COMS W 4111 Project 1 Proposal

Xiaoyang Song & Han Liu

FA 2022

1 Domain & Short Introduction

Our project focuses on the domain of restaurant inspections and ratings at New York City, and the source dataset is obtained from the website NYC Open Data. In our project, we aim to first create a database where all relevant data about NYC restaurant inspections and ratings will be stored for users to browse; then we desire to model user information and their reviews to the restaurants, which will be added interactively and dynamically to our database by users. With that being said, our web application will also have login function. In general, by completing these goals, we intend to provide our users an opportunity to interactively review the restaurant recent government inspection results, violation records, and ratings and add their favorite/disliked restaurants to their preference/hate list. For example, if users want to know something about ”Chipotle”, then all the reviews, inspection records, ratings, and violations associated with ”Chipotle” can be retrieved for users to read; furthermore, they can add ”Chipotle” to their list of favorite restaurant or write a review for ”Chipotle” if they want to. In addition, our web application is useful for different groups of people. For instance, government staffs and inspectors may use this application to check which restaurants have violated certain regulations before when determining their inspection plan and target; for researchers and data scientists, they may use this application to investigate which type of inspection can find violations the most and make some statistical inference based on that. As for our database design, we currently have 7 entity sets and 8 relationship sets with bunch of constraints. For example, *Restaurant*, *Reviews*, and *Users* are three of the entity sets we relied on. Below are some relationship examples that our model can effectively capture:

- A *User* can **Like** any number of *Restaurant*.
- A *User* can **Post** any number of *Reviews* to a certain *Restaurant*.
- A *Review* can not exist without a *User*.

Other entity sets we made are *Locations*, *Inspection*, *Grades*, and *Violations*. With these entity sets and relationship sets, we can accomplish our goals perfectly. The detailed explanation of ER modeling can be found on the next page, and the revision made after TA meetings can be found in Appendix A.

2 ER Diagram Design

2.1 Introduction to Entity Sets

In our design of the database, we include 7 entity sets in total: **Users**, **Restaurant**, **Locations**, **Review**, **Inspection**, **Grade**, and **Violation**. A detailed description and design logic are introduced below:

1. **User: User** keeps track of the information of users to our web-app.
    - *user_id* (int): globally unique primary key.
    - *account_name* (text) [not null]: users' account name.
    - *passcode* (text) [not null]: users' passcode.
    - *dob* (DATE): date of birth.
    - *district* (text): the district that the user belongs in NYC. Note that the value of this attribute can only be one of the NYC district set[1]. We keep track of this so that we can send recommendation to the users based on their geographic location.
2. **Restaurant: Restaurant** keeps track of the information of restaurant.
    - *r_id* (int): globally unique primary key.
    - *DBA* (text) [not null]: the acronym (i.e. name) of the restaurant.
    - *phone_number* (VARCHAR(10)): the phone number of restaurant. Note that we only focus on NYC restaurant so a 10-digit phone number is enough.
    - *cuisine* (text): description of the cuisine.
3. **Location:**
    - *l_id* (int): globally unique primary key.
    - *district* (text): the value of this field must belong to the NYC district sets [1].
    - *zipcode* (VARCHAR(5)): zipcode of the location.
    - *street* (text): street of the location.
    - *building* (text): building of the location.
4. **Reviews: Reviews** contain users' review to restaurants.
    - *rev_id* (int): globally unique primary key.
    - *content* (text) [not null]: description of the reviews.
5. **Inspection: Inspection** keeps track of the inspection records of restaurants.

[1]NYC district sets = {'Manhattan', 'Bronx', 'Brooklyn', 'Queens', 'Staten Island'}

- *i_id* (int): globally unique primary key.
- *i_type* (text): type of the inspection.

6. **Grade: Grade** keeps track of the grades of restaurants.
   - *g_id* (int): globally unique primary key.
   - *grade* (CHAR(1)) [not null]: grade level. Note that the value of this attribute must belong to one of the grade set [1].
   - *score* (float): specific score.
7. **Violation:**
   - *v_id* (int): globally unique primary key.
   - *code* (text): government code that specifies the violation.
   - *v_description* (text): description of the violation.
   - *critical_flag* (text): note that this attribute must belong to the flag set [2].

2.2 Introduction to Relationship Sets

In total, we have 8 Relationship sets and many constraints imposed on them.

1. *Restaurant* **Locate** at *Location*: A restaurant can locate at exactly one location, while one location may hold multiple restaurant. One example for this is food court where multiple restaurant has the same location. Note that we treat franchise stores at different locations as different *Restaurant* entities.
2. *User* **Post** *Reviews*: A user can post many reviews; however, a review can be posted by only one user. Furthermore, when a user is deleted, the review should also goes away so we make review a weak entity of user. Furthermore, we can keep track of when the review is posted.
3. *Restaurant* **Own** *Reviews*: A restaurant can have many reviews; however, a review must be made to exactly one restaurant (We make this assumption to avoid reviews which contain multiple restaurants, which is not very informative in reality). In addition, when a restaurant is deleted, its reviews should also go away, so we make review a weak entity of restaurant as well [3].

[1] grade set = {'A', 'B', 'C', 'P', 'Z'}

[2] flag set = {'Critical', 'Not Critical', 'Not Applicable'}

[3] Note that *Review* is actually a weak entity of two entity sets, so it will be deleted on cascade if either *User* or *Restaurant* is deleted

4. *User* **Like** *Restaurant*: A user can like many restaurants, and a restaurant can be liked by many users. By keep tracking this relationship, we keep track of the restaurant that a user like; we can provide users list of restaurant they like in our web application easily by querying on this relationship sets in our database.

5. *User* **Dislike** *Restaurant*: A user can dislike many restaurants, and a restaurant can be disliked by many users. The intuition behind this relationship set is very similar to that of the **Like** relationship set.

6. *Restaurant* **graded** by *Grade*: A restaurant can have many grades, and a grade can be associated with many restaurants. We also track when this grading action happens.

7. *Restaurant* **inspected** by *Inspection*: A restaurant can have many inspections at any time described by inspection time, and an inspection can be done on multiple restaurants.

8. *Violation* is **Found** by *Inspection*: A violation can be found by multiple inspection, and an inspection can find any number of violations.

9. *Restaurant* **violate** *Violation*: A restaurant can have any number of violations, and a violation can be found in any number of restaurants.

Note that the last three relationship sets together enable the users to interactive query on any of them for useful information. For instance, for a given restaurant, a user can check whether it has any violation records and also its recent inspection records. In addition, users like the government staffs can also easily check which restaurants violate certain rules given an instance of violation. The combination of these three entity sets are very powerful in reality and can be used for different people including government staffs and normal citizens.

2.3 ER-Diagram

The ER-Diagram is provided on the next page. As mentioned in lecture, the ER-Diagram just aims to show the basic structure of the database so complicated check constraints are not shown on that. But to make things clearer, basic data types are drawn out.

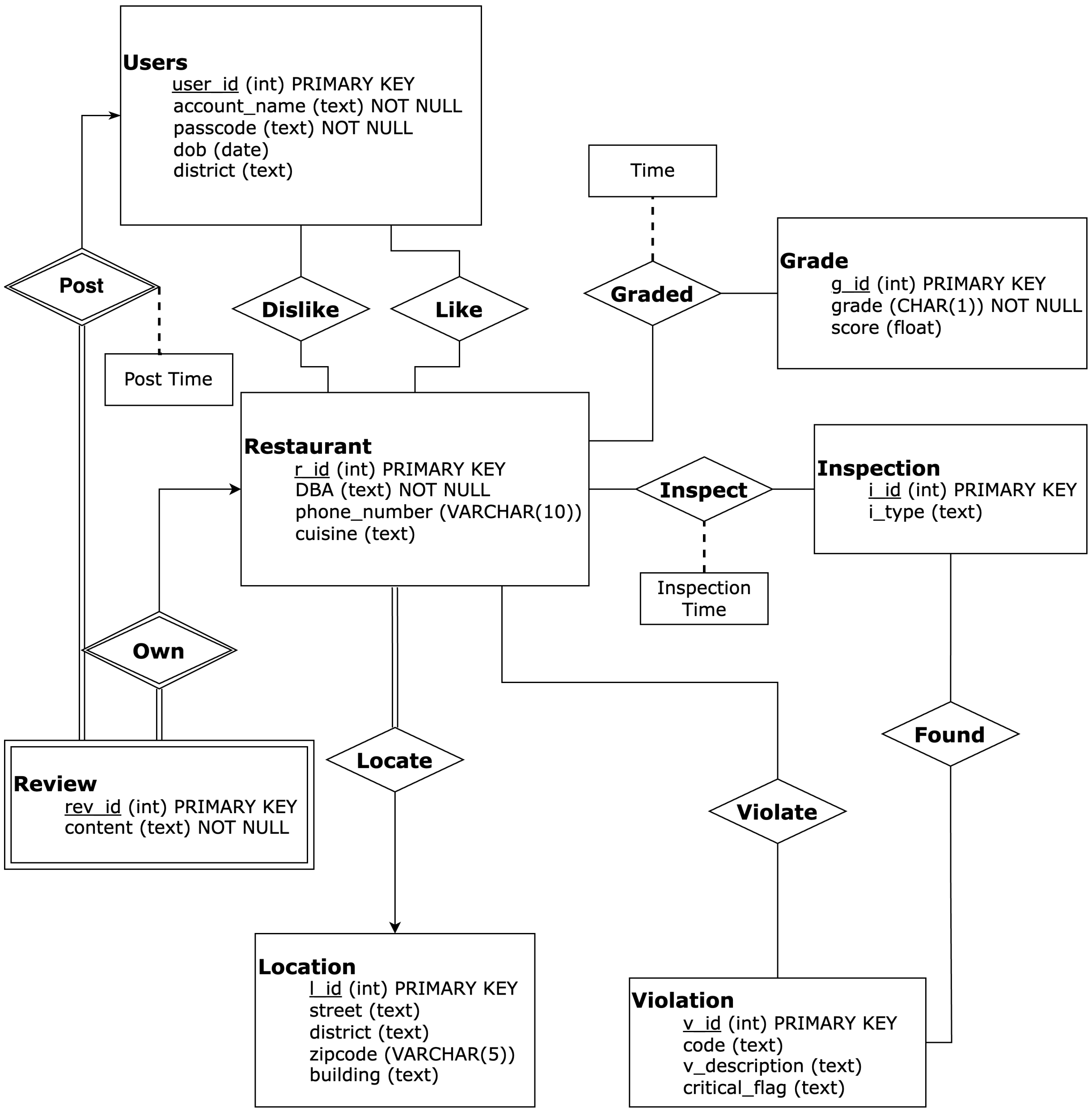


Figure 1: ER Diagram

3 Relational Schema Translation

There are several caveats about our translation:

1. When translating the **Like** and **Dislike**, we combine them into one to make things more efficient, but the functionalities are unchanged.

2. A very small portion of the variable names are not fully-consistent with Diagram because some variable names like ”Locate” are treated as keyword in SQL compiler that I used. However, the difference is very minor and our code makes it clear, so you should be able to find its correspondence in ER-Diagram easily.

The below are the code snippets (please check Figure 2 to Figure 6) for translating ER-Diagram to relational model. You can also visit our Github Repository to view this code. (https://github.com/Xiaoyang-Song/NYC-Restaurant-Inspection-Database/blob/main/p1.SQL)

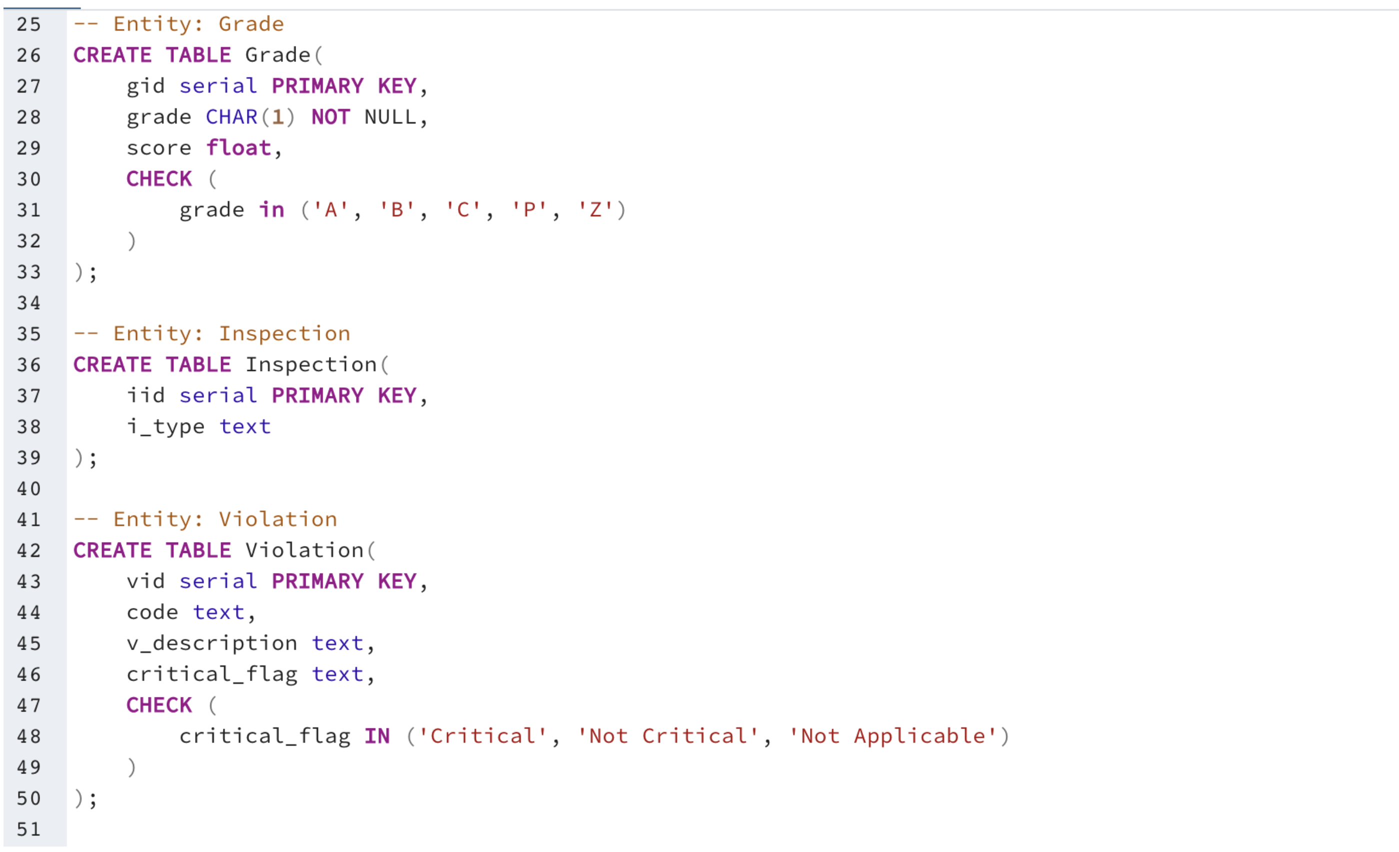
```sql
-- Project 1 Relational Model
-- Contributor: Xiaoyang Song, Han Liu

-- Entity: User
CREATE TABLE Users(
    userid serial PRIMARY KEY,
    account_name text NOT NULL,
    passcode text NOT NULL,
    dob date,
    district text,
    CHECK (
        district IN ('Manhattan', 'Bronx', 'Brooklyn', 'Queens', 'Staten Island')
    )
);

-- Entity: Restaurant
-- Note that for phone_number, we focus on US phone number only.
CREATE TABLE Restaurant(
    rid serial PRIMARY KEY,
    DBA text NOT NULL, -- DBA stands for "Doing Business As" (Acronym)
    phone_number VARCHAR(10),
    cuisine text
);

```

Figure 2: Relational Model



```sql
-- Entity: Grade
CREATE TABLE Grade(
    gid serial PRIMARY KEY,
    grade CHAR(1) NOT NULL,
    score float,
    CHECK (
        grade in ('A', 'B', 'C', 'P', 'Z')
    )
);

-- Entity: Inspection
CREATE TABLE Inspection(
    iid serial PRIMARY KEY,
    i_type text
);

-- Entity: Violation
CREATE TABLE Violation(
    vid serial PRIMARY KEY,
    code text,
    v_description text,
    critical_flag text,
    CHECK (
        critical_flag IN ('Critical', 'Not Critical', 'Not Applicable')
    )
);

```

Figure 3: Relational Model

4 Appendix A.

We made the following changes to our ER-Diagram after TA meetings:

1. Reconsider the functionality of our application: we change our designs such that our application is not only useful for normal customers but also for government staffs.
2. As for the ER-Diagram, we remove previous aggregation of the relationship sets **Inspect** between *Restaurant* and *Inspection* and remodel them with binary relationship sets. Removing unnecessary aggregation simplifies the structure and make things much clearer.
3. We changed the attributes of users to include more information like their locations, so that our web application can support more functionalities.
4. We made *Review* a weak entity set of two entity sets *Users* and *Restaurant*. This makes more sense than our previous design where we only made *Review* a weak entity of *Restaurant*.
5. There are also other changes but are not as significant as the previous four.

```sql
-- Entity: Locations
CREATE TABLE Locations(
    lid serial PRIMARY KEY,
    district text,
    zipcode VARCHAR(5),
    street text,
    building text,
    CHECK (
        district IN ('Manhattan', 'Bronx', 'Brooklyn', 'Queens', 'Staten Island')
    )
);

-- Combine relationship sets and entity sets to model weak entity
CREATE TABLE Reviews_Post_Own(
    rev_id serial PRIMARY KEY,
    content text NOT NULL,
    post_time DATE,
    userid int NOT NULL,
    rid int NOT NULL,
    FOREIGN KEY (userid) REFERENCES Users ON DELETE CASCADE,
    FOREIGN KEY (rid) REFERENCES Restaurant ON DELETE CASCADE
);

-- Relationship sets
CREATE TABLE Locates(
    rid int PRIMARY KEY REFERENCES Restaurant,
    lid int NOT NULL REFERENCES Locations
);

```

Figure 4: Relational Model

```sql
-- Relationship sets: note that when translating to relational schema
-- We combine Like and Dislike into Feel to make things more efficient
CREATE TABLE Feel(
    userid int NOT NULL,
    rid int NOT NULL,
    feel text NOT NULL,
    PRIMARY KEY (userid, rid),
    FOREIGN KEY (userid) REFERENCES Users,
    FOREIGN KEY (rid) REFERENCES Restaurant,
    CHECK (
        feel IN ('Like', 'Dislike')
    )
);

-- Relationship set: Graded
CREATE TABLE Graded(
    rid int NOT NULL,
    gid int NOT NULL,
    g_time DATE,
    PRIMARY KEY (rid, gid),
    FOREIGN KEY (rid) REFERENCES Restaurant,
    FOREIGN KEY (gid) REFERENCES Grade
);

-- Relationship Set: Inspect
CREATE TABLE Inspect(
    rid int NOT NULL,
    iid int NOT NULL,
    i_time DATE,
    PRIMARY KEY (rid, iid),
    FOREIGN KEY (rid) REFERENCES Restaurant,
    FOREIGN KEY (iid) REFERENCES Inspection
);
```

Figure 5: Relational Model

```sql
-- Relationship Set: Violate
CREATE TABLE Violate(
    rid int NOT NULL,
    vid int NOT NULL,
    PRIMARY KEY (rid, vid),
    FOREIGN KEY (rid) REFERENCES Restaurant,
    FOREIGN KEY (vid) REFERENCES Violation
);

-- Relationship Set: Found
CREATE TABLE Find(
    iid int NOT NULL,
    vid int NOT NULL,
    PRIMARY KEY (iid, vid),
    FOREIGN KEY (iid) REFERENCES Inspection,
    FOREIGN KEY (vid) REFERENCES Violation
);
```

Figure 6: Relational Model