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.

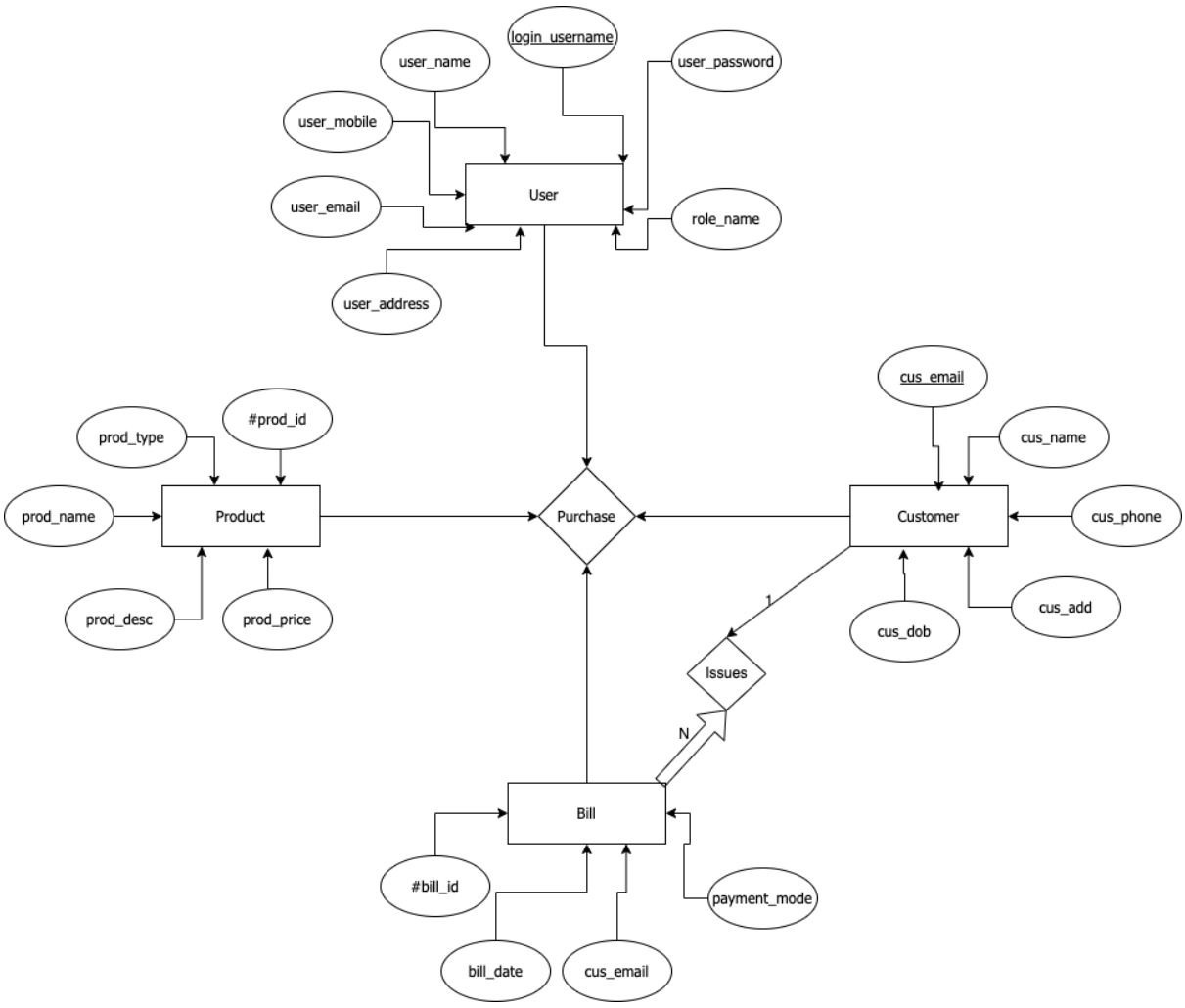
4) The 'bill' entity stores the information regarding about the purchase made by a particular customer. The relation possesses information regarding when the

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.

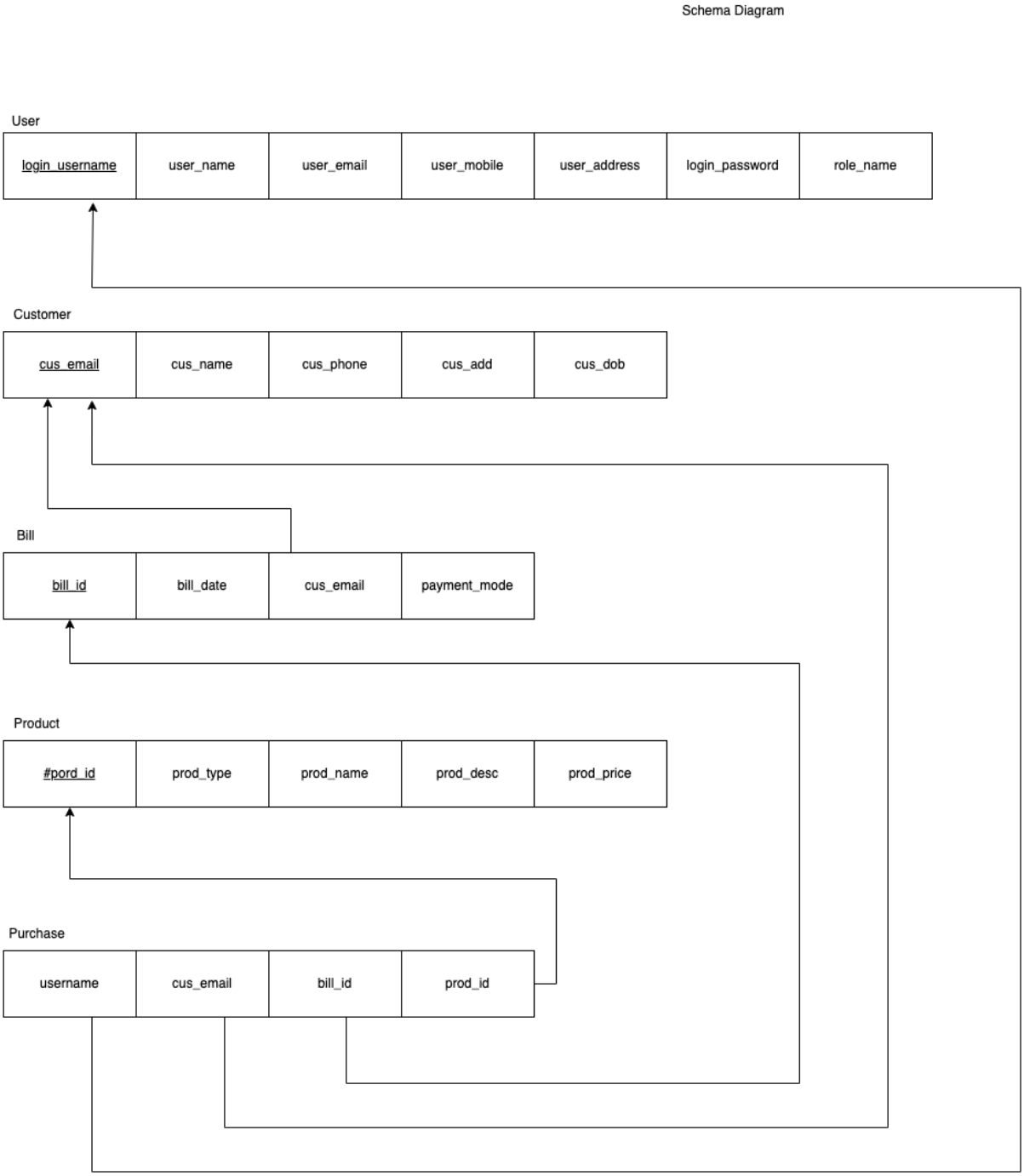
The ’purchase’ relation in the database reflects information regarding a user who sold a particular product to a customer with the bill id.

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


login_username
user_name
user_password
user_mobile
User
user_email
role_name
user_address
cus_email
#prod_id
prod_type
cus_name
prod_name
Product
Purchase
Customer
cus_phone
prod_desc
prod_price
1
cus_dob
cus_add
Issues
N
Bill
#bill_id
payment_mode
bill_date
cus_email

Relational Schema



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

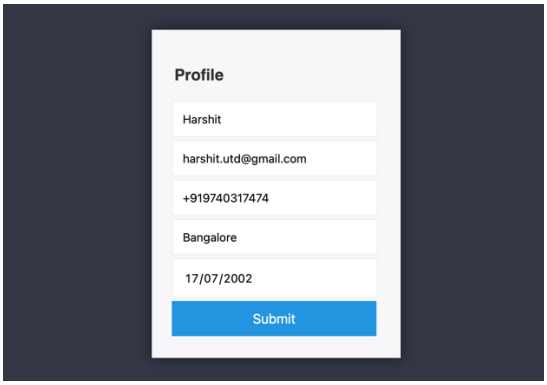
1. Populating ‘users’ table through pgadmin.

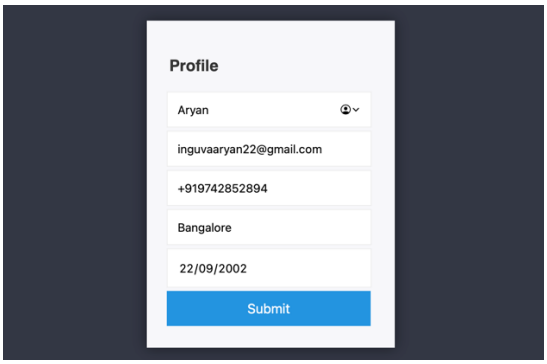
Query Query History

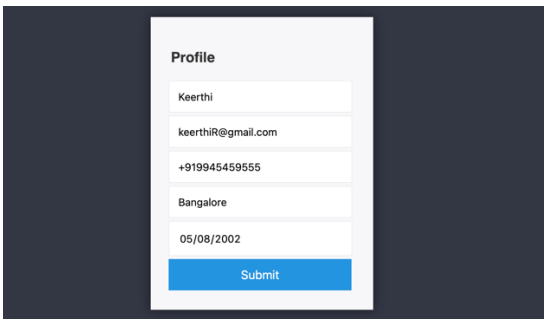
```sql
INSERT INTO users
VALUES
    ('hari22', 'hari ganesan', 'harigana22@gmail.com', '+916362486865', 'Bangalore', '123', 'employee'),
    ('droov', 'dhruv suvarna', 'droovsgmail.com', '+919740667682', 'Bangalore', '456', 'employee'),
    ('harshj', 'harsh jolad', 'harshjload@gmail.com', '+916361198577', 'Bangalore', '789', 'employee')
```

	login_username [PK] character varying (20)	user_name character varying (50)	user_email character varying (25)	user_mobile character varying (13)	user_address character varying (100)	login_password character varying (20)	role_name character varying (20)
1	hari22	hari ganesan	harigana22@gmail.co...	+916362486865	Bangalore	123	employee
2	droov	dhruv suvarna	droovsgmail.com	+919740667682	Bangalore	456	employee
3	harshj	harsh jolad	harshjload@gmail.com	+916361198577	Bangalore	789	employee

2. Populating ‘customers’ table through frontend.

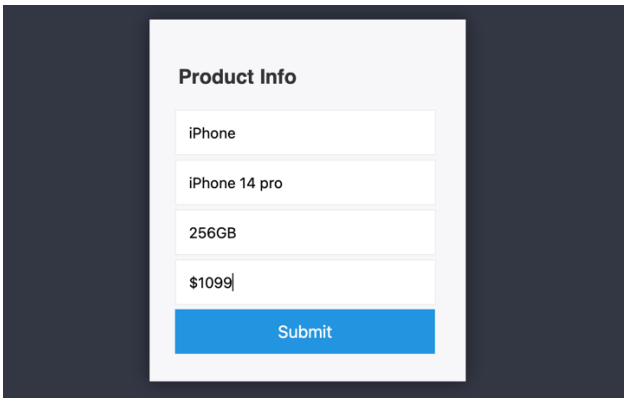


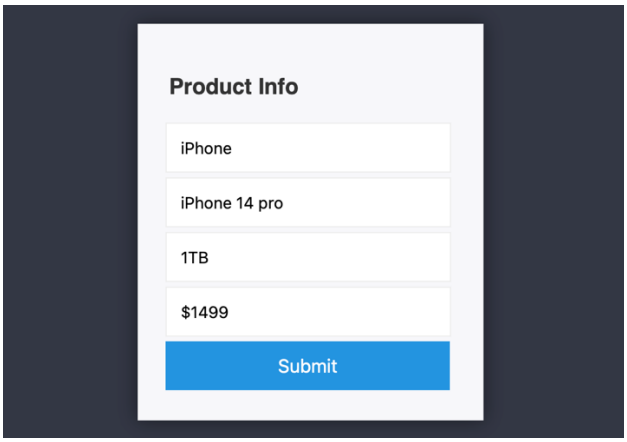




	cus_email [PK] character varying (25)	cus_name character varying (50)	cus_phone character varying (13)	cus_add character varying (100)	cus_dob date
1	harshit.utd@gmail.com	Harshit	+919740317474	Bangalore	2002-07-17
2	inguvaaryan22@gmail.com	Aryan	+919742852894	Bangalore	2002-09-22
3	keerthiR@gmail.com	Keerthi	+919945459555	Bangalore	2002-08-05

3. Populating ‘product’ table

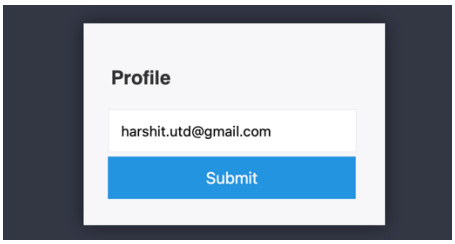


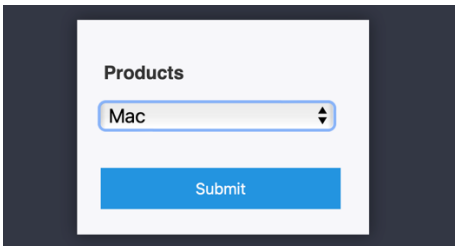


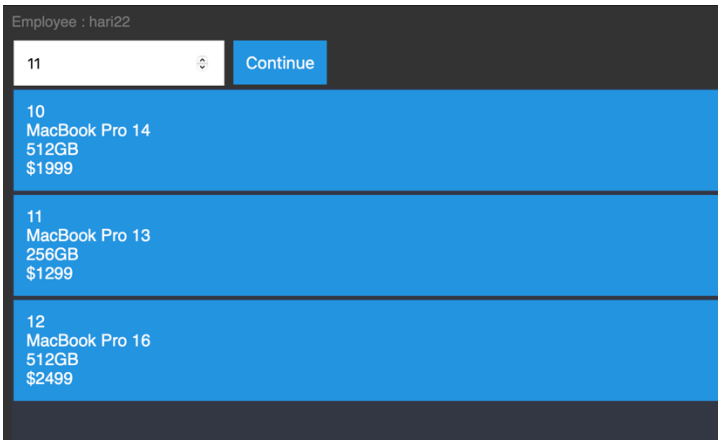
	prod_id [PK] integer	prod_type character varying (20)	prod_name character varying (50)	prod_storage character varying (10)	prod_price character varying (10)
1	7	iPhone	iPhone 14 Pro	256GB	$1099
2	8	iPhone	iPhone 14 pro	1TB	$1499
3	10	Mac	MacBook Pro 14	512GB	$1999
4	11	Mac	MacBook Pro 13	256GB	$1299
5	12	Mac	MacBook Pro 16	512GB	$2499
6	13	iPad	iPad Pro	256GB	$899
7	14	iPad	iPad Pro	1TB	$1499

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

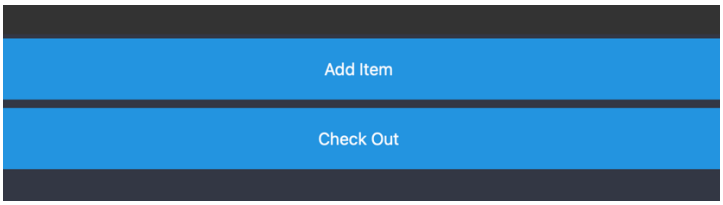
Purchase process:

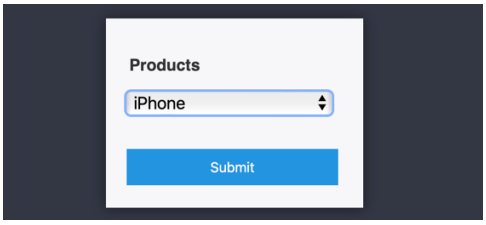


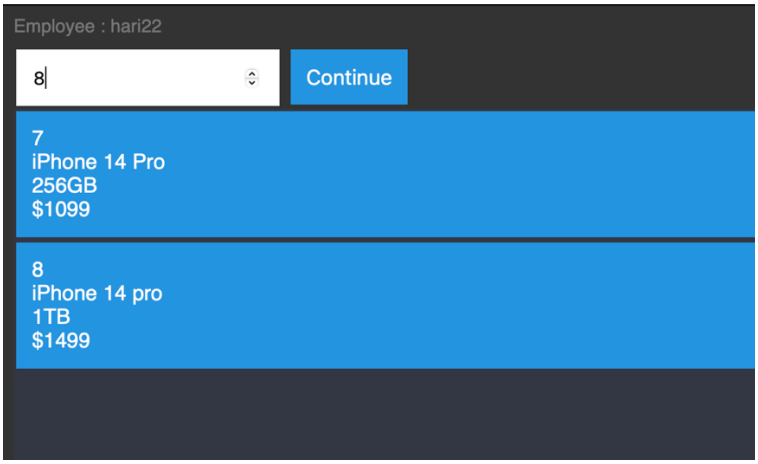




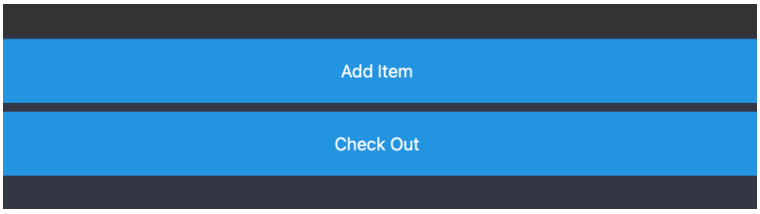
Opting ‘Add item’

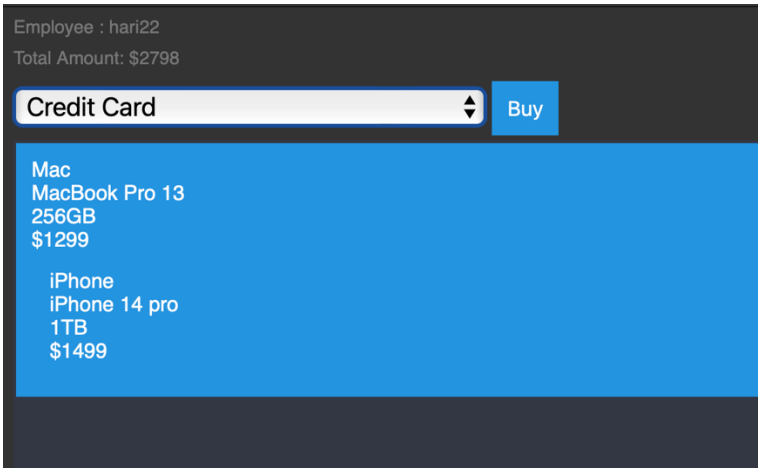






Opting ‘Check Out’:





Database after purchase:

‘bill’ relation

	bill_id [PK] character varying (25)	bill_date date	cus_email character varying (25)	payment_mode character varying (30)
1	1668526281	2022-11-15	harshit.utd@gmail.com	Credit Card

‘purchase’ relation

	cus_email character varying (25)	bill_id character varying (15)	prod_id integer	username character varying (20)
1	harshit.utd@gmail.com	1668526281	11	hari22
2	harshit.utd@gmail.com	1668526281	8	hari22

Triggers

Function creation:

```
CREATE FUNCTION clear_billPurchase()
RETURNS TRIGGER
LANGUAGE plpgsql
AS
$$
BEGIN
    DELETE FROM purchase
    WHERE cus_email = OLD.cus_email;

    DELETE FROM bill
    WHERE cus_email = OLD.cus_email;

    RETURN OLD;
END;
$$;
```

Trigger creation:

```
CREATE TRIGGER handle_billPurchase
    BEFORE DELETE ON customer
    FOR EACH ROW
    EXECUTE FUNCTION clear_billPurchase();
```

Before delete query:

'customer' relation

	cus_email [PK] character varying (25)	cus_name character varying (50)	cus_phone character varying (13)	cus_add character varying (100)	cus_dob date
1	harshit.utd@gmail.com	Harshit	+919740317474	Bangalore	2002-07-17
2	inguvaaryan22@gmail.com	Aryan	+919742852894	Bangalore	2002-09-22
3	keerthiR@gmail.com	Keerthi	+919945459555	Bangalore	2002-08-05

'bill' relation

	bill_id [PK] character varying (25)	bill_date date	cus_email character varying (25)	payment_mode character varying (30)
1	1668526281	2022-11-15	harshit.utd@gmail.com	Credit Card

'purchase' relation

	cus_email character varying (25)	bill_id character varying (15)	prod_id integer	username character varying (20)
1	harshit.utd@gmail.com	1668526281	11	hari22
2	harshit.utd@gmail.com	1668526281	8	hari22

Delete query:

Query Query History

```
DELETE FROM customer
WHERE cus_email = 'harshit.utd@gmail.com'
```

After delete query:

‘customer’ relation

	cus_email [PK] character varying (25)	cus_name character varying (50)	cus_phone character varying (13)	cus_add character varying (100)	cus_dob date
1	inguvaaryan22@gmail.com	Aryan	+919742852894	Bangalore	2002-09-22
2	keerthiR@gmail.com	Keerthi	+919945459555	Bangalore	2002-08-05

‘bill’ relation

	bill_id [PK] character varying (25)	bill_date date	cus_email character varying (25)	payment_mode character varying (30)

‘purchase’ relation

	cus_email character varying (25)	bill_id character varying (15)	prod_id integer	username character varying (20)

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)
```

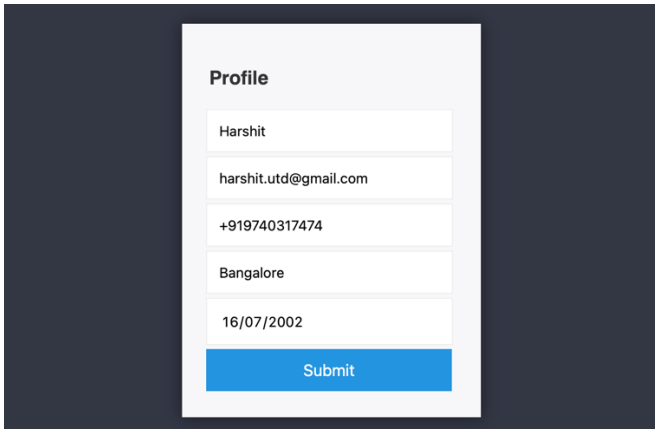
```
            return render_template('bill_product_type.html', product_types=product_types,
username=DATA['user'])
        except:
            close_connection(conn)
            return redirect('/bill')
    elif request.method == 'POST':
        prod_type = request.form.get('prod_type')
        return 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

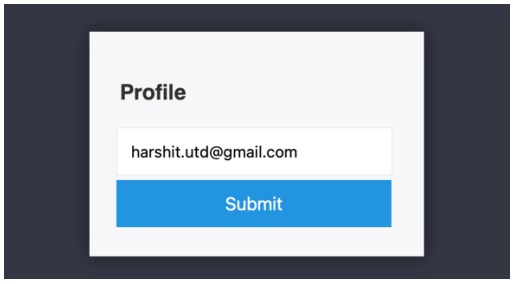


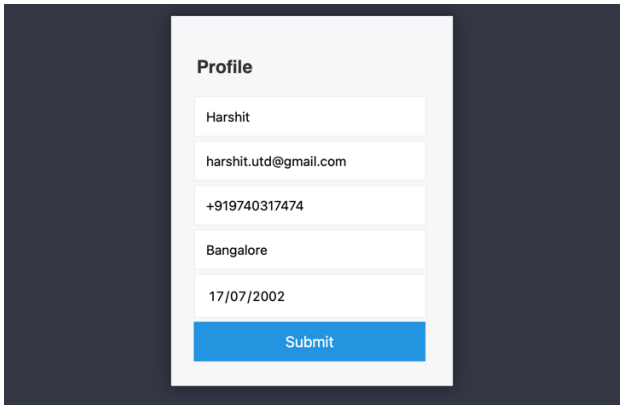
Update

Before update

3	harshit.utd@gmail.com	Harshit	+919740317474	Bangalore	2002-07-16

Steps





After 'submit'

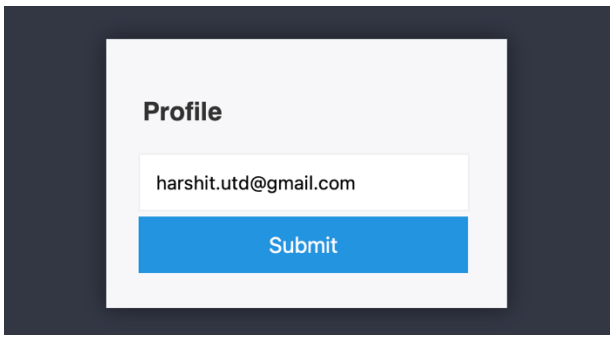
3	harshit.utd@gmail.com	Harshit	+919740317474	Bangalore	2002-07-17

Delete

Before delete

	cus_email [PK] character varying (25)	cus_name character varying (50)	cus_phone character varying (13)	cus_add character varying (100)	cus_dob date
1	inguvaaryan22@gmail.com	Aryan	+919742852894	Bangalore	2002-09-22
2	keerthiR@gmail.com	Keerthi	+919945459555	Bangalore	2002-08-05
3	harshit.utd@gmail.com	Harshit	+919740317474	Bangalore	2002-07-17

Steps



After 'submit'

	cus_email [PK] character varying (25)	cus_name character varying (50)	cus_phone character varying (13)	cus_add character varying (100)	cus_dob date
1	inguvaaryan22@gmail.com	Aryan	+919742852894	Bangalore	2002-09-22
2	keerthiR@gmail.com	Keerthi	+919945459555	Bangalore	2002-08-05

Query Box

```
SELECT *
FROM customer
```

Submit Query

Query Result

Employee : hari22

('inguvaaryan22@gmail.com', 'Aryan', '+919742852894', 'Bangalore', datetime.date(2002, 9, 22))

('keerthiR@gmail.com', 'Keerthi', '+919945459555', 'Bangalore', datetime.date(2002, 8, 5))

Continue

Queries

Aggregate Functions

1) Number of bills paid through credit card.

Query Query History

```
SELECT COUNT(*) AS bills_paid_cc
FROM bill
WHERE payment_mode = 'Credit Card'
```

Data output Messages Notifications

	bills_paid_cc bigint
1	2

2) Number of products sold by different employees.

Query Query History

```
SELECT username, COUNT(*) AS products_sold
FROM purchase
GROUP BY username
```

Data output Messages Notifications

	username character varying (20)	products_sold bigint
1	droov	3
2	hari22	5

Join Queries

LEFT OUTER JOIN

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

```sql
SELECT username, COUNT(*) number_of_iphones_sold
FROM purchase
LEFT OUTER JOIN product
ON product.prod_id = purchase.prod_id
WHERE product.prod_type = 'iPhone'
GROUP BY username
```

Data output Messages Notifications

	username character varying (20)	number_of_iphones_sold bigint
1	droov	2
2	hari22	2
3	harshj	1

2) Number of purchases of each customer.

```sql
SELECT cus_name, customer.cus_email, COUNT(*) AS number_of_purchases
FROM bill
LEFT OUTER JOIN customer
ON customer.cus_email = bill.cus_email
GROUP BY customer.cus_email;
```

Data output Messages Notifications

	cus_name character varying (50)	cus_email [PK] character varying (25)	number_of_purchases bigint
1	Keerthi	keerthiR@gmail.com	2
2	Honey	honeyjain@gmail.com	1
3	Aryan	inguvaaryan22@gmail.com	1
4	Nachiket	nachiket.st@gmail.com	2
5	Harshit	harshit.utd@gmail.com	2

NESTED QUERIES

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

Query Query History

```sql
SELECT username, COUNT(*)
FROM (SELECT username, customer.cus_email, cus_add
FROM purchase
LEFT OUTER JOIN customer
ON purchase.cus_email = customer.cus_email) AS cte
GROUP BY username, cus_add
HAVING cus_add = 'Bangalore'
```

Data output Messages Notifications

	username character varying (20)	count bigint
1	droov	3
2	hari22	5

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

```sql
SELECT *
FROM customer
WHERE cus_email IN (SELECT cus_email
                    FROM purchase
                    LEFT OUTER JOIN product
                    ON purchase.prod_id = product.prod_id
                    WHERE prod_price NOT LIKE '$___' AND prod_price NOT LIKE '$1___')
```

Data output Messages Notifications

	cus_email [PK] character varying (25)	cus_name character varying (50)	cus_phone character varying (13)	cus_add character varying (100)	cus_dob date
1	harshit.utd@gmail.com	Harshit	+919740317474	Bangalore	2002-07-17

Correlated Queries

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

```sql
SELECT *
FROM customer AS c1
WHERE NOT EXISTS   (SELECT *
                    FROM purchase
                    LEFT OUTER JOIN product
                    ON product.prod_id = purchase.prod_id
                    WHERE prod_type = 'Mac' AND c1.cus_email = purchase.cus_email)
```

Data output Messages Notifications

	cus_email [PK] character varying (25)	cus_name character varying (50)	cus_phone character varying (13)	cus_add character varying (100)	cus_dob date
1	inguvaaryan22@gmail.com	Aryan	+919742852894	Bangalore	2002-09-22
2	nachiket.st@gmail.com	Nachiket	+918861106230	Vellore	2002-02-18
3	honeyjain@gmail.com	Honey	+916395167212	Surat	2001-03-06

2) Employees who have sold all the types of products.

Query Query History

```sql
SELECT *
FROM users
WHERE NOT EXISTS    (
                        (
                            SELECT prod_id
                            FROM product
                        )
                        EXCEPT
                        (
                            SELECT prod_id
                            FROM purchase
                            WHERE username = users.login_username
                        )
                    )
```

Data output Messages Notifications

	login_username [PK] character varying (20)	user_name character varying (50)	user_email character varying (25)	user_mobile character varying (13)	user_address character varying (100)	login_password character varying (20)	role_name character varying (20)
1	hari22	hari ganesan	harigana22@gmail.co...	+916362486865	Bangalore	123	employee

Set Operations

1) Number of iPhones and Macs sold.

Query Query History

```sql
(SELECT prod_type, COUNT(*) AS sell_count
FROM purchase
LEFT OUTER JOIN product
ON product.prod_id = purchase.prod_id
WHERE prod_type = 'iPhone'
GROUP BY prod_type)
UNION
(SELECT prod_type, COUNT(*) AS sell_count
FROM purchase
LEFT OUTER JOIN product
ON product.prod_id = purchase.prod_id
WHERE prod_type = 'Mac'
GROUP BY prod_type)
```

Data output Messages Notifications

	prod_type character varying (20)	sell_count bigint
1	Mac	2
2	iPhone	5

2) Customers who have bought both Mac and iPhone

Query Query History

```sql
(SELECT DISTINCT(cus_email)
FROM purchase
LEFT OUTER JOIN product
ON product.prod_id = purchase.prod_id
WHERE prod_type = 'iPhone')
INTERSECT
(SELECT DISTINCT(cus_email)
FROM purchase
LEFT OUTER JOIN product
ON product.prod_id = purchase.prod_id
WHERE prod_type = 'Mac')
```

Data output Messages Notifications

	cus_email character varying (25)
1	keerthiR@gmail.com
2	harshit.utd@gmail.com

VIEW

View Creation

Query Query History

```sql
CREATE VIEW customer_product_relation   AS
                                        SELECT cte1.cus_email, cus_name, cus_phone, bill_id, bill_date, prod_name, prod_storage, prod_price
                                        FROM (  SELECT cus_email, bill_id, bill_date, prod_name, prod_storage, prod_price
                                                FROM (  SELECT purchase.cus_email, bill.bill_id, bill_date, prod_id
                                                        FROM purchase
                                                        LEFT OUTER JOIN bill
                                                        ON purchase.bill_id = bill.bill_id ) AS cte
                                                LEFT OUTER JOIN product
                                                ON cte.prod_id = product.prod_id    ) AS cte1
                                        LEFT OUTER JOIN customer
                                        ON cte1.cus_email = customer.cus_email
```

Querying the View

Query Query History

```sql
SELECT *
FROM customer_product_relation
```

Data output Messages Notifications

	cus_email character varying (25)	cus_name character varying (50)	cus_phone character varying (13)	bill_id character varying (25)	bill_date date	prod_name character varying (50)	prod_storage character varying (10)	prod_price character varying (10)
1	harshit.utd@gmail.com	Harshit	+919740317474	1668571337	2022-11-16	iPhone 14 pro	1TB	$1499
2	harshit.utd@gmail.com	Harshit	+919740317474	1668571337	2022-11-16	MacBook Pro 16	512GB	$2499
3	keerthiR@gmail.com	Keerthi	+919945459555	1668571418	2022-11-16	iPhone 14 Pro	256GB	$1099
4	inguvaaryan22@gmail.co...	Aryan	+919742852894	1668571450	2022-11-16	iPad Pro	1TB	$1499
5	inguvaaryan22@gmail.co...	Aryan	+919742852894	1668571450	2022-11-16	iPad Pro	256GB	$899
6	harshit.utd@gmail.com	Harshit	+919740317474	1668571574	2022-11-16	iPad Pro	256GB	$899
7	keerthiR@gmail.com	Keerthi	+919945459555	1668571626	2022-11-16	iPad Pro	256GB	$899
8	keerthiR@gmail.com	Keerthi	+919945459555	1668571626	2022-11-16	MacBook Pro 13	256GB	$1299
9	honeyjain@gmail.com	Honey	+916395167212	1668576653	2022-11-16	iPhone 14 Pro	256GB	$1099
10	nachiket.st@gmail.com	Nachiket	+918861106230	1668576679	2022-11-16	iPhone 14 pro	1TB	$1499
11	nachiket.st@gmail.com	Nachiket	+918861106230	1668576735	2022-11-16	iPhone 14 pro	1TB	$1499
12	nachiket.st@gmail.com	Nachiket	+918861106230	1668576735	2022-11-16	iPad Pro	1TB	$1499