Database Final Project Report

Subject : Cellphone Rating System

Member: 110652019 林楷傑, 110550135 黃孚翔, 110550152 成文瀚,

110705013 沈昱宏, 110950014 黃為碩

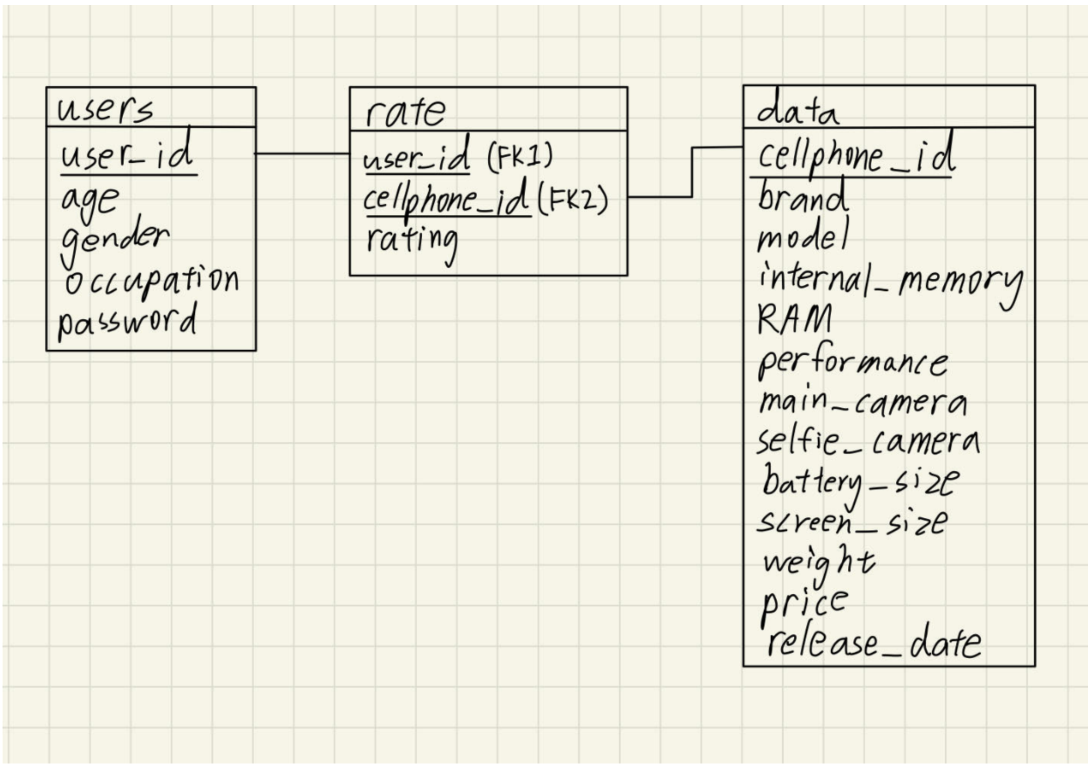
Github: https://github.com/KJLdefeated/Database-Final-project.git

Demo Video: https://drive.google.com/file/d/1xBWdgeF4hXA-7Ixw4uB62ry3d2FRD435/view?usp=share_link

1. An introduction of your application, including why you want to develop the application and the main functions of your application.

When we are choosing a cell phone, you have to compare their functions and their overall evaluation given by the other users. We develop a website for anyone who is having trouble choosing cell phones. In our application, you can have quick access to all the popular cell phone information, and you can also filter the cell phones according to your preferences. In addition, we have made some analysis based on a questionnaire, so you can know how the public or a particular group of people think of cell phones.

2. Database design - describe the schema of all your tables in the database, including keys and index, if applicable (why you need the keys, or why you think that adding an index is or is not helpful).



We use user_id and cellphone_id as keys in order to union the tables.

```
create table rate(
    user_id varchar(100),
    cellphone_id int,
    rating real,
    primary key (user_id,cellphone_id),
    foreign key (user_id) references users(user_id),
    foreign key (cellphone_id) references data(cellphone_id),
    check(rating<=10 and rating >=0)
);

create table users(
    user_id varchar(100),
    age real,
    gender varchar(100),
    occupation varchar(100),
    password varchar(100),
    primary key (user_id),
    check(gender in ('Male','Female','null'))
);

create table data(
    cellphone_id int,
    brand varchar(100),
    model varchar(100),
    internal_memory real,
    RAM real,
    performance real,
    main_camera real,
    selfie_camera real,
    battery_size real,
    screen_size real,
    weight real,
    price real,
    release_date date,
    primary key(cellphone_id)
);
```

3. Database design - describe the normal form of all your tables. If the tables are not in BCNF, please include the reason for it (performance trade-off, etc.).

After we normalize the “data” table, our tables are all in BCNF, since all redundancy based on functional dependency has been removed.

The function dependency in users:

(user_id)→(all) and user-id is the superkey of users ,so users is in BCNF.

The function dependency in rate:

(user_id,cellphon_id)→(all) similarly rate is in BCNF.

The function dependency in data:

(cellphone_id)→(all)   similarly data is in BCNF.

4. From the data sources to the database - describe the data source and the original format.

Here is the data source:

https://www.kaggle.com/datasets/meirnizri/cellphones-recommendations?select=cellphones+users.csv

The data source consists of three csv. files:

1. cellphone users (user_id, age, gender, occupation)
2. cellphone ratings (user_id, cellphone_id, rating)
3. cellphone data (cellphone_id, brand, model, operating system, internal memory, RAM, performance, main camera, selfie camera, battery size, screen size, weight, price, release date)

In cellphone data, there is originally one attribute for the operating system and a simple function dependency (brand→operating system, apple uses ios, others use android). Therefore, it violates BCNF, but we won't use the operating system, so operating is not in our data table.

5. From the data sources to the database - describe the methods of importing the original data to your database and strategies for updating the data, if you have one.

We use Amazon RDS to maintain our database, then use PGAdmin to connect this database. We first try to import our original data through PGAdmin. Since there are several errors in the previous data, we use pandas in python to deal with all the errors (all the details are in the python_modified file). The resource of our database will never be updated, so we decided to update all the data by ourselves or the other users. We can either use PGAdmin or the application to update the data. In our application, we will have a page for data updating. You can input the data according to the format, and the data will be sent into the database.

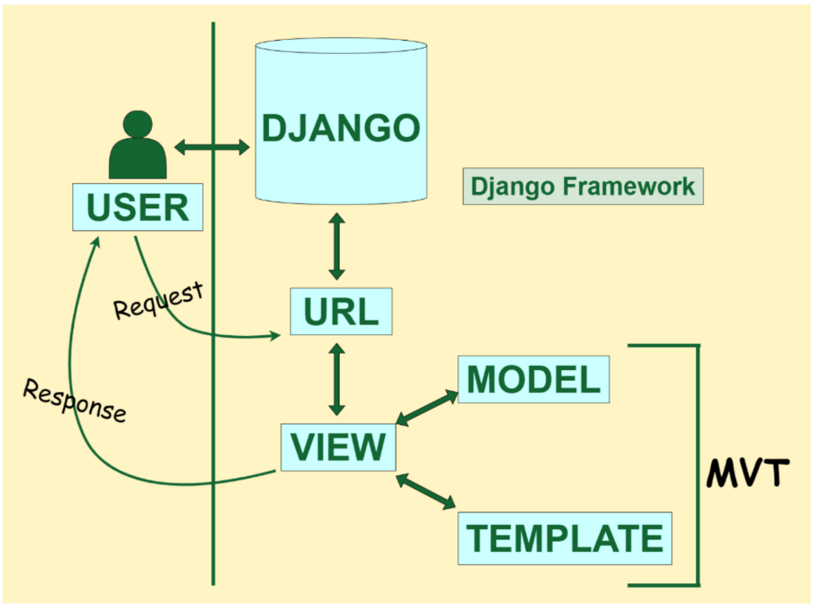
6. Application with database - explain why your application needs a database.

A database can help our application store and retrieve data in an organized and structured way. This can make it easier for our application to access and manipulate the data as needed. Without a database, our application can't store a large amount of data and manage the data efficiently, which might result in difficulties in analyzing the cell phones.

7. Application with database - includes the queries that are performed by your application, how your application performed these queries (connections between application and database), and what are the cooperating functions for your application.

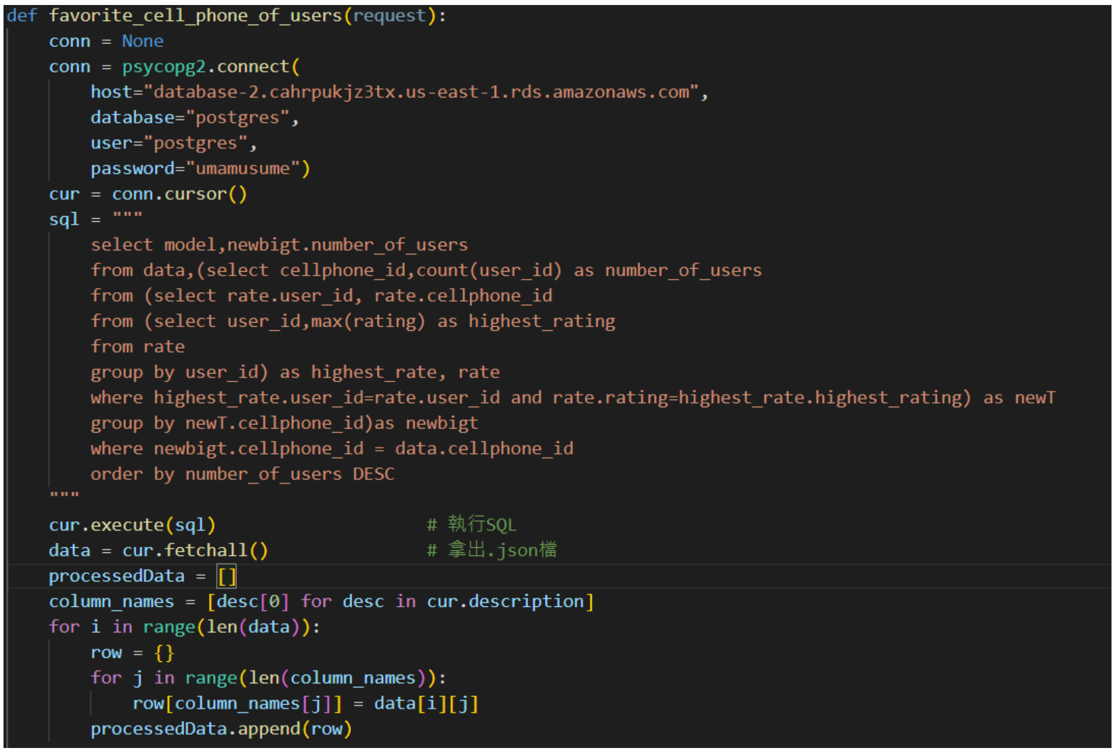
- Backend:

  We built this website with django as backend architecture and connected postgresql with django such that we can fetch the data from the database in the backend. We did not deploy this application to the server since we did not have time to study how to do it, so it can be seen only on the local end. But we will deploy the application to the AWS ec2 in the future.



- Queries and cooperating functions:
  We have implemented some functions for users to become familiar with data and check their demand. For example, finding the highest rating cellphone. We implemented the SQL query and ran it in the backend using the Python package psycopg2.

  For example, when we want to get “favorite_cell_phone_of_users”, we will exectute the following code (connect the database + run query)



```python
def favorite_cell_phone_of_users(request):
    conn = None
    conn = psycopg2.connect(
        host="database-2.cahrpukjz3tx.us-east-1.rds.amazonaws.com",
        database="postgres",
        user="postgres",
        password="umamusume")
    cur = conn.cursor()
    sql = """
        select model,newbigt.number_of_users
        from data,(select cellphone_id,count(user_id) as number_of_users
        from (select rate.user_id, rate.cellphone_id
        from (select user_id,max(rating) as highest_rating
        from rate
        group by user_id) as highest_rate, rate
        where highest_rate.user_id=rate.user_id and rate.rating=highest_rate.highest_rating) as newT
        group by newT.cellphone_id)as newbigt
        where newbigt.cellphone_id = data.cellphone_id
        order by number_of_users DESC
    """
    cur.execute(sql)                    # 執行SQL
    data = cur.fetchall()               # 拿出.json檔
    processedData = []
    column_names = [desc[0] for desc in cur.description]
    for i in range(len(data)):
        row = {}
        for j in range(len(column_names)):
            row[column_names[j]] = data[i][j]
        processedData.append(row)
```

Here we list all the query we have used:

A. cellphone average rating (with average of all ratings having cellphone_id -1)

(select 'average' as model,round(cast(avg(rating) as decimal),3) as average
from rate,data)
union
(select model,newt.average
from(select cellphone_id,round(cast(avg(rating) as decimal),3) as average
from rate
group by cellphone_id) as newt,data
where newt.cellphone_id=data.cellphone_id)
order by average desc;

B. favorite cell phone of users

(count the cell which has the highest rating given by individual user)

select model,newbigt.number_of_users
from data,(select cellphone_id,count(user_id) as number_of_users

```
from (select rate.user_id, rate.cellphone_id
from (select user_id,max(rating) as highest_rating
from rate
group by user_id) as highest_rate, rate
where highest_rate.user_id=rate.user_id and
rate.rating=highest_rate.highest_rating) as newT
group by newT.cellphone_id)as newbigt
where newbigt.cellphone_id = data.cellphone_id
order by number_of_users DESC
```

C. amount of cellphone owned by all users

```
select model,newt.count
from(select cellphone_id,count(user_id)
from rate
group by cellphone_id)as newt,data
where data.cellphone_id=newt.cellphone_id
order by newt.count desc;
```

D. list the cell phones which has average rating bigger than total average

```
select model,newt.average
from(select cellphone_id as cellphone_id,cellphones.average
from(select cellphone_id,round(cast(avg(rating) as decimal),3) as
average
from rate
group by cellphone_id) as cellphones, (select round(cast(avg(rating)
as decimal),3) as average from rate) as tavg
where cellphones.average>tavg.average)as newt,data
where newt.cellphone_id=data.cellphone_id
order by newt.average desc;
```

E. market share -brand (in this survey)

```
with brand_amount(brand,amount) as
```

```
(select data.brand, count(*) as amount
from data
join rate on data.cellphone_id=rate.cellphone_id
group by data.brand
order by amount desc)

select brand_amount.brand,
concat(round(cast(brand_amount.amount as decimal)/9.9,2),'%') as
market_share
from brand_amount
order by brand_amount.amount desc
```

G. list the average rating & number of ratings given by females

```
select
data.model,tablelast.average_rating,tablelast.number_of_ratings
from(select cellphone_id,round(cast(avg(rating) as decimal),3) as
average_rating,count(user_id) as number_of_ratings
from (select * from(select * from rate NATURAL join users) as bigT
where gender='Female')as newbigT
group by cellphone_id)as tablelast,data
where tablelast.cellphone_id=data.cellphone_id
order by tablelast.average_rating desc;
```

H. list the top 10 cellphone for a particular occupation

```
select data.model,newt.average
from(select round(cast(avg(rating) as decimal),3) as
average,cellphone_id
from(select * from rate NATURAL join users where
occupation='information technology') as bigt
group by cellphone_id
limit 10)   as newt,data
where data.cellphone_id=newt.cellphone_id
order by newt.average desc;
```

I. list the top 10 cellphone for elders (with age>average age)

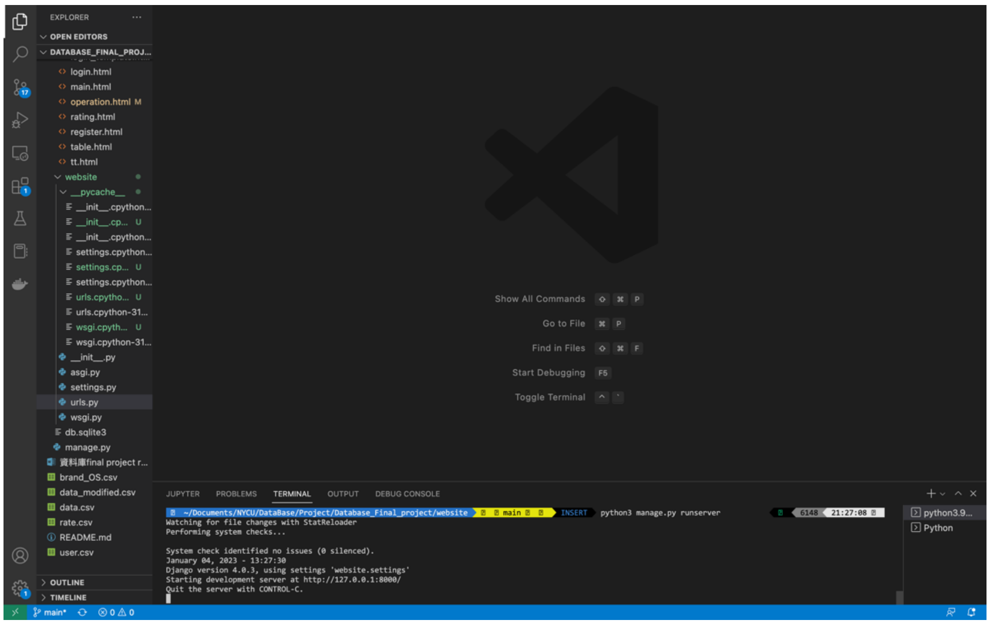
```
select data.model,newt.averageRate
```

```
from(select cellphone_id,round(cast(avg(rating) as decimal),3) as
averageRate
from(select * from rate natural join (select user_id from
(select avg(age) as avg_age from users) as avgT,users
where users.age>avgT.avg_age) as newt) as bigT
group by cellphone_id
limit 10) as newt,data
where newt.cellphone_id=data.cellphone_id
order by newt.averageRate desc;
```

- Frontend is built by HTML, Javascript and CSS. We have done our best to improve user experience.

8. All the other details of your application that you want us to know.

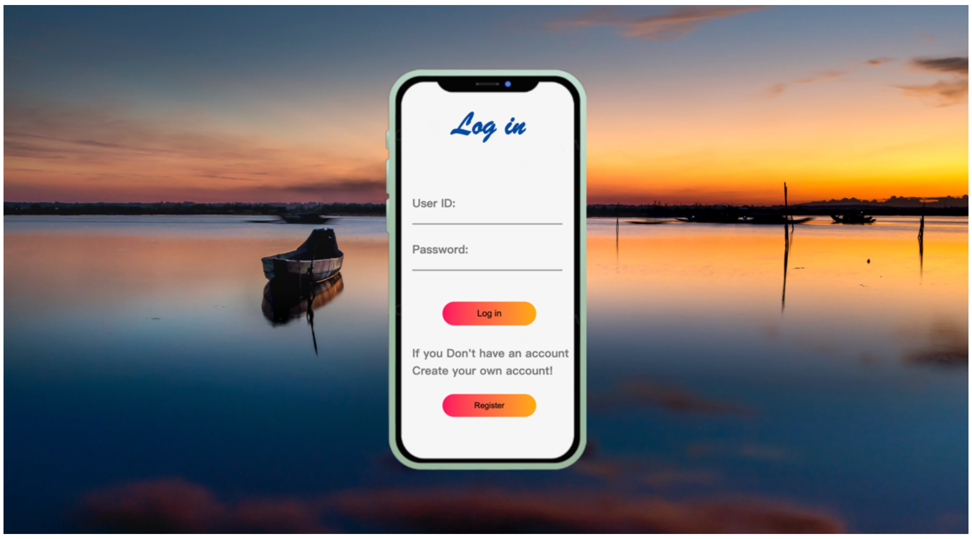
Since out application did not deploy onto the server, we need to use it in the local host. Here is how we do:
After installing the source code, Django and other package in our environment, we can enter “python manage.py runserver” in the terminal.



```
~/Documents/NYCU/DataBase/Project/Database_Final_project/website  main  INSERT  python3 manage.py runserver      6148  21:27:08
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
January 04, 2023 - 13:27:30
Django version 4.0.3, using settings 'website.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Click the link in the terminal which is http://127.0.0.1:8000/
Then we can see the website:



Other website detail are shown in the demo video.

Updata policy:

To update the data, we created the rating page.In the rating page, there are two kinds of data you can update. When you enter the rating page, you will first see an animation. After that, you choose the corresponding bar, then you can see the format of the data you want to add. If the updating is done, you will see an animation indicating the updating is done.

To access the operation page in our application, users need to login. We designed an authentication machanism and a login page as our first page. To control the login page, the users table keeps the record of all the users and their password. If a user doesn't have an account, the user needs to register, which updates the users table. After login, there are several bottons in our operating page. The description of the analysis is next to the bottons, when you press the botton, a python program with SQL query is triggered, and the result of the query will be shown under the page.