DBMS Mini Project

Title of the Project: Gadget Store Management System

Submitted by

Name: Harshit Timmanagoudar

SRN: PES1UG20CS161

V Semester Section C

Abstract

Today’s world is moving fast with technology in gadgets. Everyday there is new upgrade in gadgets and people are keen on grabbing these and exploring the features. It is always easier to provide a platform where in the management can keep a track and upgrade the existing gadgets in the database, and also to maintain a record of the purchases made by a particular customer for different purposes.

In the mini-project that I have chosen, I tried to implement the above mentioned. The tools required for the building the project:

1. PostgreSQL : Database
2. pgadmin 4 : User interface
3. Flask : Developing the frontend and connecting to the database.

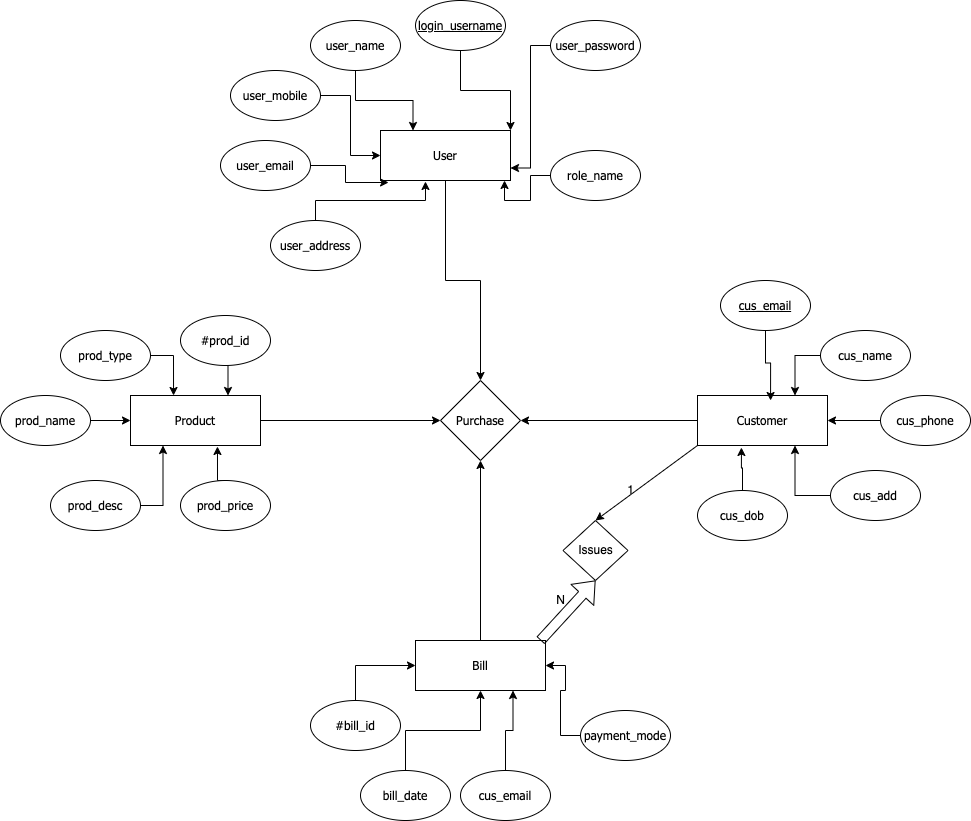
Four entities have been considered for the project namely, ‘users’, ‘customer’, ‘product’ and ‘bill’.

1. The ‘users’ entity stores all the information about the employees working for the gadget store. There is no way to create a new record for an employee through the frontend nevertheless, it can be done through the UI (pgadmin) or terminal. A particular user of the gadget store must login with his/her login username (login\_username) and login password (login\_password) before selling any product to the customer, creating or updating or deleting information related to a particular customer and for other reasons. The login username corresponding to each employee is used as primary key for the relation.
2. The ‘customer’ entity stores all the information about the customers who have purchased product(s) from the store. A customer record can be created, retrieved, updated and deleted either using the terminal (or User interface) or the frontend. The customer email (cus\_email) is used to uniquely identify each customer and is thus used as the primary key for this relation.
3. The ‘product’ entity stores all the information about the products available in the store. The product Id (prod\_id) is used as the primary key for this relation.
4. The ‘bill’ entity stores the information regarding about the purchase made by a particular customer. The relation possesses information regarding when the purchase was made and in which mode of payment the purchase was completed. This relation is automatically updated each time a purchase is made by a user. This feature of the project is handled by flask. Bill Id (bill\_id) is used as a primary key for this relation.

The ’purchase’ relation in the database reflects information regarding a user who sold a particular product to a customer with the bill id.

More information on project such as the ER diagram, Schema diagram, DDL statements used for creating the relations in the database, different ways of populating the different relations of the database, trigger, view and code required for developing the frontend and connecting it to the database is mentioned further in the document.

ER Diagram



Relational Schema

Graphical user interface, application

Description automatically generated

DDL Statements – Building the database

users

CREATE TABLE users(

login\_username VARCHAR(20) NOT NULL UNIQUE,

user\_name VARCHAR(50) NOT NULL,

user\_email VARCHAR(25) NOT NULL UNIQUE,

user\_mobile VARCHAR(13) NOT NULL UNIQUE,

user\_address VARCHAR(100),

login\_password VARCHAR(20) NOT NULL,

role\_name VARCHAR(20) NOT NULL,

PRIMARY KEY(login\_username)

);

customer

CREATE TABLE customer(

cus\_email VARCHAR(25) NOT NULL UNIQUE,

cus\_name VARCHAR(50) NOT NULL,

cus\_phone VARCHAR(13) NOT NULL,

cus\_add VARCHAR(100),

cus\_dob date,

PRIMARY KEY(cus\_email)

);

bill

CREATE TABLE bill(

bill\_id SERIAL PRIMARY KEY,

bill\_date date NOT NULL,

cus\_email VARCHAR(25) NOT NULL,

payment\_mode VARCHAR(30) NOT NULL

);

product

CREATE TABLE product(

prod\_id SERIAL PRIMARY KEY,

prod\_type VARCHAR(20) NOT NULL,

prod\_name VARCHAR(50) NOT NULL,

prod\_storage VARCHAR(10) NOT NULL,

prod\_price VARCHAR(10) NOT NULL

);

purchase

CREATE TABLE purchase(

username VARCHAR(20) NOT NULL,

cus\_email VARCHAR(25) NOT NULL,

bill\_id VARCHAR(15) NOT NULL,

prod\_id INT NOT NULL,

FOREIGN KEY(username) REFERENCES users(login\_username),

FOREIGN KEY(cus\_email) REFERENCES customer(cus\_email),

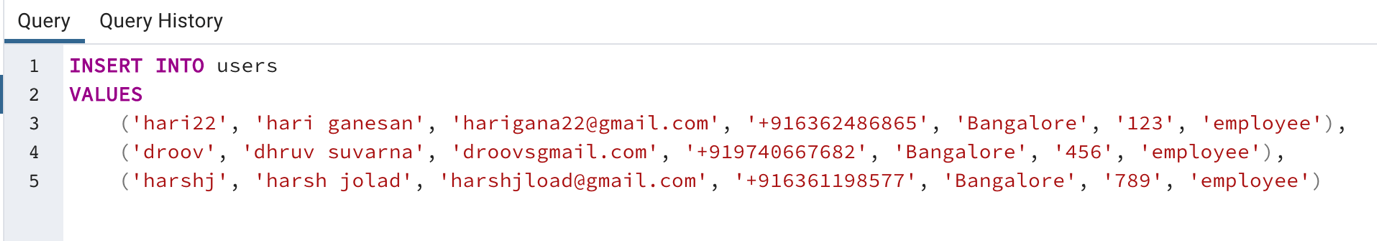
FOREIGN KEY(bill\_id) REFERENCES bill(bill\_id),

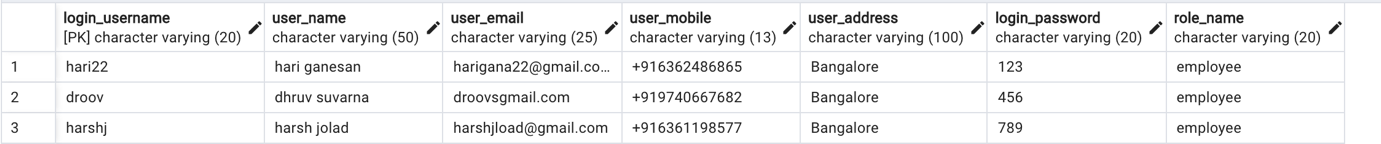
FOREIGN KEY(prod\_id) REFERENCES product(prod\_id)

);

Populating the Database

1. Populating ‘users’ table through pgadmin.





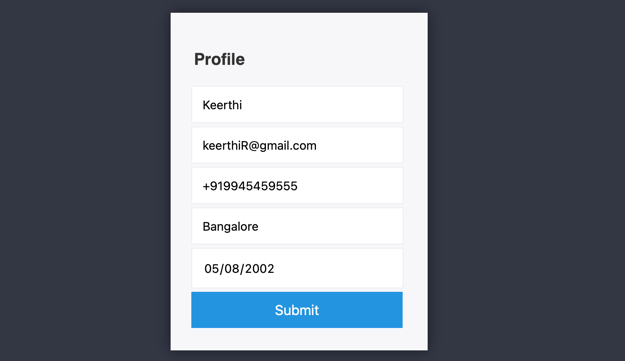
1. Populating ‘customers’ table through frontend.

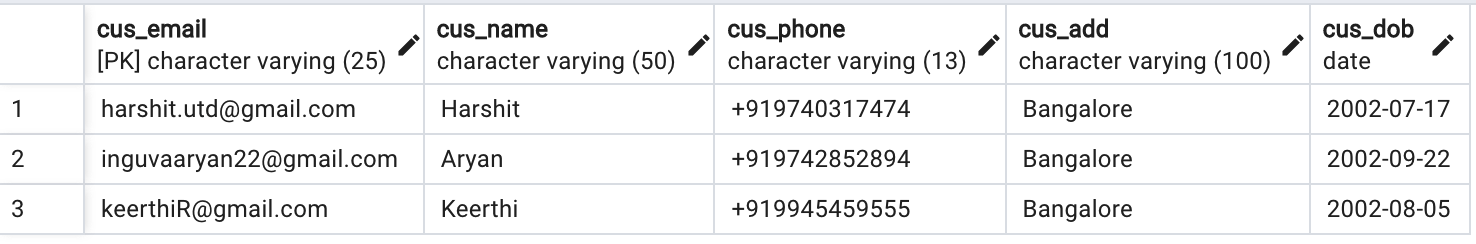
Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated





1. Populating ‘product’ table

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Table

Description automatically generated

1. Relations ‘bill’ and ‘purchase’ are populated once the purchase is made by the customer.

Purchase process:

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface

Description automatically generated

Opting ‘Add item’

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Opting ‘Check Out’:

Graphical user interface, application

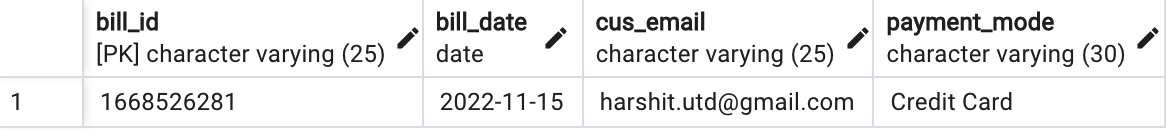
Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

Database after purchase:

‘bill’ relation



‘purchase’ relation

Graphical user interface, application, Teams

Description automatically generated

Triggers

Function creation:

Graphical user interface, text, application

Description automatically generated

Trigger creation:

Graphical user interface, text

Description automatically generated with medium confidence

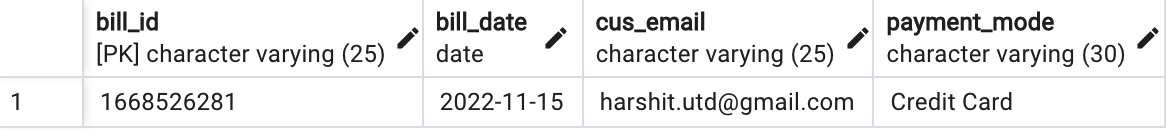
Before delete query:

‘customer’ relation

Table

Description automatically generated

‘bill’ relation



‘purchase’ relation

Graphical user interface, application, Teams

Description automatically generated

Delete query:

Graphical user interface, text, application

Description automatically generated

After delete query:

‘customer’ relation

Table

Description automatically generated

‘bill’ relation

Graphical user interface, application, table

Description automatically generated

‘purchase’ relation

Graphical user interface, text, application

Description automatically generated

Developing a Frontend

Tools used: Python libraries

* Flask
* Psycopg2

import psycopg2 as pg2

from flask import Flask, render\_template, request, redirect

import datetime

app = Flask(\_\_name\_\_)

DATABASE = 'Project'

USER = 'postgres'

PASSWORD = 'Xbox1xfifa19@postgres'

DATA = {}

def get\_cursor():

conn = pg2.connect(database=DATABASE, user=USER, password=PASSWORD)

cur = conn.cursor()

return conn, cur

def close\_connection(conn):

conn.close()

@app.route('/', methods=['GET', 'POST'])

def index():

if request.method == 'GET':

return render\_template('index.html')

elif request.method == 'POST':

login\_username = request.form.get('username')

login\_password = request.form.get('password')

try:

conn, cur = get\_cursor()

cur.execute("SELECT login\_username FROM users")

usernames = cur.fetchall()

flag0 = 0

for username in usernames:

if login\_username == username[0]:

flag0 = 1

break

if flag0 == 0:

#TODO

close\_connection(conn)

return render\_template('index.html')

else:

cur.execute("SELECT login\_password FROM users WHERE login\_username = '{}'".format(login\_username))

password = cur.fetchall()[0][0]

close\_connection(conn)

if password != login\_password:

#TODO

return render\_template('index.html')

else:

DATA['user'] = login\_username

return render\_template('options.html', username=login\_username)

except:

close\_connection(conn)

return redirect('/')

@app.route('/options', methods=['GET', 'POST'])

def options():

if request.method == 'GET':

return render\_template('options.html', username=DATA['user'])

elif request.method == 'POST':

return redirect('/options')

@app.route('/validate\_customer', methods=['GET', 'POST'])

def validate\_customers():

if request.method == 'GET':

return render\_template('validate\_customer.html', username=DATA['user'])

elif request.method == 'POST':

cus\_email = request.form.get('cus\_email')

try:

conn, cur = get\_cursor()

cur.execute("SELECT \* FROM customer WHERE cus\_email = '{}'".format(cus\_email))

customer = cur.fetchall()[0]

if not customer:

close\_connection(conn)

return redirect('/validate')

else:

DATA['customer'] = customer

close\_connection(conn)

return redirect('/bill')

except:

close\_connection(conn)

return redirect('/validate\_customer')

@app.route('/create\_customer', methods = ['GET', 'POST'])

def create\_customer():

if request.method == 'GET':

return render\_template('create\_customer.html', username=DATA['user'])

elif request.method == 'POST':

cus\_name = request.form.get('cus\_name')

cus\_email = request.form.get('cus\_email')

cus\_phone = request.form.get('cus\_phone')

cus\_add = request.form.get('cus\_add')

cus\_dob = request.form.get('cus\_dob')

try:

conn, cur = get\_cursor()

postgres\_insert\_query = """ INSERT INTO customer(cus\_email, cus\_name, cus\_phone, cus\_add, cus\_dob) VALUES (%s, %s, %s, %s, %s)"""

values = (cus\_email, cus\_name, cus\_phone, cus\_add, cus\_dob)

cur.execute(postgres\_insert\_query, values)

conn.commit()

cur.execute("SELECT \* FROM customer WHERE cus\_email = '{}'".format(cus\_email))

customer = cur.fetchall()[0]

DATA['customer'] = customer

close\_connection(conn)

return redirect('/bill')

except:

close\_connection(conn)

return redirect('/create\_customer')

@app.route('/update\_customer', methods=['GET', 'POST'])

def update\_customer():

if request.method == 'GET':

return render\_template('update\_customer.html', username=DATA['user'])

elif request.method == 'POST':

cus\_email = request.form.get('cus\_email')

return redirect("/update\_customer\_details?email={}".format(cus\_email))

@app.route('/update\_customer\_details', methods=['GET', 'POST'])

def update\_customer\_details():

if request.method == 'GET':

try:

conn, cur = get\_cursor()

cur.execute("SELECT \* FROM customer WHERE cus\_email = '{}'".format(request.args.get('email')))

customer\_details = cur.fetchall()[0]

cus\_email = customer\_details[0]

cus\_name = customer\_details[1]

cus\_phone = customer\_details[2]

cus\_add = customer\_details[3]

cus\_dob = customer\_details[4]

close\_connection(conn)

return render\_template('update\_customer\_details.html',

cus\_email=cus\_email,

cus\_name=cus\_name,

cus\_phone=cus\_phone,

cus\_add=cus\_add,

cus\_dob=cus\_dob,

username=DATA['user'])

except:

close\_connection(conn)

return render\_template('update\_customer.html')

elif request.method == 'POST':

cus\_name = request.form.get('cus\_name')

cus\_email = request.form.get('cus\_email')

cus\_phone = request.form.get('cus\_phone')

cus\_add = request.form.get('cus\_add')

cus\_dob = request.form.get('cus\_dob')

try:

conn, cur = get\_cursor()

postgres\_update\_query = """ UPDATE customer

SET cus\_name = %s,

cus\_phone = %s,

cus\_add = %s,

cus\_dob = %s

WHERE cus\_email = %s"""

cur.execute(postgres\_update\_query, (cus\_name, cus\_phone, cus\_add, cus\_dob, cus\_email))

conn.commit()

cur.execute("SELECT \* FROM customer WHERE cus\_email = '{}'".format(cus\_email))

customer = cur.fetchall()[0]

DATA['customer'] = customer

close\_connection(conn)

return redirect('/bill')

except:

close\_connection(conn)

return render\_template('update\_customer.html')

@app.route('/delete\_customer', methods=['GET', 'POST'])

def delete\_customer():

if request.method == 'GET':

return render\_template('delete\_customer.html', username=DATA['user'])

elif request.method == 'POST':

cus\_email = request.form.get('cus\_email')

try:

conn, cur = get\_cursor()

cur.execute("DELETE FROM customer WHERE cus\_email = '{}'".format(cus\_email))

conn.commit()

close\_connection(conn)

return render\_template('options.html', username=DATA['user'])

except:

close\_connection(conn)

return redirect('/delete\_customer')

@app.route('/add\_product', methods=['GET', 'POST'])

def add\_product():

if request.method == 'GET':

return render\_template('add\_product.html', username=DATA['user'])

elif request.method == 'POST':

prod\_type = request.form.get('prod\_type')

prod\_name = request.form.get('prod\_name')

prod\_storage = request.form.get('prod\_storage')

prod\_price = request.form.get('prod\_price')

try:

conn, cur = get\_cursor()

cur.execute("SELECT \* FROM product WHERE prod\_type = '{}' AND prod\_name = '{}' AND prod\_storage = '{}' AND prod\_price = '{}'".format(prod\_type, prod\_name, prod\_storage, prod\_price))

product = cur.fetchall()

if product:

close\_connection(conn)

return render\_template('options.html', username=DATA['user'])

postgres\_insert\_query = """ INSERT INTO product(prod\_type, prod\_name, prod\_storage, prod\_price) VALUES (%s, %s, %s, %s)"""

values = (prod\_type, prod\_name, prod\_storage, prod\_price)

cur.execute(postgres\_insert\_query, values)

conn.commit()

close\_connection(conn)

return render\_template('options.html', username=DATA['user'])

except:

close\_connection(conn)

return redirect('/add\_product')

@app.route('/delete\_product', methods=['GET', 'POST'])

def delete\_product():

if request.method == 'GET':

try:

conn, cur = get\_cursor()

cur.execute("SELECT DISTINCT(prod\_type) FROM product")

pt\_data = cur.fetchall()

product\_types = []

for product\_type in pt\_data:

product\_types.append(product\_type[0])

close\_connection(conn)

return render\_template('delete\_product.html', product\_types=product\_types, username=DATA['user'])

except:

close\_connection(conn)

return redirect('/delete\_product')

elif request.method == 'POST':

prod\_type = request.form.get('prod\_type')

print(prod\_type)

return redirect('/delete\_product\_type?product\_type={}'.format(prod\_type))

@app.route('/delete\_product\_type', methods=['GET', 'POST'])

def delete\_product\_type():

if request.method == 'GET':

prod\_type = request.args.get('product\_type')

try:

conn, cur = get\_cursor()

cur.execute("SELECT \* FROM product WHERE prod\_type = '{}'".format(prod\_type))

products = cur.fetchall()

close\_connection(conn)

return render\_template('delete\_product\_type.html', products=products, username=DATA['user'])

except:

close\_connection(conn)

return redirect('/delete\_product')

elif request.method == 'POST':

prod\_id = request.form.get('prod\_id')

try:

conn, cur = get\_cursor()

cur.execute("DELETE FROM product WHERE prod\_id = {}".format(prod\_id))

conn.commit()

close\_connection(conn)

return render\_template('options.html', username=DATA['user'])

except:

close\_connection(conn)

return redirect('/delete\_product')

@app.route('/products', methods=['GET', 'POST'])

def products():

if request.method == 'GET':

try:

conn, cur = get\_cursor()

cur.execute("SELECT DISTINCT(prod\_type) FROM product")

pt\_data = cur.fetchall()

product\_types = []

for product\_type in pt\_data:

product\_types.append(product\_type[0])

close\_connection(conn)

return render\_template('products.html', product\_types=product\_types, username=DATA['user'])

except:

close\_connection(conn)

return redirect('/products')

elif request.method == 'POST':

prod\_type = request.form.get('prod\_type')

return redirect('/product\_type\_list?product\_type={}'.format(prod\_type))

@app.route('/product\_type\_list', methods=['GET'])

def product\_type\_list():

if request.method == 'GET':

prod\_type = request.args.get('product\_type')

try:

conn, cur = get\_cursor()

cur.execute("SELECT prod\_name, prod\_storage, prod\_price FROM product WHERE prod\_type='{}'".format(prod\_type))

products = cur.fetchall()

close\_connection(conn)

return render\_template('product\_type\_list.html', products=products)

except:

close\_connection(conn)

return redirect('/products')

@app.route('/bill', methods=['GET', 'POST'])

def billing():

if request.method == 'GET':

try:

conn, cur = get\_cursor()

cur.execute("SELECT DISTINCT(prod\_type) FROM product")

pt\_data = cur.fetchall()

product\_types = []

for product\_type in pt\_data:

product\_types.append(product\_type[0])

close\_connection(conn)

return render\_template('bill\_product\_type.html', product\_types=product\_types, username=DATA['user'])

except:

close\_connection(conn)

return redirect('/bill')

elif request.method == 'POST':

prod\_type = request.form.get('prod\_type')

return redirect('/bill\_product\_list?prod\_type={}'.format(prod\_type))

@app.route('/bill\_product\_list', methods=['GET', 'POST'])

def bill\_product\_list():

if request.method == 'GET':

prod\_type = request.args.get('prod\_type')

try:

conn, cur = get\_cursor()

cur.execute("SELECT prod\_id, prod\_name, prod\_storage, prod\_price FROM product WHERE prod\_type='{}'".format(prod\_type))

products = cur.fetchall()

close\_connection(conn)

return render\_template('bill\_product.html', products=products, username=DATA['user'])

except:

close\_connection(conn)

return redirect('/bill')

elif request.method == 'POST':

prod\_id = request.form.get('prod\_id')

if 'prods\_cart' not in DATA:

DATA['prods\_cart'] = [prod\_id]

else:

DATA['prods\_cart'].append(prod\_id)

return render\_template('pen\_bill.html', username=DATA['user'])

@app.route('/bill\_final', methods=['GET', 'POST'])

def bill\_final():

if request.method == 'GET':

product\_list = DATA['prods\_cart']

try:

conn, cur = get\_cursor()

postgres\_select\_query = 'SELECT prod\_type, prod\_name, prod\_storage, prod\_price FROM product WHERE prod\_id IN %(product\_list)s'

cur.execute(postgres\_select\_query, { 'product\_list': tuple(product\_list) })

products = cur.fetchall()

total\_amount = 0

for product in products:

amount = product[-1]

amount = int(amount[1::])

total\_amount += amount

total\_amount = '$' + str(total\_amount)

close\_connection(conn)

return render\_template('show\_cart.html', products=products, total\_amount=total\_amount, username=DATA['user'])

except:

close\_connection(conn)

return redirect('/bill')

elif request.method == 'POST':

bill\_payment = request.form.get('payment\_mode')

ct = datetime.datetime.now()

bill\_id = str(int(ct.timestamp()))

cus\_email = DATA['customer'][0]

prod\_ids = DATA['prods\_cart']

username = DATA['user']

try:

conn, cur = get\_cursor()

cur.execute("INSERT INTO bill(bill\_id, bill\_date, cus\_email, payment\_mode) VALUES ('{}', CURRENT\_DATE, '{}', '{}')".format(bill\_id, cus\_email, bill\_payment))

conn.commit()

for prod\_id in prod\_ids:

cur.execute("INSERT INTO purchase(cus\_email, bill\_id, prod\_id, username) VALUES('{}', '{}', {}, '{}')".format(cus\_email, bill\_id, prod\_id, username))

conn.commit()

close\_connection(conn)

return render\_template('options.html', username=username)

except:

close\_connection(conn)

return render\_template('options.html', username=username)

@app.route('/query', methods=['POST'])

def query():

if request.method == 'POST':

query = request.form.get('query')

DATA['query'] = query

return redirect('/query\_result')

@app.route('/query\_result', methods=['GET', 'POST'])

def query\_result():

if request.method == 'GET':

try:

conn, cur = get\_cursor()

query = DATA['query']

cur.execute(query)

query\_result = cur.fetchall()

close\_connection(conn)

print(query\_result)

return render\_template('query\_result.html', query\_results=query\_result, username=DATA['user'])

except:

close\_connection(conn)

return redirect('/options')

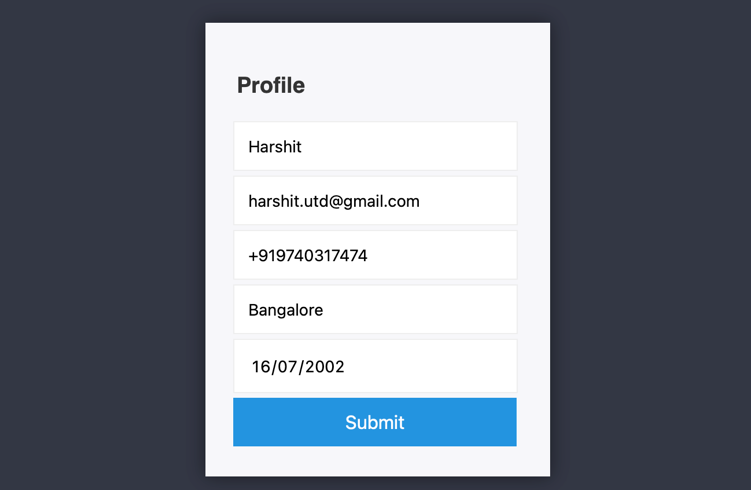
elif request.method == 'POST':

del DATA['user']

return redirect('/options')

CRUD Operations on ‘customer’ relation through frontend

Create

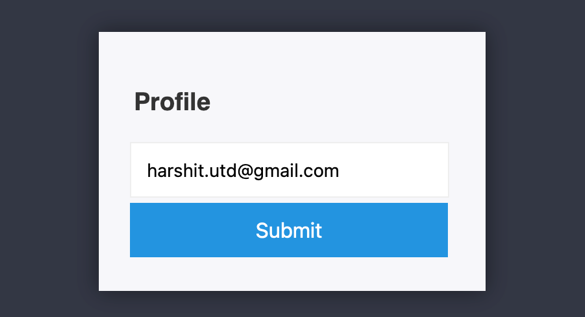


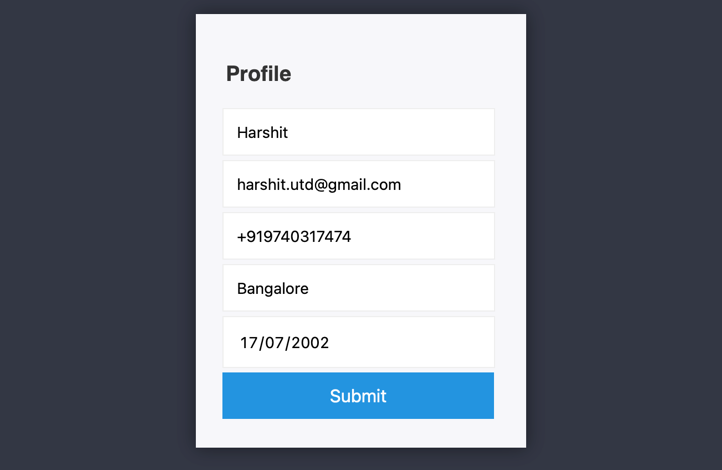
Update

Before update



Steps



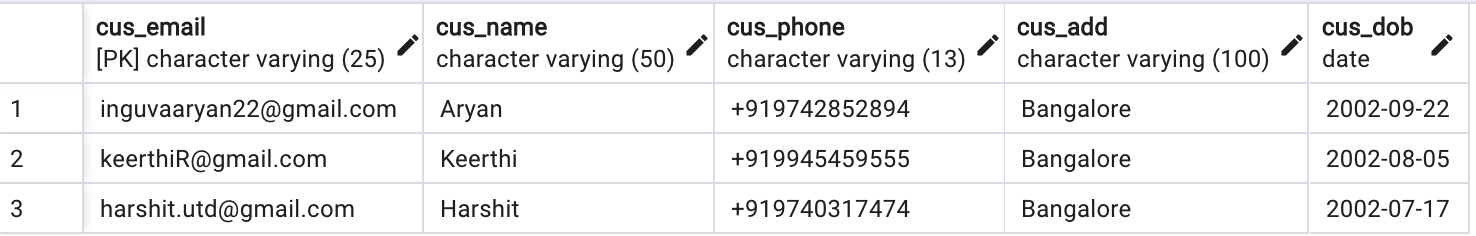


After ‘submit’

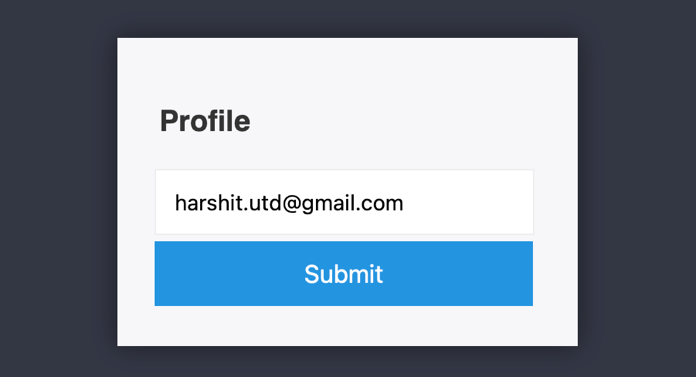


Delete

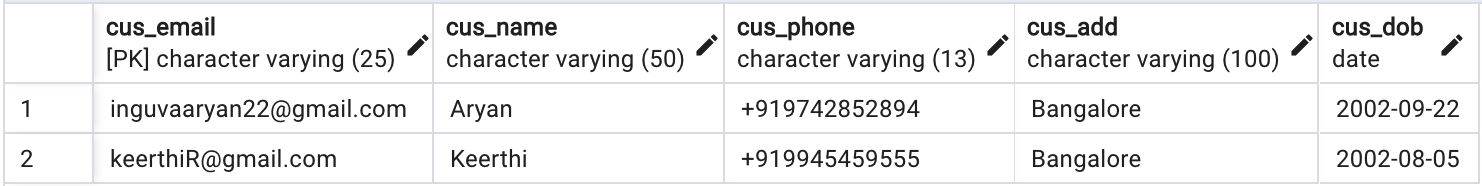
Before delete



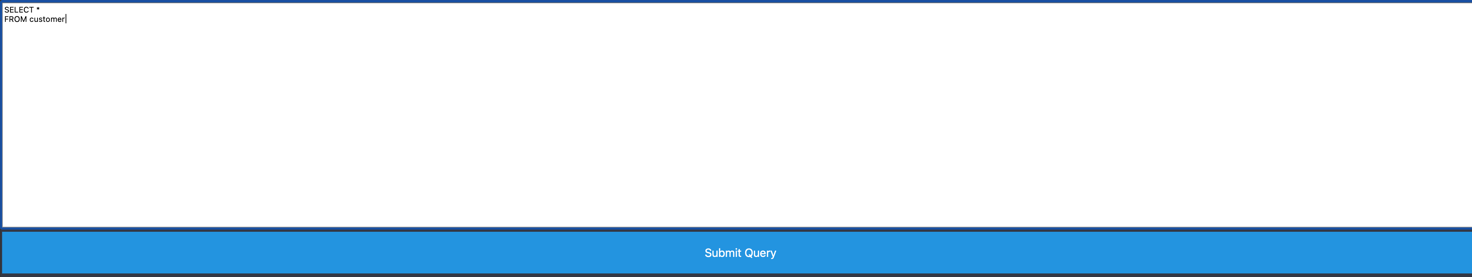
Steps



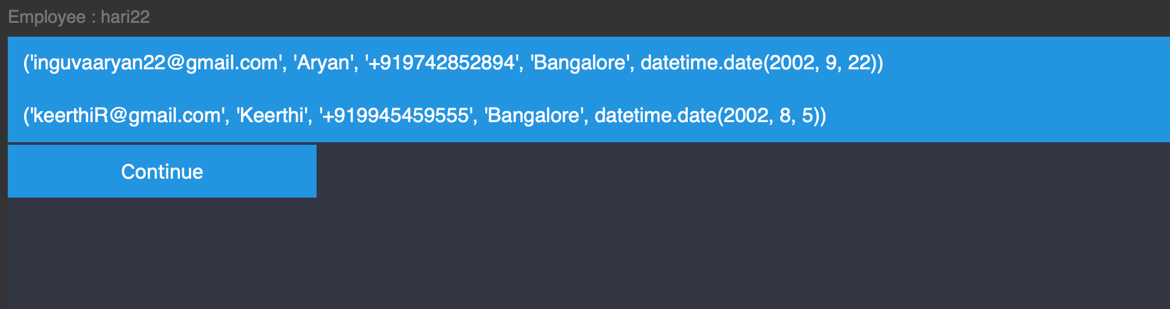
After ‘submit’



Query Box



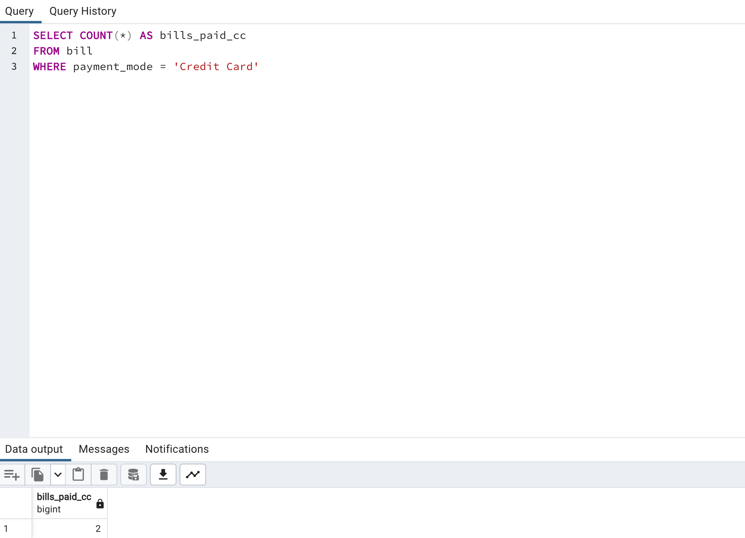
Query Result



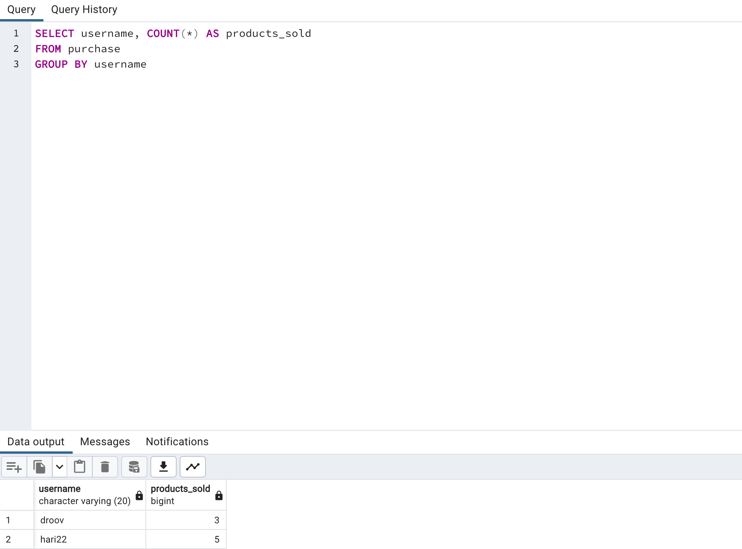
Queries

Aggregate Functions

1. Number of bills paid through credit card.



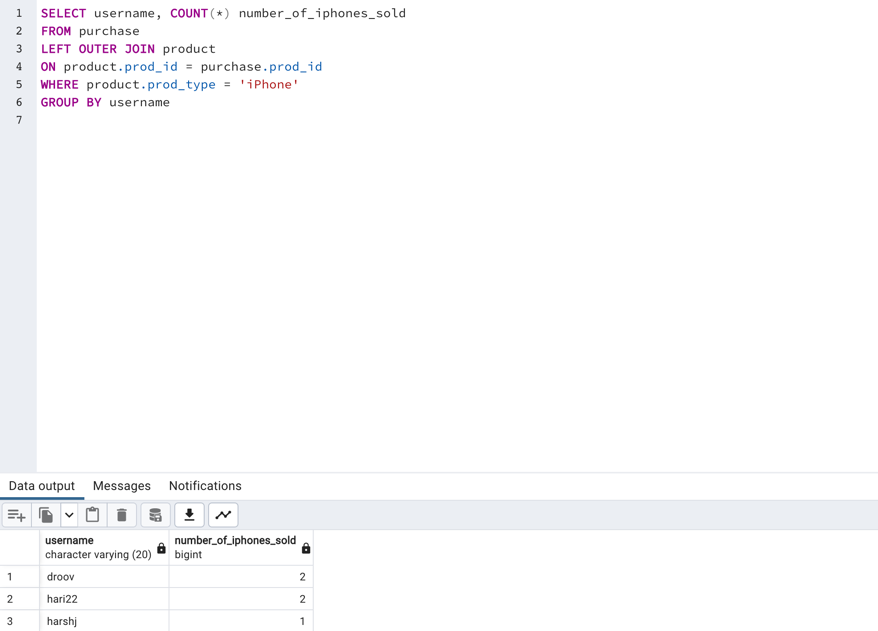
1. Number of products sold by different employees.



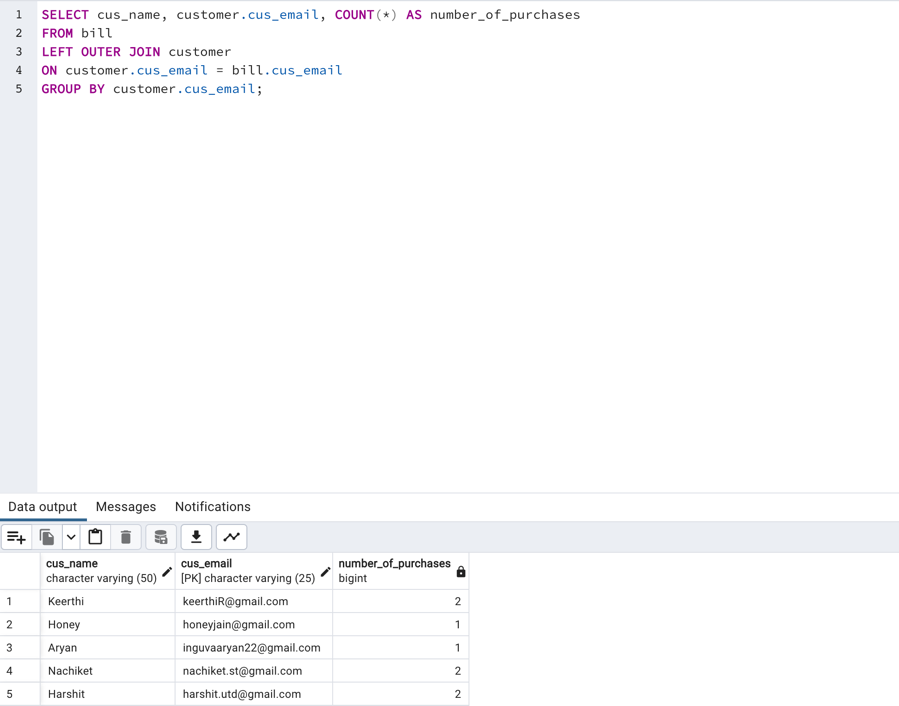
Join Queries

LEFT OUTER JOIN

1. Number of ‘iPhone’s sold by each user.



1. Number of purchases of each customer.



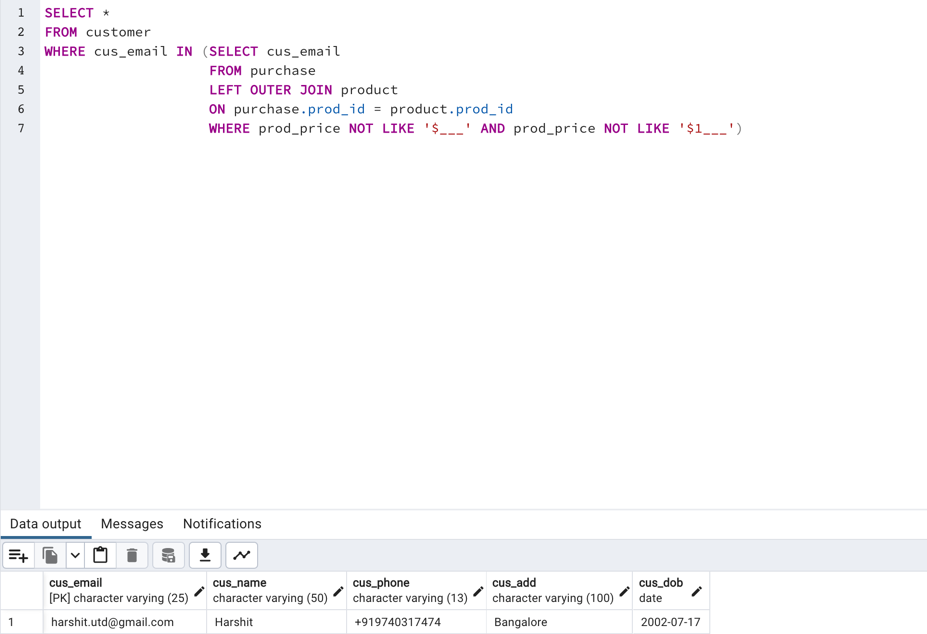
NESTED QUERIES

1. Number of products sold by each user in ‘Bangalore’

Graphical user interface, text

Description automatically generated

1. Customers who have bought a product worth more than $2000.



Correlated Queries

1. Retrieve the details of the customers who have not bought any type of Mac

Text

Description automatically generated with medium confidence

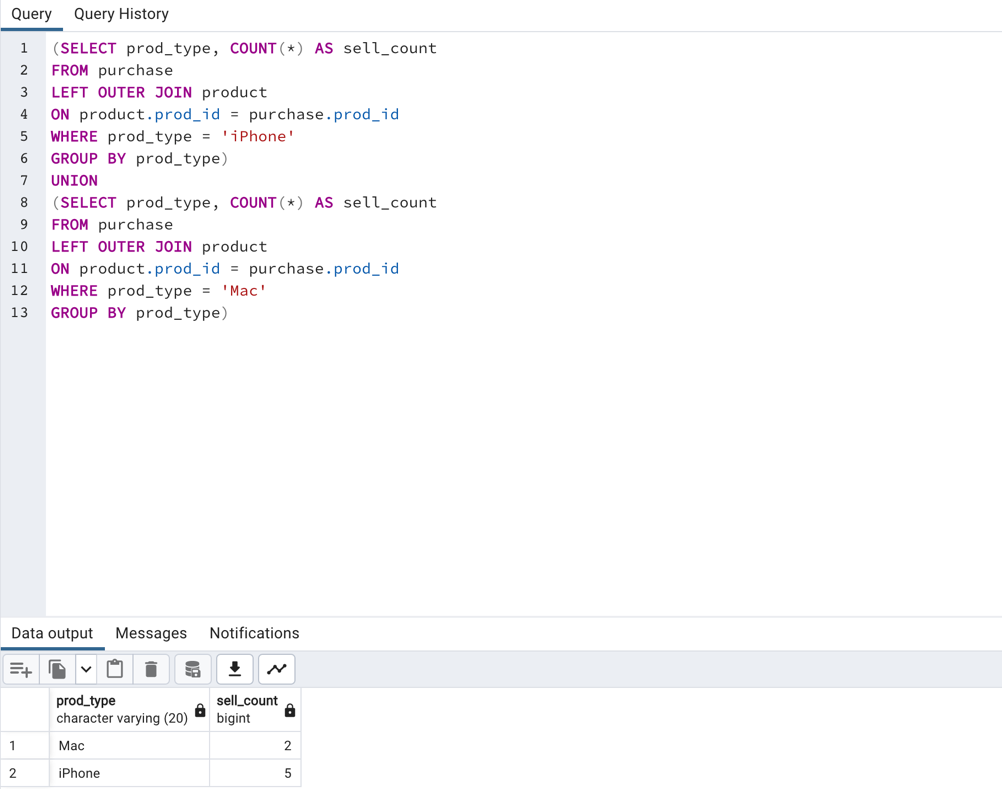
1. Employees who have sold all the types of products.

Graphical user interface, text, application, email

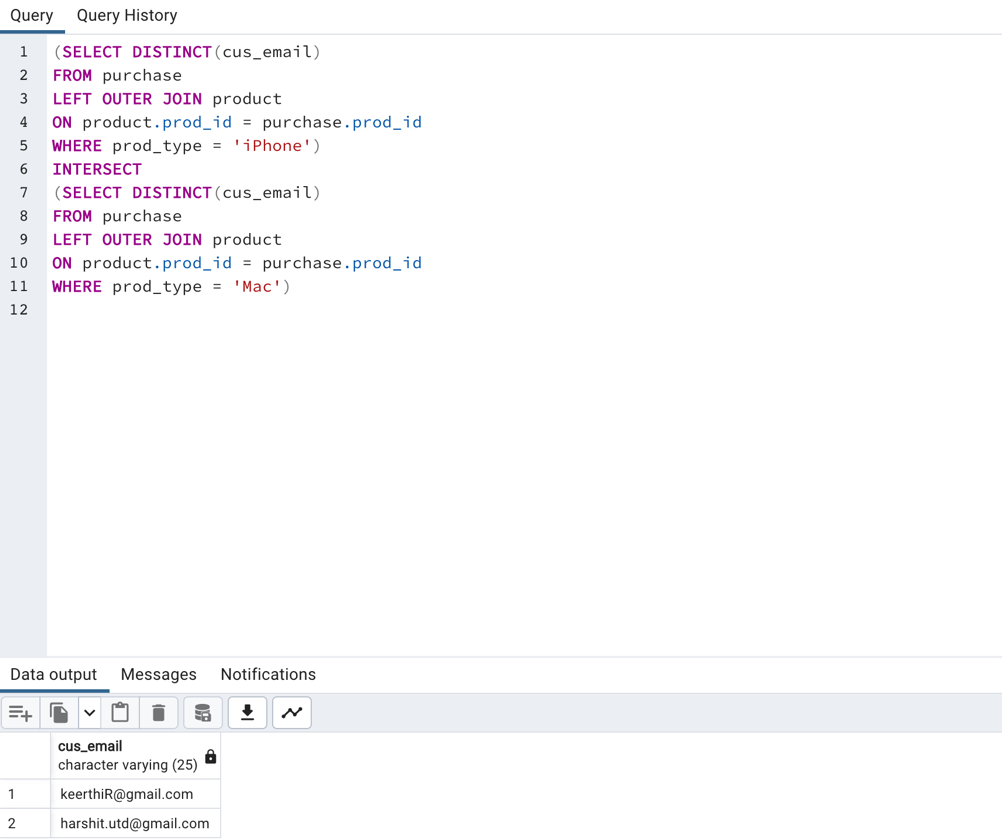
Description automatically generated

Set Operations

1. Number of iPhones and Macs sold.



1. Customers who have bought both Mac and iPhone



VIEW

View Creation

Graphical user interface, text, application

Description automatically generated

Querying the View

