CUHKSZ Food Rating and Recommendation System

CSC 3170

CUHKSZ Food Rating and Recommendation System

Author:	*Student Number:*
Xu Kai	121090650
Zhang Zirui	122090784
Chen Fanxinyu	121090028
Cai Liuyu	121040046
Lin Yuzhou	122090324
Li tao	121090279
Yu Jinglin	121090729
Zhuo Can	120090397

May 2, 2024

Contents

1 Introduction and Motivation

The Chinese University of Hong Kong Shenzhen (CUHK-SZ) is a pluralistic university which has a large number of students and staff from all over the world. As one of the most important infrastructure, the catering system in CUHK-SZ has large amount of clients and has long been provided all kinds of pleasing food to meet the demands from customers with different taste. In order to provide a convenient platform to help these large amount of staff and students to quickly and easily find out the dish that meets their taste, we decide to build a food rating and recommendation system specially for our campus.

The system will basically satisfy two functions, showing the rating grade of a dish, which helps users to decide whether to order it or not, and showing the detail information such as the name of stand and location of canteen to better help user to find the dish.

2 Database Structure Design

According to the motivation and users' requirements, we developed the E-R diagram of our database. There are 5 entities and 5 relationships (In Figure1 green for entity and blue for relationship).

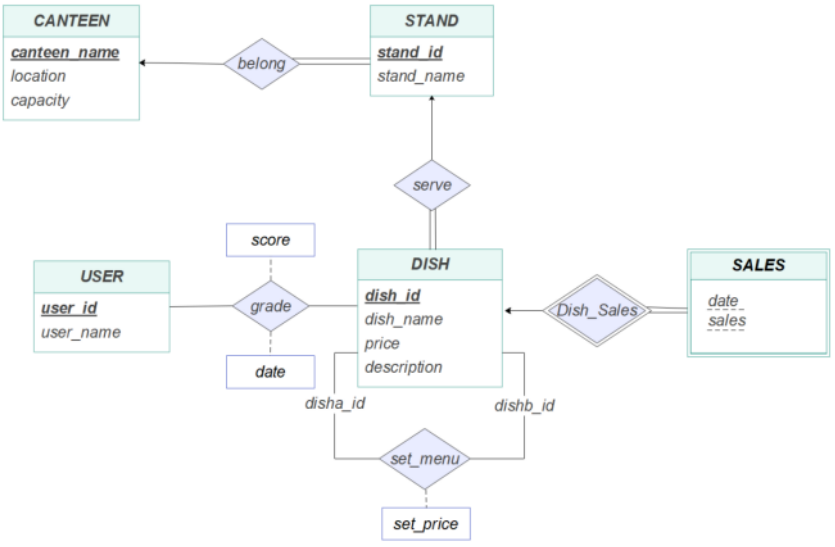

Figure 1: E-R diagram of our database

The design is under three premises according to the facts in CUHK-SZ: no

canteen has same name, a dish must served by a stand and a stand must belongs to a canteen.

The core entity set DISH is a table for dish data. Each dish has a unique dish_id as primary key. Two dishes may have the same name, so attribute dish_name cannot be a primary key. We assume each dish has to be served by a stand, so an attribute called stand_id will be added and act as foreign key referencing STAND when implementation. The description attribute is a description about dish's flavour. Assuming no canteen shares same name, the entity called CANTEEN has canteen_name attribute as primary key. Each canteen has its location and capacity. Since this is a system specially for CUHKSZ, the value for location only need to include the description about its location in school.

The entity set STAND is a table for stand data. Each stand has a unique stand_id as primary key. We assume every stand must belongs to a canteen, thus there will be a non-empty attribute called canteen_name act as foreign key referencing CANTEEN when implementation. Each stand has its own name. Two stands may have the same name, so attribute stand_name cannot be a primary key.

USER is a table that stores user data who registers an account. The primary key user_id is a numerical string which correspond to users' student ID or staff ID in school. Since student ID and staff ID is unique, user_ID must be unique. Attribute user_name is VARCHAR type attribute with limit 20.

SALES is a weak entity set with identifying entity set DISH. The discriminator attributes date and sales shows the number of transaction for a dish on a specific date.

Grade is relationship set which shows users' comments on a dish. A user can grading different dishes on different date, and a dish can receive different scores from different users, thus to specify a grade relation, we need to use user_id, dish_id and date as primary key. Grade also has non-empty descriptive attributes date and score to describe a grade relation.

Set_menu is a relationship set where disha_id and dishb_id act as roles. Both disha_id and dishb_id are correspond to a dish_id in DISH, and they need not to be distinct. Noting that we use a trick here, dishb_id can be empty, corresponding to the case that stand have some specific commodity on sales. However, since primary cannot be null, we set a special dish with special dish_id to act as dish B when implementation, and having this special dish act as dish B represent discount for dish A.

In our design, we use BCNF form that for all tables. The relation schemata are shown in Figure 2 and 3.

DISH (dish_id, stand_id, dish_name, price, description)
CANTEEN (canteen_name, location, capacity)
STAND (stand_id, canteen_name, stand_name)
USER (user_id,user_name)
SALES (dish_id, date, sales)
grade = (user_id,dish_id, score , date)
set_menu = (disha_id,dishb_id, set_price)

Figure 2: Relation schema

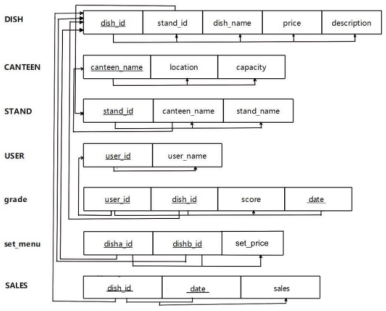

Figure 3

```
db_config = {
    'host': 'localhost',
    'user': 'root',
    'password': 'your_password',
    'db': 'cuhk_sz',
    'charset': 'utf8mb4',
    'cursorclass': pymysql.cursors.DictCursor
}
```

Figure 4

3 Implementation and Simple Queries

3.1 General Design

Based on the designed schema, we devised an operational food rating system that primarily emphasizes Flexibility, Transaction Support and Interactivity.

Flexibility: We use only Python to construct this project, include data mining part. We try our best to package each part of our project to enable easy modification and extension.

Transaction support: In the system implementation, we also emphasize the importance of transaction support. For this purpose , we design a well_defined transaction_supported procedure. For details,please refer to our codes.

Interactivity: Interactivity plays a crucial role in enhancing user engagement and facilitating database interaction in web applications. Our approach to achieving interactivity involves four key steps: packaging SQL queries as functions, integrating these functions into the web application, transforming them into user-friendly operations, and enabling easy database interaction via the web. Users can execute database operations with just a few clicks or inputs, without requiring direct SQL knowledge or access to database management tools.

3.2 Database Implementation

3.2.1 Introduction

This part we mainly focus on database initialization ,table creation, data insertion, and the management of related data dependencies. So we can manage information about dishes, canteens, stands, and related sales data within a university setting.

3.2.2 Database Connection and Configuration

The script (Figure 4) utilizes pymysql to establish a connection to a MySQL database. Connection parameters such as host, user, password, database name, and charset are specified in a configuration dictionary. This ensures a modular and secure approach to database connectivity.

3.2.3 Table Initialization and Truncation

We then create tables following the schemas in section2 .Each time we want to initialize and write data into tables, the script(Figure 5) ensures that the tables are clean by truncating existing records. This is crucial in scenarios where the database is refreshed with new data, such as in testing or periodic updates.

```
cursor.execute("TRUNCATE TABLE cuhk_sz.grade;")
cursor.execute("TRUNCATE TABLE cuhk_sz.DISH;")
cursor.execute("TRUNCATE TABLE cuhk_sz.STAND;")
cursor.execute("TRUNCATE TABLE cuhk_sz.CANTEEN;")
```

Figure 5

3.2.4 Data and Insertion

We collected rating data via questionnaires and used the Faker library to generate realistic but fictitious sales data . Then we inserted these data into table we (an example of data insertion for DISH table is shown in Figure6) .

```
for stand_id, dish_name, price, description in some_generated_list:
    cursor.execute("INSERT INTO DISH (stand_id, dish_name, price, description) VALUES
(%s, %s, %s, %s);", (stand_id, dish_name, price, description))
```

Figure 6

3.2.5 Dynamic Data Handling

The script (Figure 7) dynamically handles data with respect to dates and associated sales figures, demonstrating an advanced use of Python's datetime and random libraries. Sales data is linked to specific dates and dishes, illustrating a typical use case in business analytics.

```
for dish_id in dish_ids:
    random_date = random.choice(date_list)
    sales = int(random.betavariate(alpha, beta) * multiplier)
    cursor.execute("INSERT INTO cuhk_sz.SALES (dish_id, date, sales) VALUES
(%s, %s, %s);", (dish_id, random_date.strftime('%Y-%m-%d'), sales))
```

Figure 7

3.3 Web and Query Implementation

3.3.1 Introduction

The core idea of our web design is integrating SQL queries as functions on a web platform to simplify user interactions with the database. Just as mentioned previously, we achieve this by packaging SQL queries into functions and integrating these into a web interface. The queries we packaged are shown in Figure 8, and Figure 9 is an example of fetching dish details. Then we can directly use these functions

in our web construction file. In the following sub_sections of 3.3 we will show the some details of our web.

```
#Select information for snapshot
select dish_id, score from grade

#Search food rating by ID in descending order of time
SELECT user_id, score, date FROM grade WHERE dish_id = id ORDER BY date DESC;

# Rate for the food
INSERT INTO grade (user_id, dish_id, score, date) VALUES ('{user_id}', '{dish_id}', {score}, '{format_date}');

# Search information by ID
SELECT * FROM table_name WHERE key = id
```

Figure 8

```
def get_dish_details(dish_name):
    query = "SELECT * FROM DISH WHERE dish_name = %s;"
    cursor.execute(query, (dish_name,))
    return cursor.fetchall()
```

Figure 9

3.3.2 Authentication

A user login system is often the initial interaction point between a website and its users (show in Figure 10). In our system, users can register and undergo login verification using their account credentials. Additionally, users can retrieve their passwords through their registered email. During the verification process, passwords are encrypted using a predetermined hash function and compared to those stored in the database, thereby enhancing password storage security.

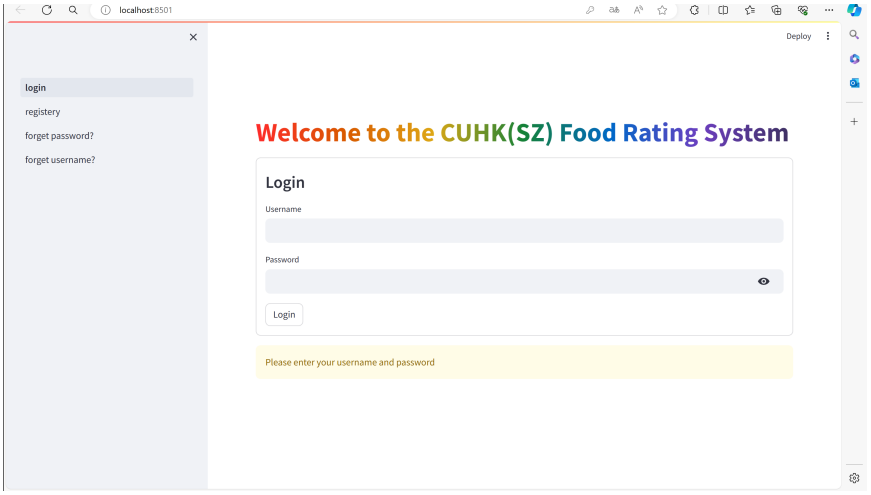

Figure 10

3.3.3 Snapshot

To present food information more intuitively, we choose bar chart snapshots. Utilizing SQL code, we've selected the top five foods based on both their ratings and prices (shown in figure 11). Subsequently, it is also possible to display different combinations according to various needs, such as lowest prices, lowest ratings, and so forth.

Figure 11

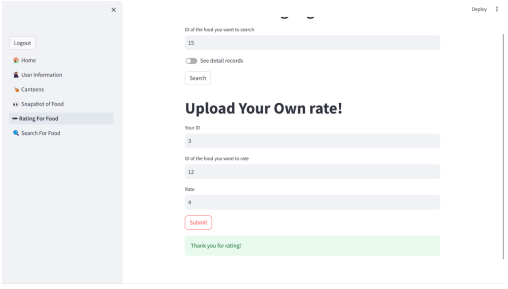

Figure 12

3.3.4 Rating

Rating food is one of the main functions and features of this system. Firstly, users can query the average rating of a food by entering its ID. Simultaneously, users can input a password to view specific ratings for the food, including rating dates and users (show in Figure12). Furthermore, users can rate specific foods by inputting their user ID and the food's ID. The rating results will be stored in the database

immediately. The next time when the food's rating is queried, the rating will be automatically updated and displayed.

3.3.5 Dish Rating Search

In addition to querying food ratings, this system also supports inquiries for other information (shown in figure 13 and 14). For instance, users can input stall IDs to retrieve stall information or select cafeteria names to view overall cafeteria details, such as seating capacity and stall locations.

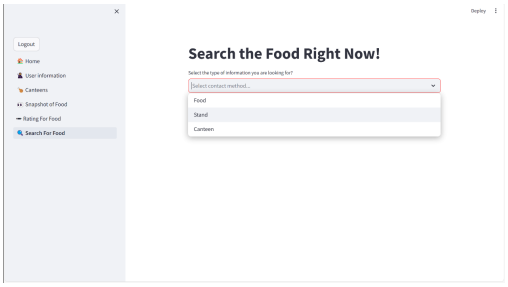

Figure 13

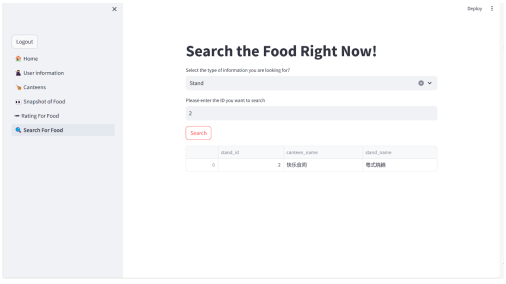

Figure 14

4 Data Mining

4.1 Data Mining Motivation and Target

In previous part, we have completed 'rating' task. We collected students' rating data via questionnaires and based on them we built a database. Now we want to mine these data and find useful patterns behind them. In detail, we aim to use machine learning methods to construct a dish recommendation system, so that we can provide personalized dish recommendations for each user.

Most recommendation mechanisms have a naive idea which is similar to 'association' discussed in lectures. In our canteen dish case, it is finding dishes that are rated high by 'similar' users, if a 'new' such user likes such dish, suggest the others to him/her.

4.2 Data Mining Methodology

The biggest two problems most recommendation mechanisms faced is how to find 'similar' user groups and how to predict the a user's preference of a unrated dish (can be important in recommendation order). In this project we use matrix decomposition

to solve these two problems at the same time. Specifically, we learn feature vectors for each user and dish.

We first use SQL queries to extract data from 'grade' relation in our database, and form them into so_called user-item matrix R. Just as Figure15 shows if we have m users and n dishes, then there are m rows and n columns in R, each row represents a user and column represents a dish. In R, (i,j) entry shows the rating of user i to dish j, if user i has not rated on dish j we mark the entry as 'x' or 0.

Then we decompose matrix R into two matrices: P and Q , such that the inner product of i_th column in P and j_th row in Q quantifies the preference of user i for dish j, as shown in Figure15. Notice in Figure15, k represents the potential dimension of user/dish feature vectors, which can show the representation ability in some sense.

$$R \approx P \times Q^T = \hat{R}$$

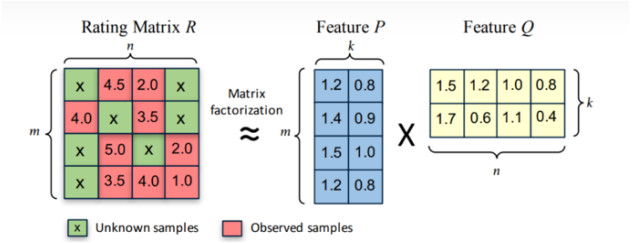

Figure 15

Finally, we can predict the preference or potential rating of user i to dish j by inner product of feature vectors, shown in Equation(1).

$$max_{p^*,q^*} \sum_{(u,i)\in M} (r_{ui} - p_u^T q_i)^2 + \lambda(||p_u||_2^2 + ||q_i||_2^2) \tag{1}$$

4.3 Model Training

The core idea of model training is that we want the inner products of existed rating pairs to be close to the real ratings. At the same time, we do not want to see overfitting, so our training objective function is shown in Equation(2).

$$\hat{r}_{ui} = p_u^T q_i = \sum_{k=1}^{K} p_{uk} q_{ki} \tag{2}$$

In practice, we use mini_batch SGD(stochastic gradient descent) method to optimize the objective function. Table1 is our training data statistics and Figure16 is our training loss per epoch.

Table 1

Dataset	#Users	#Items	#Interactions	Density
CUHKSZ Food	142	50	710	10%

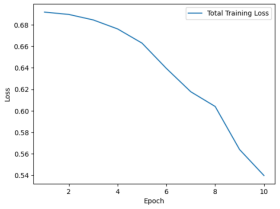

Figure 16

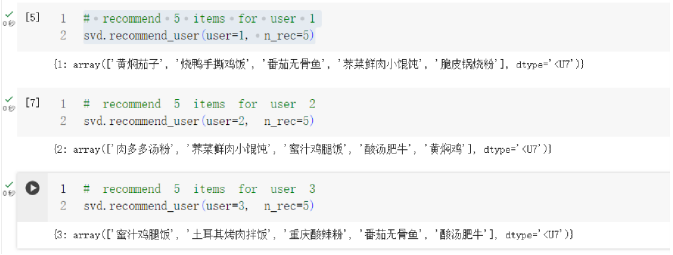

Figure 17

4.4 Recommendation Examples

After training, we show three recommendation examples here. Figure17 is our dish recommendations for user 1,2,3.

5 Interaction with LLM

In this project , we interacted with large language model ChatGPT3.5 to let it help us improve data access efficiency and generate accurate SQL queries.

5.1 Improve Data Access Efficiency

In this part, we gave GPT our database schema and asked how can we improve the data access efficiency. Then based on the suggestions GPT answered us, we did two changes to our original database. This interaction is shown in Figure18.

First, the core of our database is ‘grade’ relation and we need to access it in most of our parts. When designing website ,we need to query on it to get top 5

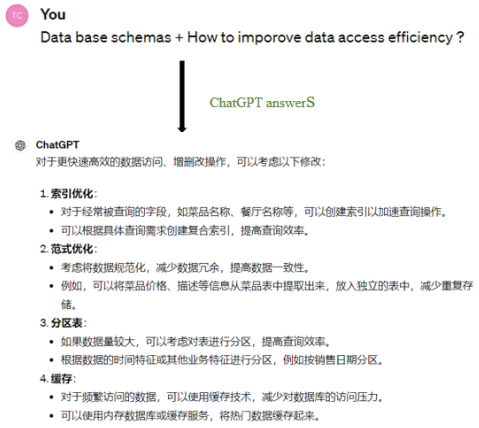

Figure 18

```
# Creat index
CERAT INDEX idx_dish_id ON grade(dish_id);

# Partition relation
PARTITON BY RANGE (YEAR(date))(
    PARTITION p0 VALUES LESS THAN (2023),
    PARTITION p1 VALUES LESS THAN (2024),
    PARTITION p2 VALUES LESS THAN MAXVALUE
);
```

Figure 19

rating dishes, in data mining part we need to extract user-item matrix from it. So constructing an index on attribute 'dish_id' in relation 'grade' may accelerate our access on the relation and thus the whole process of our project.

Second, as time progresses, the 'grade' and 'sales' relations are expected to grow in size continuously. We hope our database has a good ability of expanding, so we partitioned relations 'sales' , 'grade' by attribute 'date'.

The SQL pseudo-code of the two changes are shown in Figure19.

5.2 Generate Accurate Queries

We want GPT can generate accurate queries on our database when we input query demand or description in human language. However, due to the ambiguity of human language, achieving accuracy was challenging. What's more, we can not guarantee that the queries generated by GPT can execute correctly on our database.

To solve these problems, we use a model called 'self-correction model' to generate prompts automatically which can guide GPT generate better queries. Figure20 shows the work flow of this model.

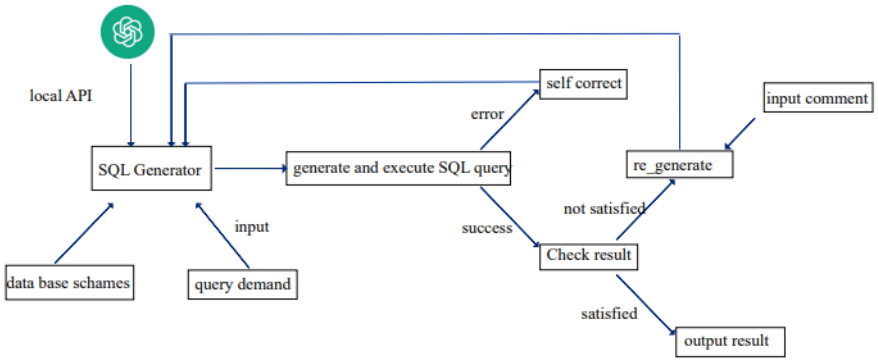

Figure 20

Specifically, in self-correction model, we connect to ChatGPT via API and designate it as the SQL Generator. We provide the database schema and describe our query demands or descriptions in human language, allowing the SQL Generator to generate and execute SQL queries. If an error occurs, we feed the error information back to the SQL Generator and repeat the process. If success, we obtain the query results for human verification. If the user is unsatisfied, we provide feedback to the SQL Generator and iterate again. Once the user is satisfied, we output the final result.

6 Conclusion

Food rating and recommendation systems have proven to be helpful and practical in various applications, such as Meituan. In this project, we have implemented basic functions of a canteen dish rating and recommendation system for CUHK(SZ). We hope our project can help in providing users with reliable references through precise ratings and user comments .