COMS W 4111 Project 1 Progress Report

Xiaoyang Song (xs2485) & Han Liu (hl3608)

UNI used to create schema: xs2485

1 Modifications

First of all, at this point, we do not change our applications and the functionalities that we introduced in proposal. However, after starting the implementation of data extraction and insertion, we made the following changes to our ER-Diagram and relational model schema, and add some real-world assumptions that we rely on:

1.1 ER-Diagram

1. We **deleted** the relationship set **Found** between entity set *Inspection* and *Violation*. We made this change because this relationship is redundant according to our ER-Diagram; in another word, removing this relationship set will not reduce any functionality of our model. As Professor Wu mentioned in lecture, everything that can be computed should not be included in the diagram.

   The main reason we can remove this is that *Inspection* and *Violation* are indirectly related by *Restaurants*; with that being said, we can perform everything between them by joining these three entity sets and the corresponding relationship sets together. **Found** is thus not necessary.

2. We **added** an attribute v_time for relationship set **Violate** to record when this violation happens. In our design, this is the same as its corresponding inspection time, which are recorded by i_time in **Inspect**.

3. Note that the name of attributes show up on the ER-Diagram may not be exactly the same as the one we used during implementation. This is because we want to make the ER-Diagram more informative while keep the implementation easy.

1.2 Relational Model Schema

1. We **added** a not null attribute and foreign key reference lid to the schema of *Restaurant* so that it can capture the exactly-one constraint precisely. Correspondingly, we **deleted** the table **Locate** because we already include all information in table *Restaurant*. However, we choose not to change the name of the relation *Restaurant* when we combine them because this will make everything more informative.

2. We **changed** the primary key of relationship set **Inspect** to be tuple (rid, iid, i_time). We made this change because a *Restaurant* can receive the same type of *Inspection* for multiple times. However, our previous schema with only (rid, iid) does not allow this. Adding inspection time into the primary key enable us to capture this property.

3. Following exactly the same logic as for **Inspect**, we **changed** the primary key of relationship set **Violate** to be tuple (rid, vid, v_time).

4. Following exactly the same logic as for **Inspect**, we **changed** the primary key of relationship set **Graded** to be tuple (rid, gid, g_time).

1.3 Assumptions

When manually inspecting the source data and extracting/inserting them into database, we made the following assumptions, which are realistic and do not contradict with our design. **Note that the assumptions that are made in project proposal still holds.**

1. A *Restaurant* will not be **inspected** by the same *Inspection* type twice at the same day.

2. A *Restaurant* will not be **graded** twice at the same day.

3. A *Restaurant* will not **violate** by the same *Violation* twice at the same day.

4. These three are just some template real-world assumptions that we tend to make for our project. More details will come out as we fully finish our implementations.

2 Partial Design Choice

When extracting the data from source (.csv file) and insert them into the database, we found the following design choices useful and helpful for downstream tasks which we want to perform. Note that those choices do not violate our assumptions.

1. For record with district not in NYC district set[1], we remove the record and do not insert them into databases. These are clear outliers. There is no need to deal with missing values because all records in source file have district information.

2. For attributes in the tables which do not have not null constraints (i.e. optional attributes), if their values are not provided in the source data, we manually assign some informative values for them to indicate this fact. For example, $cuisine$ is an optional attribute in the table *Restaurant*, when its value is not provided in the record, we put "No description is provided" to the table. Note that this is more informative than a NULL.

3. For records whose restaurants acronym is not provided, we do not remove it and instead assign "Unknown" as their names. This prevents the information from being lost.

4. When extracting grades data and insert them into table *Grades* and **Graded**, we omit the grading records with missing g_time; this is because grading time is crucial for our application and g_time is a part of the key of table **Graded**.

5. More design choice will come out as we actually start implementing the web app.

3 ER-Diagram and Relational Model Schema

Please refer to the Appendix for latest ER-Diagram and the CREATE TABLE code blocks.

4 Populating Tables

We implement the pipeline to extract data from source file and then automatically insert them into the database with edge case handling. Therefore, for this progress report, we extract 100 records (100/65499 rows in the source file) and insert them into databases for table *Restaurant*, *Locations*, *Grade*, *Inspection*, *Violation*, **Graded**, **Violate**, and **Inspect**.

However, for tables *Users*, *Reviews_Post_Own*, and **Feel**, because we do not have records for them from source file, we **manually insert 10 or more records** for each of them to illustrate the functionality of our database. **In our design, these three tables will be gradually populated/updated as more *Users* create accounts, share their feelings, and post reviews.** For more details of data extraction/insertion and manual insertion procedure, please check here. And all source codes for extraction and insertion can be found on https://github.com/Xiaoyang-Song/NYC-Restaurant-Inspection-Database.

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

5 Interesting Queries

1. Find the "Doing Business As" (i.e. dba) of Restaurants, along with their rid, that are in 'Manhattan' and have rating level (i.e. $Grade$) of 'A'. (no duplicates)

```
SELECT R.rid, R.dba
FROM Restaurant AS R, Graded as GR, Grade as G
WHERE R.rid=GR.rid AND GR.gid=G.gid AND G.grade='A'
INTERSECT
SELECT R.rid, R.dba
FROM Restaurant AS R JOIN Locations as L on R.lid=L.lid
WHERE L.district='Manhattan'
```

2. Find the distribution of inspection types (i.e. i_type), that is, find the total number of inspections for each inspection type. The final results should be ordered in descending order according to the total number of inspections for each type. The output of this query should be several rows with two columns (i_type, count).

```
SELECT I.i_type, COUNT(*) AS count
FROM Inspection as I, inspect AS IR, Restaurant AS R
WHERE I.iid = IR.iid AND R.rid = IR.rid
GROUP BY I.i_type
ORDER BY COUNT(*) DESC
```

3. Find the "Doing Business As" (i.e. dba) of restaurants, along with their rid and total number of positive reviews, which have the highest number of positive reviews (assume there is no tie). For simplicity here, a review is positive if and only if it contains the word 'good'. (We do not consider restaurant without reviews) The output of this query should be a single row with three columns.

```
SELECT R.rid, R.dba, COUNT(*)
FROM Restaurant AS R, Reviews_Post_Own AS REV
WHERE R.rid=REV.rid AND REV.content LIKE '%good%'
GROUP BY R.rid, R.dba
ORDER BY COUNT(*) DESC
LIMIT 1
```

Note that these three queries are designed based on real-world applications. The detailed results can be found at the last section of this notebook.

6 Appendix

6.1 ER-Diagram

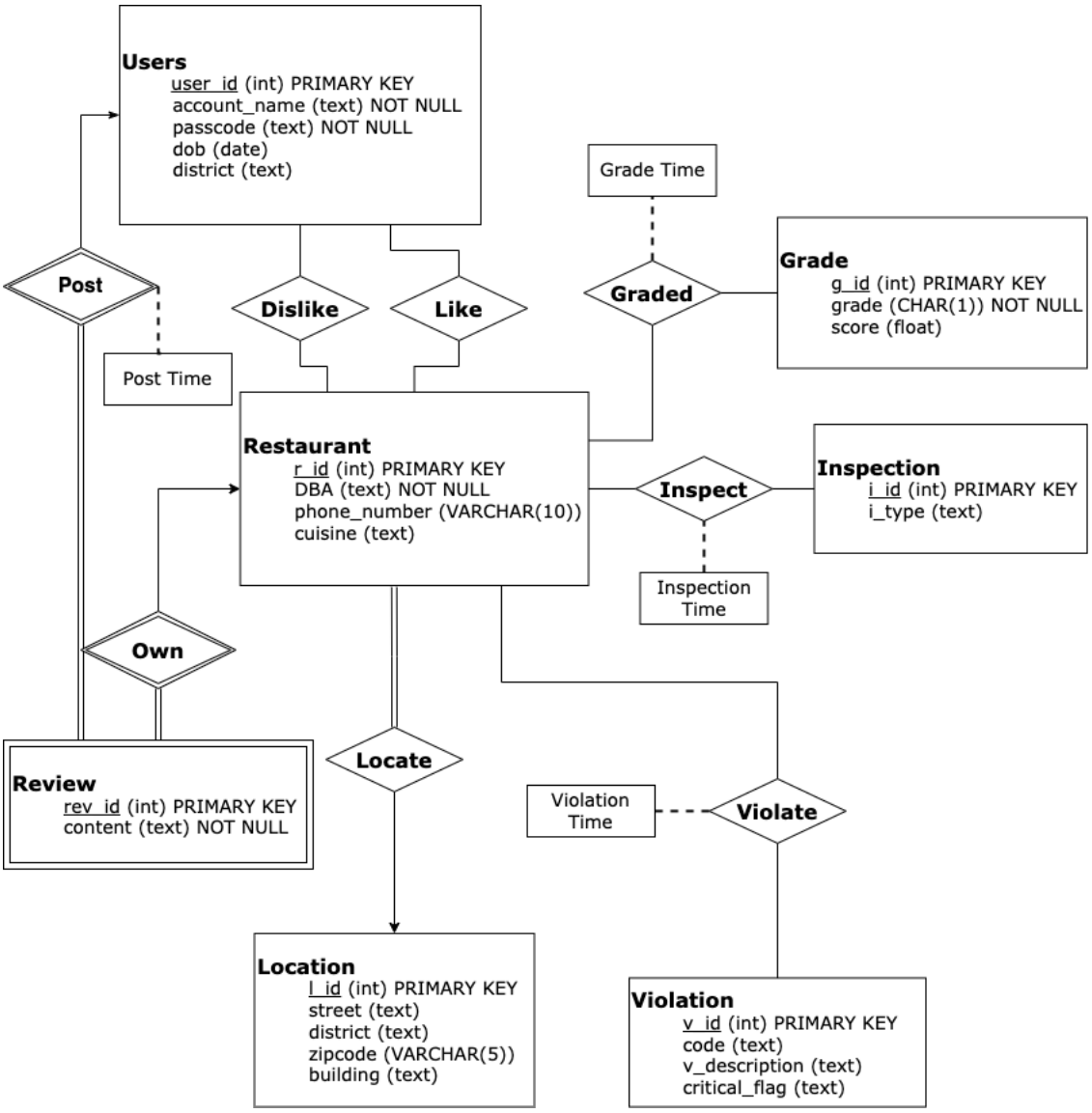

Figure 1: Latest ER-Diagram

6.2 CREATE TABLES

```
-- Entity: Users
CREATE TABLE IF NOT EXISTS 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: Locations
CREATE TABLE IF NOT EXISTS Locations(
    lid serial PRIMARY KEY,
    district text,
    zipcode VARCHAR(5),
    street text,
    building text,
    CHECK (
        district IN ('Manhattan', 'Bronx', 'Brooklyn',
        'Queens', 'Staten Island')
    )
);

-- Entity: Restaurant
-- Combine relationship sets and entity sets to model exactly-one
CREATE TABLE IF NOT EXISTS Restaurant(
    rid serial PRIMARY KEY,
    DBA text NOT NULL,
    phone_number VARCHAR(10),
    cuisine text,
    lid int NOT NULL,
    FOREIGN KEY (lid) REFERENCES Locations
);

-- Entity: Grade
CREATE TABLE IF NOT EXISTS Grade(
    gid serial PRIMARY KEY,
    grade CHAR(1) NOT NULL,
```

```
    score float,
    CHECK (
        grade in ('A', 'B', 'C', 'P', 'N', 'Z')
    )
);

-- Entity: Inspection
CREATE TABLE IF NOT EXISTS Inspection(
    iid serial PRIMARY KEY,
    i_type text
);

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

-- Combine relationship sets and entity sets to model weak entity
CREATE TABLE IF NOT EXISTS 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
-- note that when translating to relational schema
-- We combine Like and Dislike into Feel to make
-- our design more efficient and elegant
CREATE TABLE IF NOT EXISTS 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 IF NOT EXISTS Graded(
    rid int NOT NULL,
    gid int NOT NULL,
    g_time DATE NOT NULL,
    PRIMARY KEY (rid, gid, g_time),
    FOREIGN KEY (rid) REFERENCES Restaurant,
    FOREIGN KEY (gid) REFERENCES Grade
);

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

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