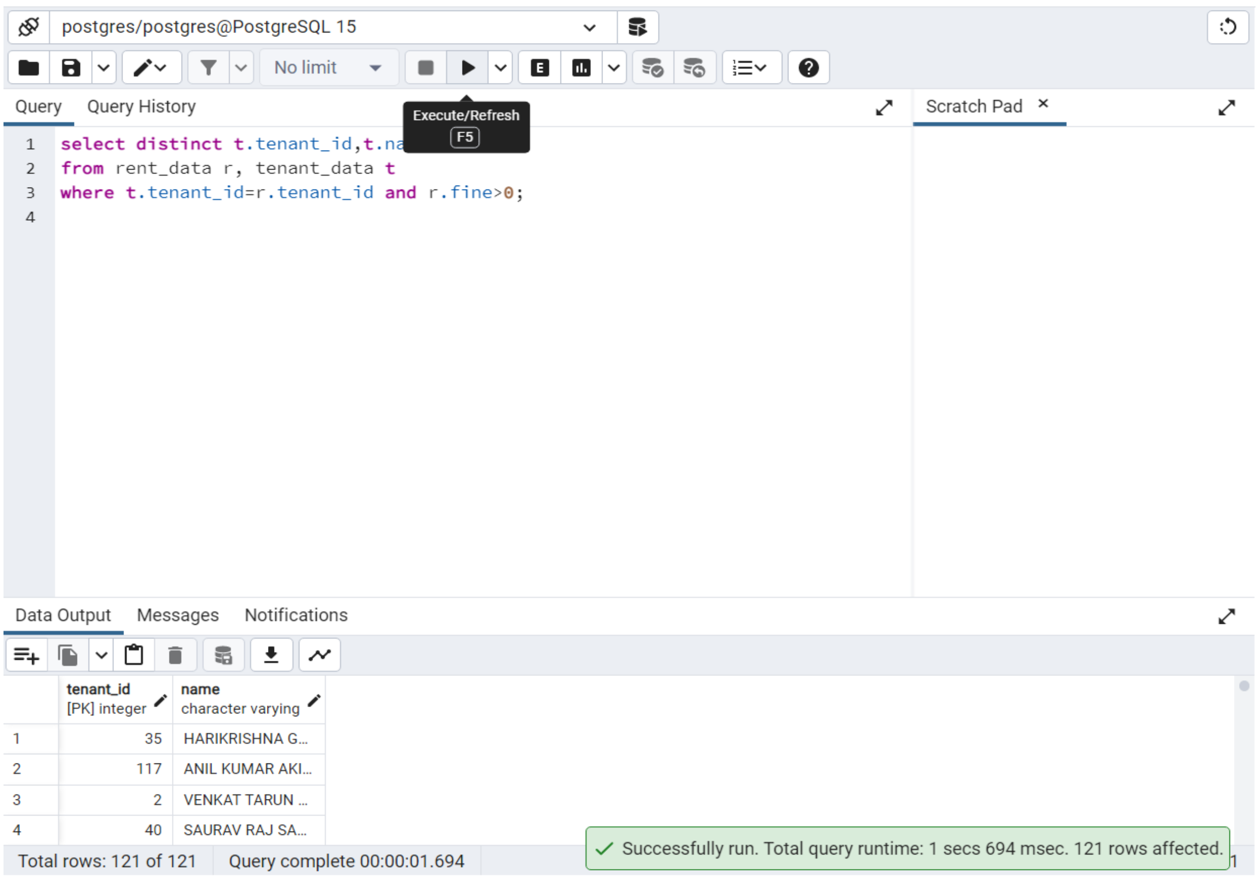
Total rows: 1 of 1   Query complete 00:00:01.215   Ln 4, Col 23

10. List the customers who paid the rent after the due date?

```
select distinct t.tenant_id,t.name

from rent_data r, tenant_data t

where t.tenant_id=r.tenant_id and r.fine>0;
```

<u>OUTPUT:</u>

No of rows returned 121



```
select distinct t.tenant_id,t.na
from rent_data r, tenant_data t
where t.tenant_id=r.tenant_id and r.fine>0;
```

	tenant_id [PK] integer	name character varying
1	35	HARIKRISHNA G...
2	117	ANIL KUMAR AKI...
3	2	VENKAT TARUN ...
4	40	SAURAV RAJ SA...

Total rows: 121 of 121 Query complete 00:00:01.694

✓ Successfully run. Total query runtime: 1 secs 694 msec. 121 rows affected.