PES UNIVERSITY
UE19CS301 Database Management Systems
Assignment – 4
Clothes Rental Service Database Management System

Team Members
1. Gayathri Sunil PES1UG19CS167
2. Jahnavi Sivaram PES1UG19CS192
3. Jigisha M Narrain PES1UG19CS197

Using psycopg2 to connect the frontend GUI to our PostgreSQL database-

- Familiarity with and ease of use of Python
- Straightforward and concise syntax
- Useful built-in methods
- Good error handling
- Supports all necessary SQL query types

Using tkinter as our GUI-

- Comes bundled with initial Python installation
- Efficient integration with psycopg2 statements and PostgreSQL back-end database
- Fast compared to other GUI toolkits
- Flexible and stable
- Simple and concise syntax
- Many customisation options to format interactive display window

Dependencies installed for database connectivity-

Our front-end has been implemented using the following libraries in Python, the reasons for which have been listed above-

- tkinter
- psycopg2
- PIL (for images)

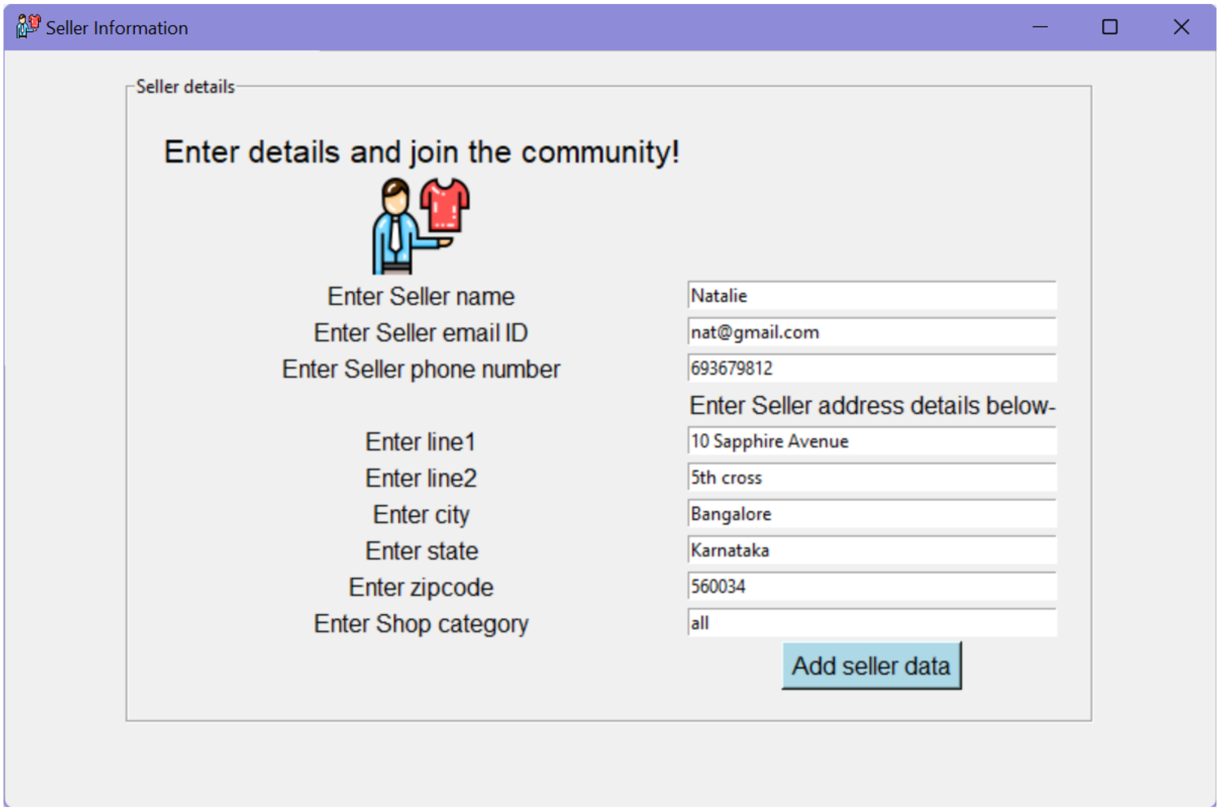
Some of the specific modules we have used for implementation are-

- From tkinter
  - Tk
  - title
  - iconbitmap
  - Label
  - pack
  - grid
- From psycopg2
  - connect

    - cursor
    - execute
    - fetchall
- From PIL
    - Image
    - ImageTk

Sample queries executed from front-end-

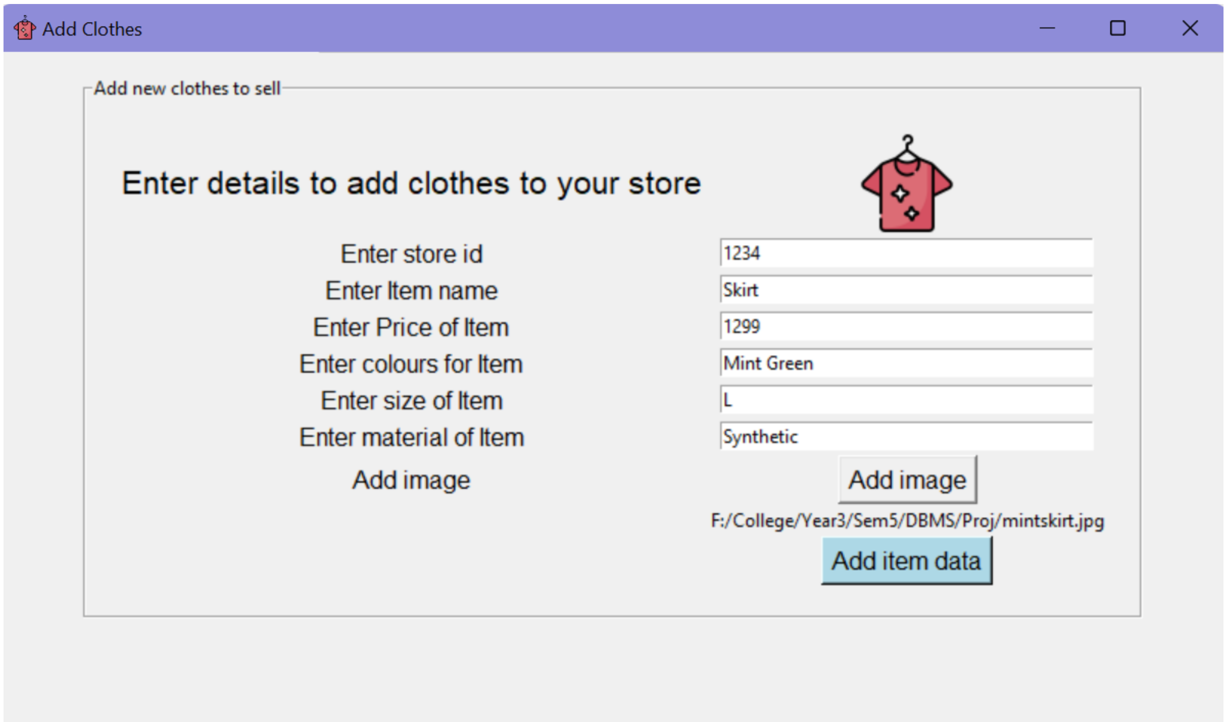
1. Adding seller information-



Enter details and join the community!

Enter Seller name	Natalie
Enter Seller email ID	nat@gmail.com
Enter Seller phone number	693679812
	Enter Seller address details below-
Enter line1	10 Sapphire Avenue
Enter line2	5th cross
Enter city	Bangalore
Enter state	Karnataka
Enter zipcode	560034
Enter Shop category	all

2. Adding new clothes to the store to be sold



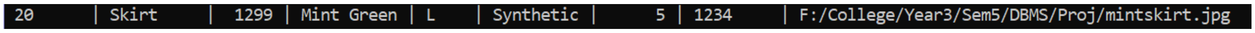
Add new clothes to sell

Enter details to add clothes to your store

Enter store id	1234
Enter Item name	Skirt
Enter Price of Item	1299
Enter colours for Item	Mint Green
Enter size of Item	L
Enter material of Item	Synthetic
Add image	Add image

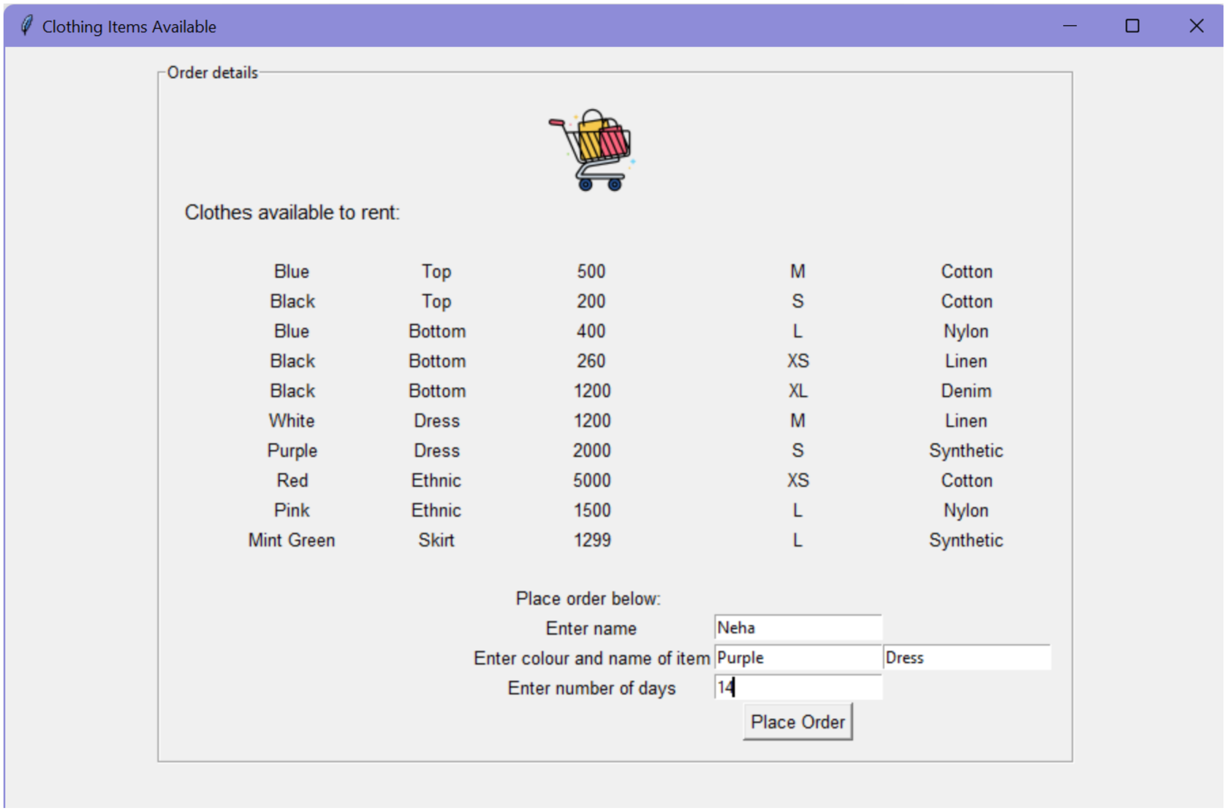
F:/College/Year3/Sem5/DBMS/Proj/mintskirt.jpg

Update in backend-



```
 20       | Skirt      |  1299 | Mint Green | L     | Synthetic |       5 | 1234      | F:/College/Year3/Sem5/DBMS/Proj/mintskirt.jpg
```

3. Viewing all available clothes and placing order-



Order details

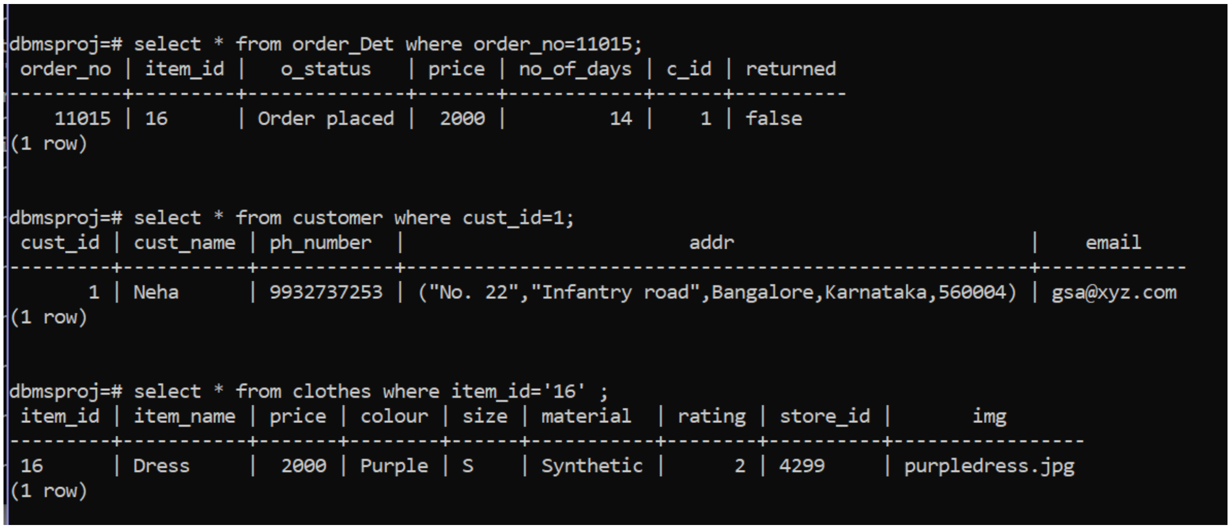
Clothes available to rent:

Blue	Top	500	M	Cotton
Black	Top	200	S	Cotton
Blue	Bottom	400	L	Nylon
Black	Bottom	260	XS	Linen
Black	Bottom	1200	XL	Denim
White	Dress	1200	M	Linen
Purple	Dress	2000	S	Synthetic
Red	Ethnic	5000	XS	Cotton
Pink	Ethnic	1500	L	Nylon
Mint Green	Skirt	1299	L	Synthetic

Place order below:

Enter name	Neha	
Enter colour and name of item	Purple	Dress
Enter number of days	14	

The following updates were made in the backend-

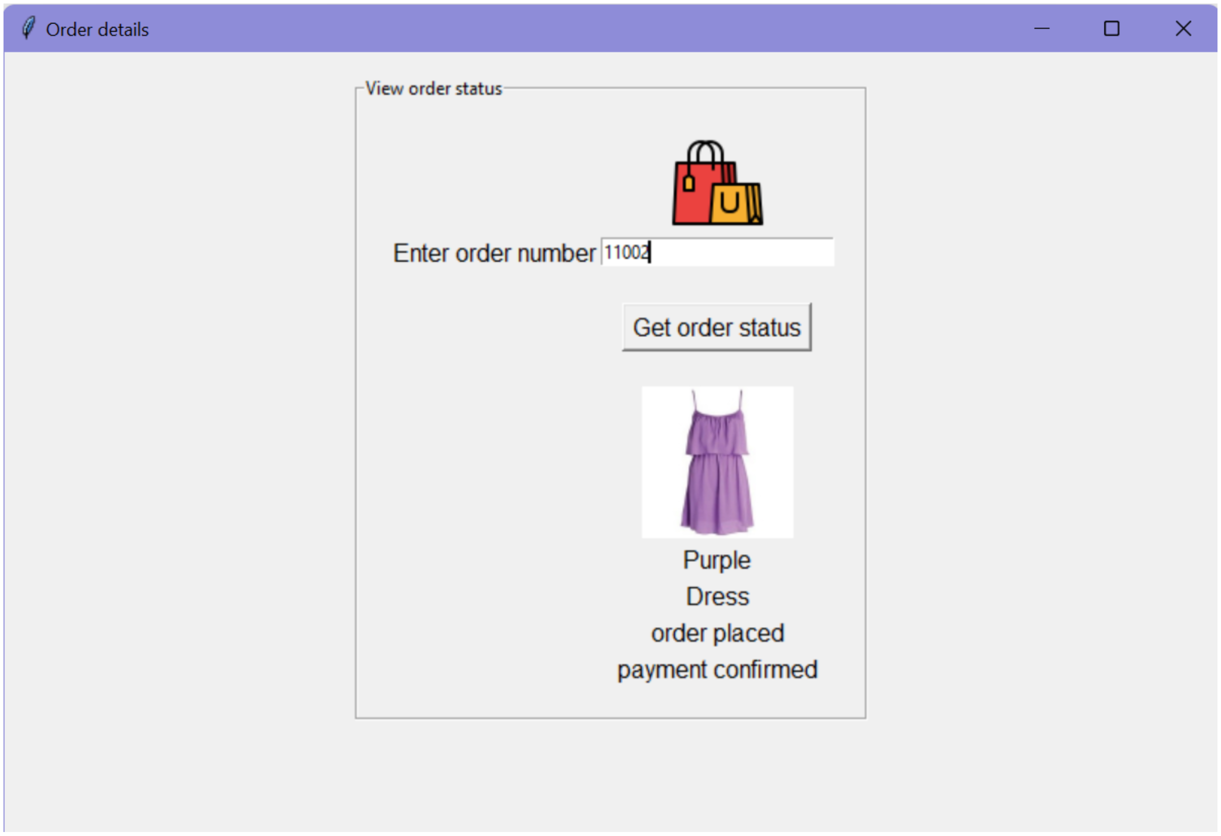


```
dbmsproj=# select * from order_Det where order_no=11015;
 order_no | item_id |   o_status    | price | no_of_days | c_id | returned
----------+---------+---------------+-------+------------+------+----------
    11015 | 16      | Order placed |  2000 |         14 |    1 | false
(1 row)


dbmsproj=# select * from customer where cust_id=1;
 cust_id | cust_name | ph_number  |                         addr                          |    email
---------+-----------+------------+-------------------------------------------------------+-------------
       1 | Neha      | 9932737253 | ("No. 22","Infantry road",Bangalore,Karnataka,560004) | gsa@xyz.com
(1 row)


dbmsproj=# select * from clothes where item_id='16' ;
 item_id | item_name | price | colour | size | material  | rating | store_id |      img
---------+-----------+-------+--------+------+-----------+--------+----------+----------------
 16      | Dress     |  2000 | Purple | S    | Synthetic |      2 | 4299     | purpledress.jpg
(1 row)
```

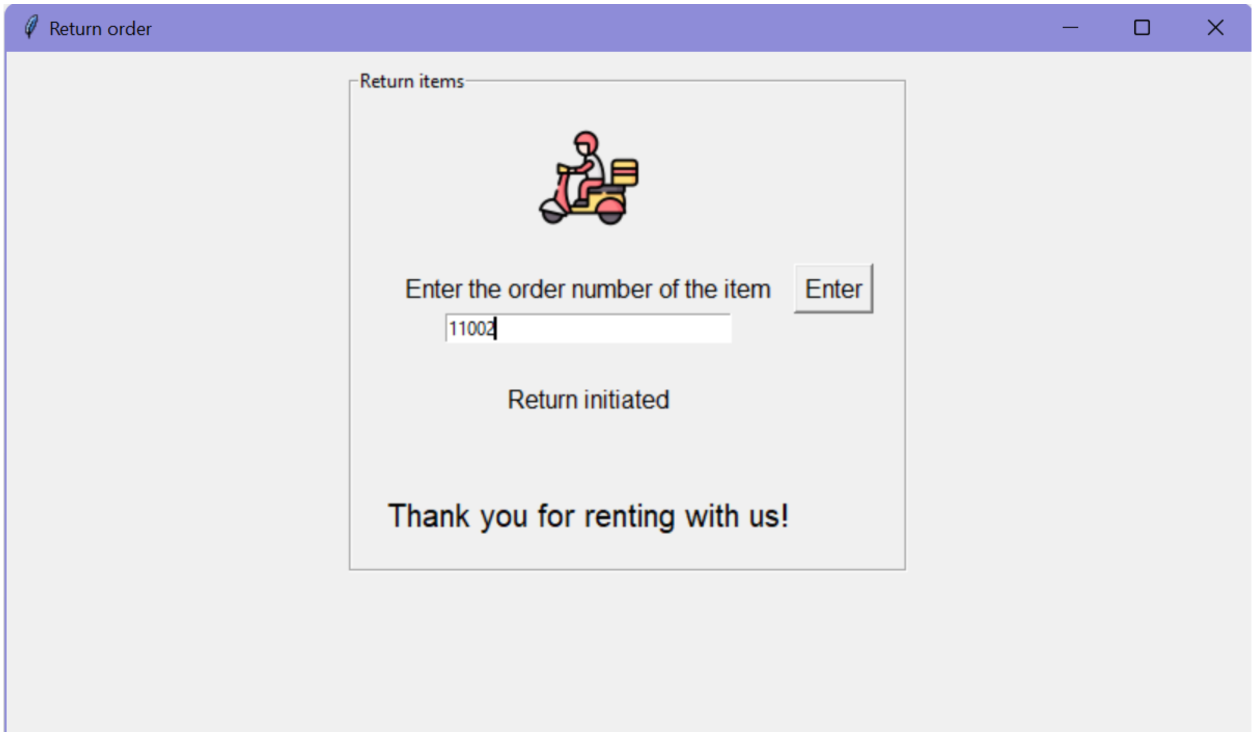
4. Getting details about order placed-



Enter order number 11002

Purple
Dress
order placed
payment confirmed

5. Returning the order



Enter the order number of the item

11002

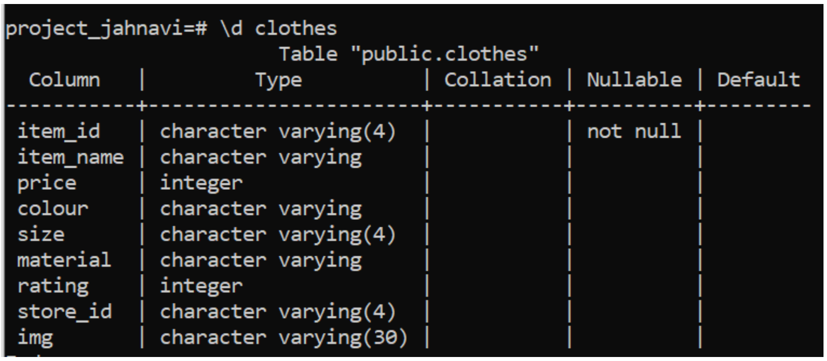
Return initiated

Thank you for renting with us!

Schema change queries-

1. *Adding column img to table clothes-*

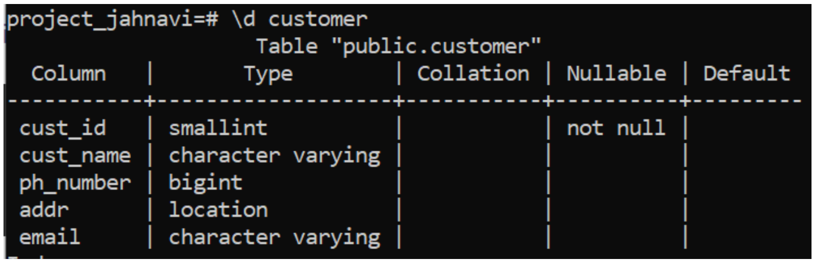
```
alter table clothes
add column img varchar(30);
```



```
project_jahnavi=# \d clothes
                       Table "public.clothes"
  Column   |         Type          | Collation | Nullable | Default
-----------+-----------------------+-----------+----------+---------
 item_id   | character varying(4)  |           | not null |
 item_name | character varying     |           |          |
 price     | integer               |           |          |
 colour    | character varying     |           |          |
 size      | character varying(4)  |           |          |
 material  | character varying     |           |          |
 rating    | integer               |           |          |
 store_id  | character varying(4)  |           |          |
 img       | character varying(30) |           |          |
```

2. *Changing the data type of cust_id in table customer from int to smallint-*

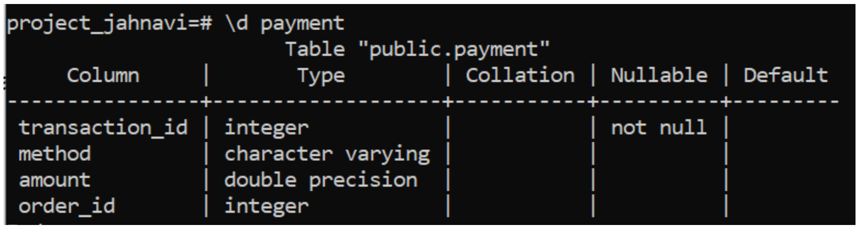
```
alter table customer
alter column cust_id type smallint;
```



```
project_jahnavi=# \d customer
                    Table "public.customer"
  Column   |       Type        | Collation | Nullable | Default
-----------+-------------------+-----------+----------+---------
 cust_id   | smallint          |           | not null |
 cust_name | character varying |           |          |
 ph_number | bigint            |           |          |
 addr      | location          |           |          |
 email     | character varying |           |          |
```

3. *Dropping p_status column from table payment-*

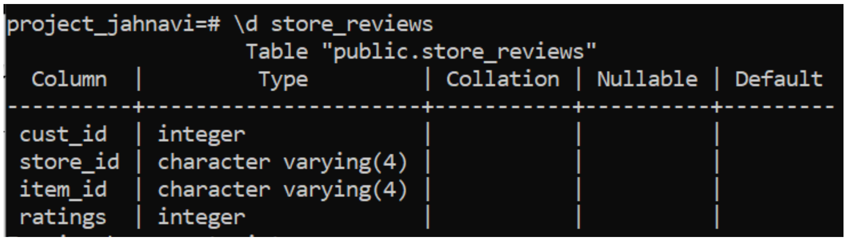
```
alter table payment
drop column p_status;
```



```
project_jahnavi=# \d payment
                      Table "public.payment"
     Column     |       Type        | Collation | Nullable | Default
----------------+-------------------+-----------+----------+---------
 transaction_id | integer           |           | not null |
 method         | character varying |           |          |
 amount         | double precision  |           |          |
 order_id       | integer           |           |          |
```

4. *Changing name of rat column to ratings in table store_reviews-*

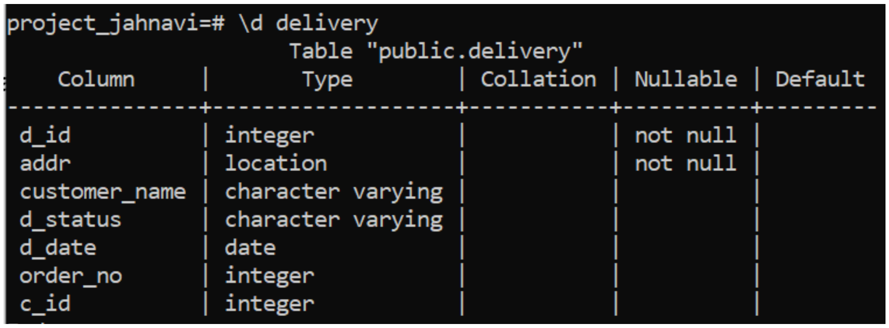
```
alter table store_reviews
rename column rat
to ratings;
```



```
project_jahnavi=# \d store_reviews
                   Table "public.store_reviews"
  Column  |         Type         | Collation | Nullable | Default
----------+----------------------+-----------+----------+---------
 cust_id  | integer              |           |          |
 store_id | character varying(4) |           |          |
 item_id  | character varying(4) |           |          |
 ratings  | integer              |           |          |
```

5. *Setting NOT NULL constraint to column addr in table delivery-*

```
alter table delivery
alter column addr
set not null;
```



```
project_jahnavi=# \d delivery
                      Table "public.delivery"
    Column     |       Type        | Collation | Nullable | Default
---------------+-------------------+-----------+----------+---------
 d_id          | integer           |           | not null |
 addr          | location          |           | not null |
 customer_name | character varying |           |          |
 d_status      | character varying |           |          |
 d_date        | date              |           |          |
 order_no      | integer           |           |          |
 c_id          | integer           |           |          |
```

Changes in business model that could lead to

- **Schema changes**
  Addition of ___ column in table Order.
  Addition of img column to table Clothes.
  Redefined attribute data-types.

- **Migration**
  1) Not easy to import complex JSON to DB.
  2) Works slowly with storage of big raw video/audio in SQL tables.
  3) Cannot handle large volumes of data, makes an impact on speed.
  4) Hardware resource consumption is high; memory management needs to be optimized.
  5) Query processing gets slow if there is a large number of records.
  6) PostgreSQL faces scalability issues.

Database Migration: MongoDB (NoSQL)-

Why should we use MongoDB

- MongoDB is a document oriented database that enables you to manage data of any structure, not just tabular structures defined in advance and developers can reshape the data on their own.
- Scaling in terms of volume of traffic or size of data (or both) needs to be distributed across regions can be handled by MongoDB’s scale-out architecture.
- Multi-cloud database that works the same way in every public cloud, can store customer data in specific geographic regions, and support the latest serverless and mobile development paradigms.
- Query performance in MongoDB can be accelerated by creating indexes on fields in documents and subdocuments. MongoDB allows any field of a document, including those deeply nested in arrays and subdocuments, to be indexed and efficiently queried.

- The downside of PostgreSQL compared to MongoDB is that it relies on relational data models that are unfriendly to the data structures developers work with in code, and that must be defined in advance, slowing progress whenever requirements change.
- In MongoDB scalability is built-in through native sharding, enabling a horizontal scale-out approach. MongoDB Atlas has a broad multi-cloud, globally aware platform at the ready, all fully managed for you.
- PostgreSQL uses a scale-up strategy. This means that at some point, for high performance use cases, you may hit a wall or have to divert resources to finding other ways to scale via caching or denormalizing data or using other strategies.
- PostgreSQL can support replication but more advanced features such as automatic failover must be supported by third-party products developed independently of the database. Such an approach is more complex and can work slower and less seamlessly than MongoDB’s in-built self-healing capabilities.

Steps to migrate from PostgreSQL to MongoDB

1. To migrate data, you’ll extract it from PostgreSQL and then import it to MongoDB using the mongoimport tool. Two different ways to extract the data are: returning queries as tab-separated values (TSV) or as JSON.

2. Prepare your application for connecting to MongoDB., MongoDB has support for all of the major programming languages as well as many popular frameworks.

3. Consider the schema changes that would be best for your data, while keeping in mind MongoDB schema best practices and avoiding anti-patterns.

4. Export the data from your PostgreSQL databases by piping the result of an SQL query into a COPY command, outputting the result either as JSON or TSV.

5. Restructure the data to fit your MongoDB schema by using mongoimport (or as an alternative: use bulkWrite operations to load the data).

Contribution-

- GUI implementation, Schema change queries – **5 hrs** Gayathri Sunil

- Connection of PostgreSQL to psycopg2, Report compilation – **5 hrs** Jahnavi Sivaram

- Modifying queries for front-end, Migration Write-up – **5 hrs** Jigisha Narrain