Food Delivered Database

Jiawei Zhao 001495711, Ke Yuan 001491111
Mengying Wang 001357559, Keyi Cao 001302011

Database Purpose:

The purpose of the database is to maintain the data using to complete a food delivered order, which is widely used in food delivery application.

Business Problems Addressed:

- Provide restaurant owner with advice on improving service.
- Generate a rank on different kinds of restaurants.
- Customers can view others' comments to know about the restaurant and the food better.
- Allow customers search the information of restaurants and their foods, coupons, activities, opening hours, rating and any other useful contents.
- Allow customers placing, cancelling and rating orders, they also can check the status of their orders.
- Permit restaurants release activities, make coupons, edit food information and categories and receive orders.

Business rules:

- There are 3 kinds of roles who can log in: customers, restaurants' managers and delivery man. All of them use their mobiles or emails for registering.
- Each customer can have more than one address. Each address must have detailed information with location, receiver's name and telephone number.
- Customers can use only one coupon per order.
- Customers must have a legal license when they register as a delivery man or restaurant manager.
- Restaurant manager can edit their food category casually.

Design Requirements (Credit to Professor Simon Wang):

- User Crow’s Foot Notation.
- Specify the primary key fields in each table by specifying beside the fields.
- Draw a line between the fields of each table to show the relationship between each table.
- Specify which table is on the one side of the relationship by placing a one next to the field where the line starts.
- Specify which table is on the many side of the relationship by placing a crow’s feet symbol next to the field where the line ends.

Design Decisions:

Entity name	Why entity included	How it related to other entities
Customer	These entities save the registered Information (mobile or email, user id, username and password) of users. Our users can be divided into three roles: customers (“Customer” table in database), restaurants and delivery man. These entities will be used when we need to verify users’ identity such as “login”, “make a payment” or “reset the password”.	Customer is the one of the most important entities in our database, it represents the customers who make order. It is related to “Customer_address” “Order” and “Order_comment”. Each customer can hold many addresses (one-to-many), then they can choice one of them to submit an order. Each customer can submit many orders (one-to-many). After the order completed, customer can rate the order by writing a comment for the order (one-to-many).
Restaurant		Restaurants’ detailed information, such as address and open time, is saved in this entity. Restaurant managers must upload one legal license ("Res_license”, one-to-one) when they register as a user, they can release activities (“Res_activity”, one-to-many) and make coupons (Coupon, one-to-many), they can edit their food category casually

		("Food_category", one-to-many) and they can receive many orders ("Order", one-to-many) from customers.
Deliver		Deliver is related to "Deliver_license" due to delivery man must upload one legal driver license when they register as a user, they have one-to-one relationship. Deliver also has one-to-many relationship with "Deliver_task", which means the tasks received by delivery man.
Res_activity	For promotion, restaurants can release some activities, such as "Buy one, Get one 50% OFF during Lunar New Year". It must include the activity id, restaurant id, target customers, discount details and period of validity.	This entity has one-to-many relationship with "Restaurant" due to activities are released by restaurants.
Coupon	Coupon can be published by both the restaurant and the application agent. For example, when customer consume more than 100 dollars. He can get a $10 discount. These can be saved in this entity.	It needs a foreign key representing the publisher. If a coupon is published by application agent. It can be used in all restaurants. The corresponding column value is "all"
Pay	We need to track every payment in the transactions. That's an important part for both sides. The status, payment method and dealing time should be logged.	This entity is related to order. We need a foreign key for order to represent which order we are dealing with.
Food category	Every restaurant has their food classified into kinds of category. We give the owners permissions to create personal categories for their menu. Then we keep all kinds of categories in this entity. We can select some most often	This entity is designed to save the created categories. We need to relate this entity to restaurant. Add restaurant id as foreign key.

	categories as suggestion for the owner.	
Food	We keep all kinds of food in this entity.	Every dish has a category. We design a foreign key to track the belonging relationship. Every category can be identified by category entity. So, we know which restaurant it belongs to.
Customer_address	This entity is used to save all the delivery address. A customer can have several addresses stored in the system. In real world situation, different people may have the same address. Although this could happen, we still decide to implement only one-to-many relationship because we can keep customer's address separately in system. In a real table, it could happen that two addresses have the same content but different "address_id".	Related to " Customer": a customer can have seral different address. An address should depend on a customer. That's why we decide to add a "Customer _id" here as a foreign key. Although in real life different people may share the same address and that makes customer and address have many-to –many relationships, we decide to make it simple. A one-to-many relationship also works in this condition. We treat the same address separately. In many real-world scenarios, the same address can be described with different ways. Related to "Order": where this order should be sent to? The address of receiver should be shown for the deliverer. That's why we need to create a relationship between "Customer _address" and "Order".
Res_license	These entities save the license information. A restaurant needs a license to run. A qualified deliverer needs a license to do his or her job. I must point out here that although they both have	Related to "Restaurant": They have one-to-one relationship. One restaurant can only have one license and on the contrary, it works too.
Deliver_license		Related to "Deliver": One-to-one relationship. I prefer to call this

	“lisence_id” attribute, they are totally different.	entity “deliverer”. One deliver man only needs one license to work. One license can authorize one deliverer to work.
Order_food	This is the associative entity between food and orders. They are a many-to-many relationship. So, we need an extra entity. Besides in this entity, we have the attributes called “amount”. It can reflect the number the customers purchased for a specific kind of food.	As an extra entity, it is linked to the “order” and “food”.
order	This entity contains all kinds of information of an order. As for an order, we not only need to know the information of restaurants and the coupon to the customers， but also should contains the time when the payment is done, the location where food should be delivered to.	Related to “Order_comment”: As one customer only need to make comment once, so it is a one-to-one relationship. Related to “Pay”: One order only needs to pay for once, so this is a one-to-one relationship Related to “Restaurant": A restaurant can have many orders, but one order can only have one restaurant, so this is a one-to-many relationship. Related to “Coupon”: Every order can use one kind of coupon or not use coupon. Related to “Order_food”: This is linked to extra entity, as “Order” and “Food” are an many-to-many relationship. Related to “Customer”: One order can only be owned by one customer. Related to “Customer address”: One order only needs to be delivered to one location. Related to “Deliver_task”:

		Customer can use the delivery service or pick up by themselves. As the core of the food delivery, this entity is almost related to all the information.
Order_comment	This entity is responsible for customer to comment his order. When the foods have been delivered, the customers can make comments on the taste of the food and the quality of the delivery, which will give other customers some initial information of restaurant and help them make the choice of the food.	Related to "Customer": Customer is the may character who write down their comment for every order. They can evaluate the performance of deliverer, food quality and pricing. One customer may leave a lot of comments for different orders. That makes “Customer” and “Order_comment” a one-to-many relationship. Related to "Order": customers can choose to leave some comments for an order or not. Sometimes people leave no comments. This is optional.
Deliver_task	Use to save a task allocated to a deliver person. It will tell the deliver the details of the task and remind the deliver the start time in case the deliver becomes too long, and the “status” will show whether the deliver has been doon.	Related to "Order": An order can be one and only one deliver task. But sometimes people want to pick up food themselves so that in this situation, an order will not be defined as a deliver task. Or sometimes an order is canceled before being assgined to a deliverer because something went wrong. But at most one order can only be one deliver task. Related to "Deliver": A deliver task can only be execute by one deliverer. However, one deliverer can execute many deliver tasks. This is a one-to-many relationship.