- Drop photo & hours Table
  - `DROP TABLE photo;`
  - `DROP TABLE hours;`

- Insert Preferred Food Categories using the SQL file
  - `source <FILEPATH>`

- Create Indexes for faster searches and updates
  - `CREATE INDEX businessIDIndex ON business(id);`
  - `CREATE INDEX categoryBusinessIDIndex ON category(business_id);`
  - `CREATE INDEX categoryIndex ON category(category);`
  - `CREATE INDEX userIDIndex ON user(id);`
  - `CREATE INDEX reviewUserIndex ON review(user_id);`
  - `CREATE INDEX reviewBusinessIndex ON review(business_id);`
  - `CREATE INDEX friendUserIndex ON friend(user_id);`
  - `CREATE INDEX friendFriendIndex ON friend(friend_id);`

- Copy of the category table
  - `CREATE TABLE copy_category AS (SELECT * FROM category);`

- Remove categories from the category table
  - `DELETE FROM category WHERE category.category NOT IN (SELECT * FROM temp);`

- Create useful business Table whose categories are in the desired categories.
  - `CREATE TABLE useful_businesses(business_id varchar(22));`
  - `INSERT INTO useful_businesses SELECT distinct business_id FROM category;`

- Add column to the business table to state whether the business is useful or not.
  - `ALTER TABLE business ADD is_useful TINYINT(1);`
  - `CREATE INDEX useful_businessIndex ON useful_businesses(business_id);`
  - `UPDATE business b SET is_useful = 1 WHERE b.id IN (SELECT * FROM useful_businesses);`
  - `UPDATE business b SET is_useful = 0 WHERE b.id NOT IN (SELECT * FROM useful_businesses);`
  - `DELETE FROM business WHERE business.is_useful = 0;`
  - `DELETE FROM business WHERE business.state IN ('01','6','3','C','B','30');`
  - `DELETE FROM useful_businesses WHERE useful_businesses.business_id NOT IN (SELECT id FROM business);`
  - `DELETE FROM category WHERE category.business_id NOT IN (SELECT id FROM business);`

- DELETE FROM tip WHERE tip.business_id NOT IN (SELECT id FROM business);
- DELETE FROM review WHERE review.business_id NOT IN (SELECT id FROM business);
- DELETE FROM checkin WHERE checkin.business_id NOT IN (SELECT id FROM business);
- DELETE FROM attribute WHERE attribute.business_id NOT IN (SELECT id FROM business);
- DELETE FROM user WHERE user.review_count < 2;
- DELETE FROM review WHERE review.user_id NOT IN (SELECT id FROM user);
- DELETE FROM friend WHERE friend.user_id NOT IN (SELECT id FROM user);
- DELETE FROM friend WHERE friend.friend_id NOT IN (SELECT id FROM user);

❖ User-Business Mapping

- CREATE TABLE user_business_mapping (user_id VARCHAR(22), business_id VARCHAR(22));
- INSERT INTO user_business_mapping SELECT user.id, business_id FROM review JOIN user ON review.user_id=user.id GROUP BY user.id, business_id;
- CREATE INDEX user_business_UserIndex ON user_business_mapping(user_id);
- CREATE INDEX user_business_BusinessIndex ON user_business_mapping(business_id);
- ALTER TABLE user_business_mapping ADD city VARCHAR(255), ADD state VARCHAR(255);
- UPDATE user_business_mapping AS ub SET city = (SELECT b.city FROM business AS b WHERE ub.business_id = b.id);
- UPDATE user_business_mapping AS ub SET state = (SELECT b.state FROM business AS b WHERE ub.business_id = b.id);

❖ User Location Mapping

- CREATE TABLE user_location (user_id VARCHAR(22), city VARCHAR(255), state VARCHAR (255));
- ALTER TABLE user_location ADD UNIQUE(user_id);
- INSERT IGNORE INTO user_location(user_id, city, state) SELECT user_id, city, state FROM (SELECT user_id, city, state, count(*) AS number FROM user_business_mapping GROUP BY user_id, city, state) AS T WHERE number = (SELECT max(number) FROM (SELECT user_id, city, state, count(*) AS number FROM user_business_mapping GROUP BY user_id, city, state) AS X WHERE X.user_id = T.user_id);

❖ User Location in User Table

- ALTER TABLE user ADD (city VARCHAR(255), state VARCHAR(255));

- ➢ `create index userlocationIndex on user_location(user_id);`
- ➢ `UPDATE user AS u SET u.city  = (SELECT l.city FROM user_location AS l WHERE u.id = l.user_id);`
- ➢ `UPDATE user AS u SET u.state  = (SELECT l.state FROM user_location AS l WHERE u.id = l.user_id);`
- ➢ `ALTER TABLE user ADD (is_elite TINYINT(1));`
- ➢ `CREATE INDEX eliteIndex ON elite_years(user_id);`
- ➢ `UPDATE user SET is_elite = 1 WHERE user.id IN (SELECT DISTINCT(user_id) FROM elite_years);`
- ➢ `UPDATE user SET is_elite = 0 WHERE user.id NOT IN (SELECT DISTINCT(user_id) FROM elite_years);`

❖ User-Category Mapping

- ➢ `CREATE TABLE user_category_mapping (user_id VARCHAR(22), category VARCHAR(255));`
- ➢ `INSERT INTO user_category_mapping SELECT * FROM (SELECT ub.user_id, c.category FROM user_business_mapping ub INNER JOIN category c ON ub.business_id = c.business_id) AS a GROUP BY a.user_id, a.category;`

❖ **SQL Queries for Exploratory Analysis**

➢ Under the review table, we want to count and compare the number feedback for those reviews (were they funny? or cool?)

```
SELECT stars, sum(useful) as useful, sum(funny) as funny,
sum(cool) as cool from review group by stars;
```

➢ This query is trying to identify users within a range of review count and also identify their friend network.

```
SELECT * from (SELECT * from friend limit 100000) as f Join
(SELECT id, name, review_count from user where review_count
&gt; 0 and review_count &lt; 50) as u on f.user_id = u.id ;
 Join user as u2 on u2.id = f.friend_id
```

➢ Identify the ratio of number of users to number of friends they have

```
SELECT count(distinct user_id) from (select * from friend
limit 100000 ) as f ;
```

➢ Get an idea of how many business are from a specific state

```
SELECT count (*) FROM business where state = ‘NV’ order by
stars DESC;
```

- Identify businesses that belong a certain category, are open and have a minimum number of reviews. This helps us identify if we have sufficient data for a given category

  ```
  SELECT * from business as b join category as c on
  b.id=c.business_id where state like "NV" and is_open = 1
  and review_count >= 50;
  ```

- The average number of friends that each user has. To understand the network. Split the friend table into multiple parts (of 50,000 each) to ensure safe execution of queries on sql server.

  ```
  SELECT round (avg(fin_frnCount)) as "Average Friends Count"
  from (select avg(frnCount) as fin_frnCount from (select
  count(friend_id) as frnCount from (select * from friend
  limit 0, 50000) as f1 group by f1.user_id) as final1
  UNION
  SELECT avg(frnCount) as fin_frnCount from (select
  count(friend_id) as frnCount from (select * from friend
  limit 50001, 100000) as f2 group by f2.user_id) as final2
  UNION
  SELECT avg(frnCount) as fin_frnCount from (select
  count(friend_id) as frnCount from (select * from friend
  limit 100001, 200000) as f group by     user_id) as final3
  UNION
  SELECT avg(frnCount) as fin_frnCount from
  (select count(friend_id) as frnCount from (select * from
  friend limit 200001, 300000)
  ```

- SELECT category from category where category ='Restaurants';
- SELECT distinct category from category;
- SELECT * from category;
- SELECT id,name,review_count,useful from user where useful > 50;
- SELECT user.id, friend.friend_id from user inner join friend on user.id = friend.id
  where user.useful >50;