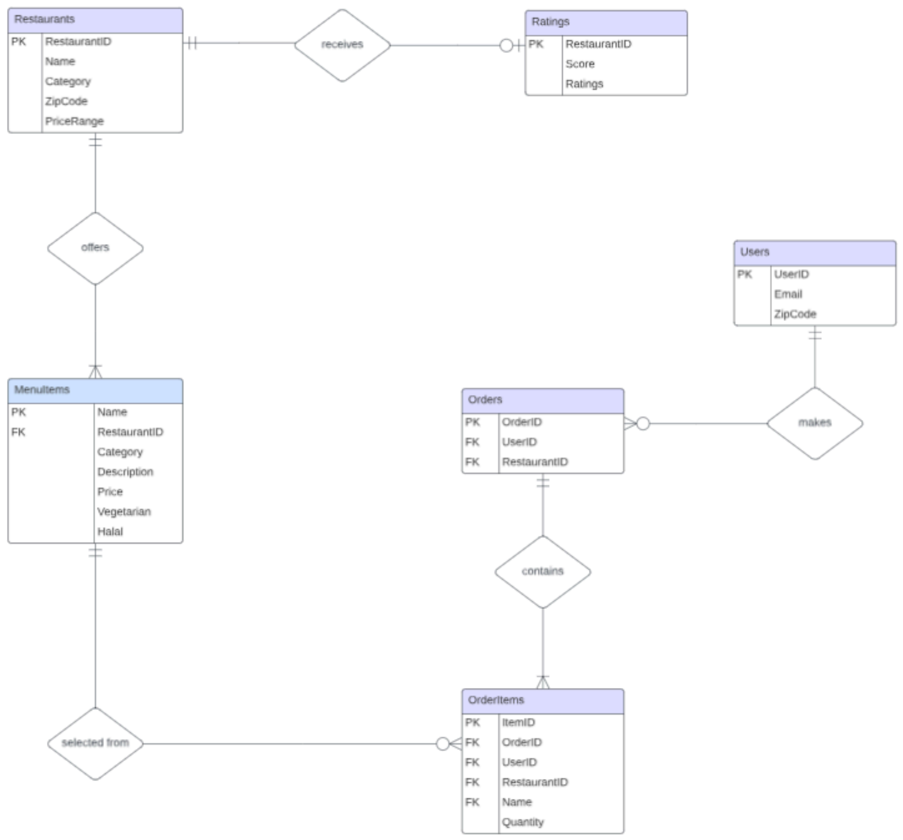

Link to UML (for better visibility): https://lucid.app/lucidchart/3c26b174-53e9-4d19-bb62-6e9a107f65a2/edit?viewport_loc=-57%2C-69%2C2848%2C1324%2C0_0&invitationId=inv_e7dc1679-0553-4335-bf9b-285b56e0ef59

BCNF

The data we procured from Kaggle was separated into Restaurants and restaurant-menu csv files. We split up the information into separate entities to meet the requirements and handle data more efficiently. By decomposing the origins dataset into the entities ensures that each table conforms to BCNF. Each table is designed such that every functional dependency has a superkey on the left hand side. The normalization process will help with data integrity and query performance.

SuperKeys : Attributes

Users: UserID → Email,Zipcode
Restaurants: RestaurantID → Name, Category, ZipCode, PriceRange, Ratings, Score
Ratings: RestaurantID → Score, Rating

MenuItems: Name → RestaurantID, Category, Description, Price, Vegetarian, Halal
OrderItems: ItemID → OrderID, UserID, RestaurantID, Name, Quantity
Orders: OrderID → UserID, RestaurantID

Entity Descriptions

Restaurants: Contains restaurant information
Ratings: Contains ratings for restaurants
MenuItems: Contains menu items for each restaurant
Orders: Contains orders from users
OrderItems: Contains information about what menu items are in an order
Users: Contains user information

Reasoning for Entities instead of Attributes

Restaurants: We chose to do this to show a brief summary of each restaurant in a condensed space. Every restaurant in the entity has almost all of the attributes for this table (ie not many null values). Attributes of the restaurant

Ratings: We chose to have Ratings as an entity instead of an attribute because many of the restaurants don't have ratings included. Instead of including it as an attribute within the Restaurants table, we created the Ratings table so that there were less cells with no data.

MenuItems: We chose to include MenuItems as an entity because they refer to the restaurants entity, but has a very large content since it includes every menu item of every restaurant. Therefore we wanted this to be an entity since making it into an attribute would be a lot of information.

Orders: We chose to have orders as an entity because it refers to the users table to confirm which orderID belongs to them.

OrderItems: We chose to create OrderItems as an entity because we needed a way to connect MenuItems and Orders so that an order has the capability to have multiple menu items on it instead of just one (since adding this onto the Orders table would result in only 1 item due to not being able to have the OrderID repeated multiple times). Therefore we created OrderItems as an entity to be able to reference multiple menu items within one order as well as the quantity of any given menu item within the order.

Users: We wanted to include users since they are able to have multiple orders and have a specific userID.

Relationship Assumptions

(* = any number of)

- Each restaurant can have exactly one rating - One-to-One
- A restaurant can have 1 to * menu items, but a menu item can belong to exactly one restaurant - One-to-Many
- A menu item can appear in 0 to * order items, but an order item refers to exactly one menu item - One-to-Many
- An order contains 1 to * order items, but an order item belongs to exactly one order - One-to-Many
- A user can have 0 to * orders, but an order belongs to exactly 1 user - One-to-Many
- An order corresponds to exactly one restaurant - One-to-One

Relational Schema

- Restaurants(RestaurantID: INT [PK], Name: CHAR[225], Category CHAR[225], ZipCode: VARCHAR(10), PriceRange: CHAR[225])
- MenuItems(Name: CHAR[225] [PK], RestaurantID: INT[FK to Restaurants.RestaurantID], Category: CHAR[225], Description: CHAR[225], Price: CHAR[225], Vegetarian: BOOLEAN, Halal: BOOLEAN)
- Ratings((RestaurantID: INT [PK], Score: DECIMAL, Ratings: DECIMAL)
- Orders(OrderID: INT [PK], UserId: INT [FK to Users.UserID], RestaurantID: INT[FK to Restaurants.RestaurantID])
- Users(UserID[PK], Email: CHAR[225], ZipCode: VARCHAR(10))
- OrderItems(ItemID: INT[PK], OrderID: INT[FK to Orders.OrderID], UserID: INT [FK to Users.UserID], RestaurantID: INT[FK to Restaurants.RestaurantID], Name: CHAR[225] [FK to MenuItem.Name], Quantity: INT)

Relationships

- Offers(RestaurantID:INT [PK] [FK to MenuItems.ResturantID], RestaurantID:INT [FK] [FK to Restaurant.ResturantID])
- Selected from(Name: CHAR[225] [PK] [FK to OrderItems.Name], Name: CHAR[225] [FK to MenuItems.Name])

- Receives(RestaurantID:INT [PK] [FK to Ratings.ResturantID], RestaurantID:INT [FK] [FK to Restaurant.ResturantID])
- Contains(OrderId: INT [PK][FK to OrderItems.OrderID], OrderID: INT[FK to Orders.OrderID], UserID: INT [FK to OrderItems.UserId], UserID: INT[FK to OrderItems.UserID] RestaurantID:INT [PK] [FK to OrderItems.ResturantID], RestaurantID:INT [FK] [FK to Orders.ResturantID])
- Makes(UserID: INT[PK][FK to Orders.UserID], UserID: INT[FK to Users.UserID])