CSCI620.1 Group2 Phase 1

Shuo Yao
Rochester Institute of Technology
sy9663@rit.edu

Yi Lan
Rochester Institute of Technology
yl8217@rit.edu

Haoran Sheng
Rochester Institute of Technology
hs1911@rit.edu

Jack Smith
Rochester Institute of Technology
jss7268@rit.edu

1. DATA MODELING

This part justified the selection of data from the Yelp dataset and presented the design of the ER model.

1.1 Data Selection

Our original Data set has information including `business`, `Review`, `User`, `Check in`, `Tip` and `Photo`. After reading this Data Description of data set, we found that the more important among the information is `business`, `user` and `review`. After that, we made some modifications. For example: In `User` Data, because the `review count` can be generated from `review` table, so it is not required. Some attributes, such as `number of votes` and `number of fans`, was not considered to be included in the database since they are not related to our design. In `business` Data, `stars` will cause duplicate information because it can be obtained from the `star rating` attribute from `reviews`. In addition, some information like `latitude` and `longitude`, we think they are redundant information since we have already know their accurate location.

1.2 E-R Model Design

Initially, we only designed three entities including `business`, `review` and `user`. Later, we found that `review` table does not need to have its own unique ID, because the Primary Key of this entity can be `User ID`, `Business ID` and `date`. So, we deleted the unique ID of the `review`. Regarding the business part, we started out as a whole and did not classify it in detail. Later, after many discussions with the professor, combined with the advice given by the professor, we divided the entity into different categories. After researching the data, we specified three business categories: `food`, `shopping` and `beauty`. At this point, these three categories each had their own attribute `food type provided`, `shopping service type` and `Beauty service type`. But their attributes were considered to be too flat. To solve this problem, we use Python to run through all of the business data and then filter out the common and unique attributes among those category:

Unique attributes for `food` category:
RestaurantsReservations, number of occurrences: 66,671
RestaurantsDelivery, number of occurrences: 51,833
OutdoorSeating, number of occurrences: 51,632
RestaurantsGoodForGroups, number of occurrences: 51,385
Alcohol, number of occurrences: 45,349
GoodForMeal, number of occurrences: 29,913
RestaurantsTableService, number of occurrences: 17,168

Unique attributes for `shopping` category:
WheelchairAccessible, number of occurrences: 4,508
DogsAllowed, number of occurrences: 1,892

Unique attributes for `beauty` category:
AcceptsInsurance, number of occurrences: 1,670
HairSpecializesIn, number of occurrences: 1,006

Common attributes among `food`, `shopping` and `beauty`:
Parking, number of occurrences: 110,337
GoodForKids, number of occurrences: 51,123
BusinessAcceptsCreditCards, number of occurrences: 43,636

The other thing to be specifically pointed out is the `locate-in` relationship between businesses. This relationship considered the case that an airport or train station can have food and shopping, a shopping mall can have food, shopping and beauty, etc. To be mentioned, special location such as `airport`, `train station` and `shopping mall` are also in the business data. We queried out those businesses and queried other businesses having the same address with them and connect them together. The attribute location type was recorded and saved as an attribute. By doing this, user can find certain type of business in certain type of special locations more efficiently.
Figure 1 shows our final E-R model.

2. DATABASE ESTABLISHMENT

This part describes the creation and population of the database.

2.1 Data Import

We use JavaScript in SQL shell8.0 to parse the original data materials. The files are then stored in SQL server as native supported JSON document. We utilize the SQL server's ability to search within a JSON object to directly query its content and convert the query result into tables that we desire. As for the composite JSON object, we unwrap the data into JSON objects and normal columns for further process and analyses. After we have a satisfying table design, the redundant data are dropped from the table, whereas useful data are moved into our final working tables by `insert...select` queries. In the light of 'noises' in the

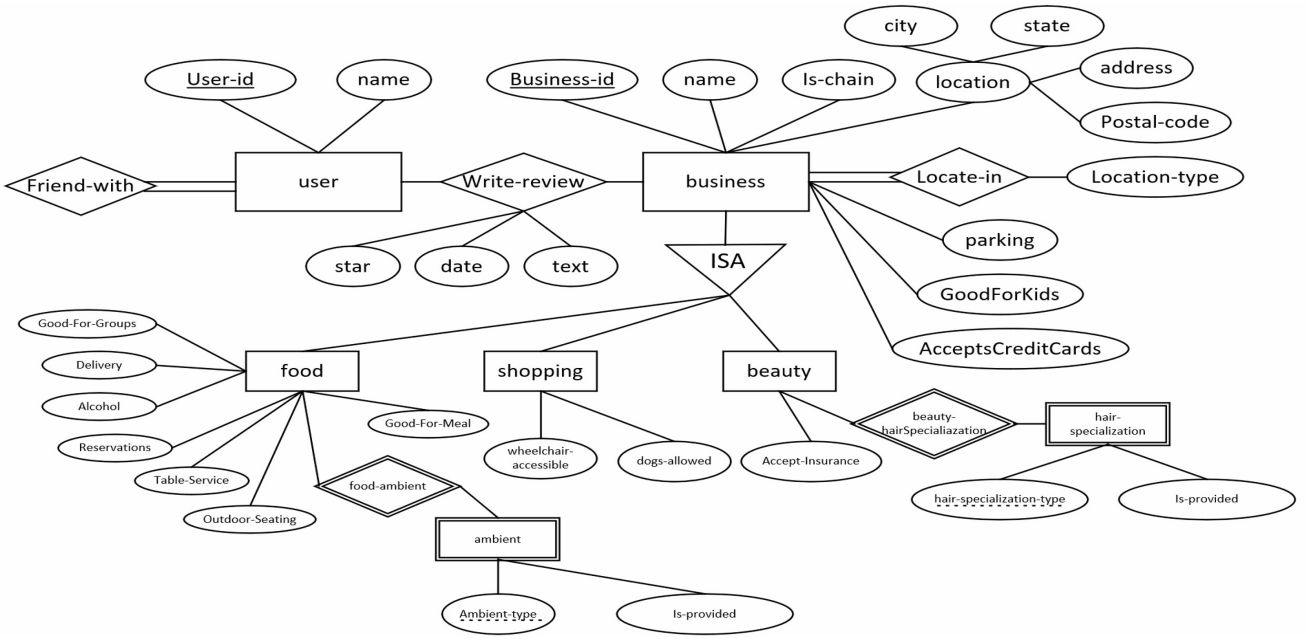

Figure 1: ER Diagram

data set, we first identify those by trying to create candidate keys in our table, those rows show complete duplication are considered as noises. Moreover, some extremely weird rows that display abnormal behaviors under our model are considered as noises. For example, a User data with an id but null in the name field is considered noise.

2.2 Partitioning

For some tables with a large amount of data (such as review table), we split it horizontally into multiple tables. Take the Primary Key to do the hash. For example, there are `Business ID`, `User ID`, `Date`, serve as primary key in the review table. Concatenate those fields together into a long string and use the hash function to find the table where the corresponding data is located. Because the hash function that is known to the SQL server, thus when a query is send to the partitioned table the server automatically knows which partition or partitions it should work on, if the given query searches among the primary index in some manner.

2.3 Normalization

The first step in normalization is to get our relationships into 1st Normal Form. To get the schema into 1st Normal Form, we also had to ensure that only single values were present in each row. This required splitting the categories of businesses into several 'is-a' relationships. These include `beauty`, `food`, and `shopping`, to start. The business attributes also needed to be turned into single-value attributes to get into 1NF. This required creating columns for different attributes in either the general business table, or in the specific 'is-a' tables. Many of these were simple boolean values, but some required the addition of weak entity sets. These were the `Ambience`, and `HairSpecializesIn` because they contained json defining whether a business had any of many attributes (such as a classy, romantic, casual, and/or etc. ambience).

To become 2NF, aggregate and duplicate values from the data were removed. This included values such as the number of stars that businesses have and the number of reviews that a user has. Storing these attributes. that can be calculated from other data. leads to inconsistencies that shouldn't be in the database. Since most candidate keys were unique id values, nothing needed to be done to ensure that there were no partial dependencies.

To become 3NF the transitive dependency of `business-name` determining whether a business was a chain had to be removed. This was achieved by creating a table that stored whether a business name was a chain, as chain status was calculated based on whether there were multiple distinct business with the same name.

The only overlapping candidate key was for the `business` table, wherein knowing the `business-name`, `address`, `state`, `city`, and `postal-code` would also be a super key that determined anything that the `id` itself would determine. To convert into BCNF, a new relation called `business-location` was created that separated location attributes from the rest of the business attributes. This made the `business` table smaller, and also converted it into BCNF at the same time. The `business-location` table uses the `id` as the primary key, and the non-key attributes are `address`, `city`, `state`, and `postal-code`.

3. NEXT STEP

Query scenario designing: We will implement the basic query scenarios that the professor will provide for the Yelp data set. In addition, we will also develop more complex query scenarios that aim to demonstrate the quality of our relational schema design. The query scenarios that we wish to create will involve multiple tables, filters, and aggregation.

Indexing: After we create our query scenarios, we will need to create indexes in our tables. We will measure the effectiveness of the indexes by comparing query speeds with

and without our indexes. Our goal is to create indexes that dramatically increase the speed of the queries that we will run to maximize the efficiency of our data management system.

Query system: The next step is to Develop a query system to allow users to access to the data. The system will have an user-friendly interface and support queries involving single-table queries, multiple-table queries, nested queries, and aggregation. We will be creating a Java application to cover the backend and frontend portions of the query system. The Java SQL connector is a great tool we can use to connect to our database, and the Java framework provides several tools to allow us to create a well-designed interface that will allow users to easily run several queries and interpret the results.

4. LIST OF DELIVERABLE

This section describes the list of deliverables in the submission. Please see README.md in each folder for more detailed information.

1. Folder name: MySQLscript

MySQL script for creating the schema and tables, and importing data.

2. Folder name: Python

Python files populating the normalized tables with data in order to complete the normalization process. The MySQL scripts in this directory provide a view of create table commands that were used to generate certain tables that these python scripts operate on.

3. Folder name: ERmodeling-Script

Python file which reads through all of the business data and then filter out the common and unique attributes among category 'food', 'shopping' and 'beauty'.