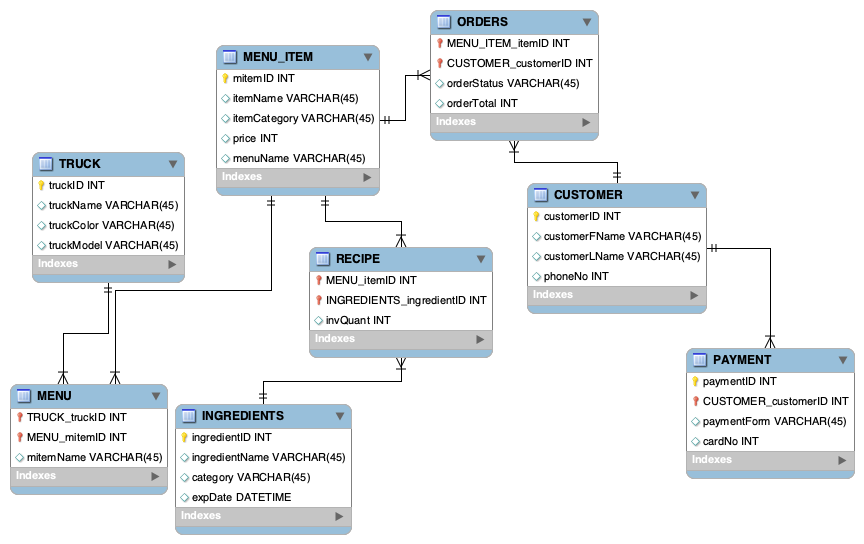
BUS315: Database Management W’22, Team II

Danny Kung, Paradis Ikirezi, Katie Lydon, and Isaiah Medina

**Database Model**



**Figure 1. Database model describing business process for Food for Thought, LLC’s (FFT) three business. Model was created in MySQLWorkbench and later forward engineered.**

Our data model maps the relationship between customers, menus, ingredients and trucks for the Food for Thought, LLC. Our goal when creating this model was to generate a model that showed the relationships between the trucks and the menus they offered, what items were on those menus, what ingredients were needed for such menu items, who was buying what menu item, and how they were paying for it. We anticipated our data model to be complex; therefore, our first step was to identify and map independent tables

We started by creating an independent Customer table that contained customer information such as name and ID. We then modeled the relationship a customer has to their payment method (how one customer can have many payment methods in our business model). Next, we created the independent Menu Item table which included attributes such as menu item ID, price, item category and item name. Our third independent table titled Ingredients which included attributes such as ingredient ID, name, and quantity. The last independent table was Truck, which include truck name and vehicle model.

Our next step was to build associative tables. We first explored how the Menu Item entity related to several other different entities. We knew that many different customers could order many different menu items, so we created an associative table, named Orders, that linked Menu Items and the Customer. Next, we created another associative table called recipes to link Ingredients and Menu Items. This table shows us how much of an ingredient is required to make a menu item. This table would be important in case the company wishes to expand this model to analyze vendors and wholesale ordering in the future. We created our third associative table linking Menu Item and Truck, Menu, to help model which menu items were sold on each truck as a truck can serve many menu items and the same menu item can be sold on many trucks.

A design choice we made for all associative tables was to utilize composite foreign key rather than producing a primary key for each associative table. For example, in our Orders associative table, we used a composite foreign key from both parent tables (customer ID and menu item ID) serve as primary keys. Therefore, a business rule is that when tracking an order number, you have to look at the unique combination of a customer ID and a menu item ID. In our data set / business, a customer never orders the same thing twice, so this does create a primary key / unique identifier.

**Data Dictionary:**

**TRUCK**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **columnName** | **dataType** | **Data Category** | **Description** | **Required** | **Sample** |
| truckID | INT | Surrogate primary key of the table | Unique number assigned to each truck | Yes | 1 |
| truckName | VARCHAR(45) | Attribute | Name of the food truck | Yes | Lex Mex |
| truckColor | VARCHAR(45) | Attribute | Color of the food truck | No | Green |
| truckModel | VARCHAR(45) | Attribute | Model of the food truck | No | Ford Stepvan |

**MENU\_ITEM**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **columnName** | **dataType** | **Data Category** | **Description** | **Required** | **Sample** |
| mitemId | INT | Surrogate Primary key of table | Unique number assigned to each item | Yes | 7 |
| itemName | VARCHAR(45) | Attribute | Name of menu item | Yes | Chicken burrito |
| itemCategory | VARCHAR(45) | Attribute | Category of the menu the item belongs in | No | Main |
| price | INT | Attribute | Price of menu item | Yes | 8 |
| menuName | VARCHAR(45) | Attribute | Name of truck that serves the item | Yes | Lex Mex |

**MENU**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **columnName** | **dataType** | **Data Category** | **Description** | **Required** | **Sample** |
| TRUCK\_truckID | INT | \*Primary and foreign key of the table | Unique number assigned to each truck | Yes | 1 |
| MENU\_mitemID | INT | \*Primary and foreign key of the table | Unique number assigned to each menu item | Yes | 7 |
| mitemName | VARCHAR(45) | Foreign key of the table | Name of menu item | Yes | Chicken Burrito |

\*Data Category: means that multiple foreign keys are combined into a composite key

**RECIPE**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **columnName** | **dataType** | **Data Category** | **Description** | **Required** | **Sample** |
| MENU\_itemID | INT | \*Primary and foreign key of the table | Unique number assigned to each menu | Yes | 7 |
| INGREDIENTS\_ingredientID | INT | \*Primary and foreign key of the table | Unique number assigned to each ingredient | Yes | 10 |
| invQuant | INT | Attribute | Inventory quantity needed for recipe. | No | 80 |

\*Data Category: means that multiple foreign keys are combined into a composite key

**SQL Queries:**

**1.** a. **The aim of the query:** to identify which customers are paying in cash instead of using a card. A business may analyze the customers who are using cash when deciding about whether to continue to accept cash or only cards. It will also help us keep track of which customers used cash instead of card when we are calculating profits.

b. **How we performed the query**: This query selects the distinct customerIDs and card numbers from orders. Orders is joined with customers and with payments to help find the card numbers that are NULL.

c. **Query:**

Which customers are paying in cash?

**SELECT DISTINCT customerID, cardNo**

**FROM orders JOIN customer ON orders.CUSTOMER\_customerID=customer.customerID JOIN payment ON customer.customerID=payment.CUSTOMER\_customerID**

**WHERE cardNo IS NULL;**

d. **Result:**

**+ --------------- + ----------- +**

**| customerID | cardNo |**

**+ --------------- + ----------- +**

**| 1 | |**

**| 4 | |**

**| 10 | |**

**+ --------------- + ----------- +**

**2.** a. **The aim of the query:** to determine which items on the Lex Mex menu are vegetarian. Information from this query can help managers decide on what menu items to offer so as to meet the desires of their diverse customers. It also helps the vegetarian customers easily navigate their options.

b. **How we performed the query**: This query selects items and truck name from the menu item entity but restricts the items only to those from the Lex Mex truck and excludes steak tacos, steak burrito, chicken tacos, and chicken burrito using a NOT IN function. It then orders our results by the name of the item.

c. **Query**:

What are the vegetarian items at the Lex Mex food truck?

**SELECT itemName, menuName**

**FROM menu\_item**

**WHERE menuName = "Lex Mex" AND itemName NOT IN ("steak tacos", "steak burrito", "chicken burrito", "chicken tacos")**

**ORDER BY itemName;**

d. **Result:**

**+ ------------- + ------------- +**

**| itemName | menuName |**

**+ ------------- + ------------- +**

**| chips | Lex Mex |**

**| chocolate ice cream | Lex Mex |**

**| churros | Lex Mex |**

**| coca-cola | Lex Mex |**

**| guacamole | Lex Mex |**

**| refried beans | Lex Mex |**

**| rice | Lex Mex |**

**| salsa | Lex Mex |**

**| vanilla ice cream | Lex Mex |**

**| veggie tacos | Lex Mex |**

**| water | Lex Mex |**

**+ ------------- + ------------- +**

**3**. a. **The aim of the query:** to list the highest price of any menu item served by Oink-Moo BBQ. This query enables customers who might not remember the full truck name to use only the word “oink” to obtain all items sold by that truck and then find out the highest price of any item by that truck.

b. **How we performed the query**: This query uses the Max aggregate function on the menu item entity and restricts the menu name to those with “oink” in the name by using REGEXP.

c. **Query:**

What is the highest price of items sold by Oink-Moo BBQ?

**SELECT MAX(price)**

**FROM menu\_item**

**WHERE menuName REGEXP ‘oink’;**

d. **Result:**

**+ --------------- +**

**| MAX(price) |**

**+ --------------- +**

**| 10 |**

**+ --------------- +**

**4**. a. **The aim of the query**: This query seeks to determine which trucks have more than the average number of customers for all trucks. This helps all the trucks see which truck is the most popular with customers so that they can learn potential ways to attract more customers. This also helps managers allocate resources accordingly.

b. **How we performed the query**: selects a distinct truck name and counts the number of customers from the joined tables to truck, menu, menu item, and orders. The query then groups them by the customer ID and uses the Having function to restrict the trucks to those having more than 2 customers.

c. **Query**:

Which truck(s) have higher than the average number of customers?

**SELECT DISTINCT truckName, COUNT(CUSTOMER\_customerID)**

**FROM truck JOIN menu ON truckID=TRUCK\_truckID JOIN menu\_item ON MENU\_mitemID=mitemID JOIN orders ON mitemID=MENU\_ITEM\_itemID**

**GROUP BY CUSTOMER\_customerID**

**HAVING COUNT(CUSTOMER\_customerID)>2;**

**d. Result:**

**+ -------------- + ------------------------------- +**

**| truckName | COUNT(CUSTOMER\_customerID) |**

**+ -------------- + ------------------------------- +**

**| Lex Mex | 3 |**

**+ -------------- + ------------------------------- +**

**5**. a. The aim of the query: This query is looking to find which ingredients are in the menu item that costs the most. This will help us keep track of which ingredients are being used for the most expensive items on the menus so that we can keep track of our inventory and when we should restock.

b. How we performed this query: This query selects the item name, ingredient name, and price from the joined tables of ingredients, recipe, and menu item. It then employs the use of an inner query to restrict the price to be equal to that of the maximum in the entire menu item entity.

c. **Query:**

What are the most expensive menu items and what ingredients do they require?

**SELECT DISTINCT itemName, ingredientName, price**

**FROM ingredients JOIN recipe ON ingredientID=INGREDIENTS\_ingredientID JOIN menu\_item ON MENU\_itemID=mitemID**

**WHERE price = (SELECT MAX(price) FROM menu\_item);**

d**. Result:**

**+ ------------- + ------------------- + ---------- +**

**| itemName | ingredientName | price |**

**+ ------------- + ------------------- + ---------- +**

**| steak burrito | Tortilla | 10 |**

**| steak burrito | Steak | 10 |**

**| steak burrito | Rice | 10 |**

**| large mac | Macaroni Pasta | 10 |**

**| large mac | Cheddar Cheese | 10 |**

**| brisket | Brisket | 10 |**

**+ ------------- + ------------------- + ---------- +**

**6**. a. **The aim of the query:** This query could help us show specific inventory quantities that are higher than the average so that each restaurant knows whether or not they may need to restock any time soon.

b. **How we perform the query**: First, it selects the distinct and non-repeated ingredients and their inventory quantity from the joined menu item, recipe, and ingredients tables. It then checks to see which ingredients in the current inventory are greater than the average inventory quantity for an ingredient through an aggregate function of AVG and inner query. We then ordered by inventory quantity to see them all.

c. **Query:**

Which ingredients do we have in inventory where their inventory quantity is greater than

the average inventory quantity amount for an ingredient?

**SELECT DISTINCT ingredientName, invQuant**

**FROM menu\_item JOIN recipe ON menu\_item.mitemID=recipe.MENU\_itemID JOIN ingredients ON ingredients.ingredientID=recipe.INGREDIENTS\_ingredientID**

**WHERE invQuant >(SELECT AVG(invQuant) FROM recipe JOIN ingredients ON ingredients.ingredientID=recipe.INGREDIENTS\_ingredientID )**

**ORDER BY invQuant;**

d**. Result:**

**+ ------------------- + ------------- +**

**| Coca-Cola | 75 |**

**| Cabbage | 80 |**

**| Mayonaise | 80 |**

**| Tortilla | 80 |**

**| Water | 100 |**

**| Macaroni Pasta | 100 |**

**| Cheddar Cheese | 100 |**

**+ ------------------- + ------------- +**

**7**. a. **The aim of the query**: This query can be used for employees working in the kitchen to determine the order status of specific menu items as well as its price. This should help them stay organized so that they know what orders to work on and which ones have been completed.

b. **How we performed the query**: the query selects the item name, order status, and price from the menu\_item table and orders table. We use the DISTINCT function so that there are no repeats and are specifically looking for where the order status is delivered. We then ordered the items by their prices, smallest to largest.

c. **Query:**

What items are delivered and what are they priced at?

**SELECT DISTINCT itemName, orderStatus, price**

**FROM menu\_item JOIN orders ON mitemID=MENU\_ITEM\_itemID WHERE orderStatus="Delivered"**

**ORDER BY price;**

d. **Result**:

**+ ------------- + ---------------- + ---------- +**

**| itemName | orderStatus | price |**

**+ ------------- + ---------------- + ---------- +**

**| salsa | Delivered | 2 |**

**| vanilla ice cream | Delivered | 3 |**

**| chips | Delivered | 3 |**

**| steak burrito | Delivered | 10 |**

**+ ------------- + ---------------- + ---------- +**

**8.** a. **The aim of the query**: Our query shows us on which days did we have customers that did not pay in cash. This helps the food trucks keep a record of when customers paid with card in case there is a dispute about when a purchase was made on their card. Selects the date where there were customers who had a cardNo value of NULL.

b. **How we performed the query**: We used the DATE function as it extracts date, month and year from the payment entity. We proceeded to restrict our results to those whose cardNo was Null.

b. **Query:**

On which date did customers pay with credit card?

**SELECT DATE(pay\_date)**

**FROM payment**

**WHERE cardNo IS NOT NULL;**

d. **Result:**

**+ ---------------- + ----------------------- +**

**customerFName| customerLName| pay\_date**

**+ ---------------- + ----------------------- +**

**|Anne** **| Navarro|**  **2022-03-21**

**|Sammy|** **Haney|** **2022-01-10**

**|Bobby|** **Marsh|** **2022-02-14**

**|Dashawn|** **Carey|**  **2022-03-18**

**|Mira|** **Roach|**  **2022-02-24**

**|Daphne|** **Ryan|**   **2022-01-15**

**|Judith|** **Castro|** **2022-03-21**

**+ ---------------- + ----------------------- +**

**9.** a. **The aim of the query:** Our query shows us the most popular forms of payment and how many of each type was used when ordering. This helps the trucks determine whether they may need to carry more cash on hand or if they should focus on digital payments.

b. **How we perform the query**: The query selects the payment form type and counts the number of payment forms under each category using the COUNT function. It then groups the data by the 3 different payment forms.

c. **Query:**

How many payments were made by cash, credit, and debit?

**SELECT paymentform, COUNT(paymentform)**

**FROM payment**

**GROUP BY paymentform;**

d. **Result:**

**+ ---------------- + ----------------------- +**

**| paymentform | COUNT(paymentform) |**

**+ ---------------- + ----------------------- +**

**| cash | 3 |**

**| credit | 3 |**

**| debit | 4 |**

**+ ---------------- + ----------------------- +**

**10.** a. **The aim of the query:** This query tries to show us and our customers what items are the cheap on each menu, their price, what category they are in, and which truck sells them. Specifically, it’s looking to show what items are sold at a price less than the average. This can help customers see what menu items are on the cheaper side if they are trying to save money.

b. **How we performed the query:** The query selects the item name, price of the item, what category it's in, and what truck the item comes from. We then find which items are priced below the average price of all the menu items combined using the AVG function and an inner query.

c. **Query:**

What items are less than the average price of all the items and what are they priced at?

**SELECT itemName, price, itemCategory, menuName**

**FROM menu\_item**

**WHERE price < (SELECT AVG(PRICE) FROM menu\_item);**

d. **Result:**

**+ ------------- + ---------- + ----------------- + ------------- +**

**| itemName | price | itemCategory | menuName |**

**+ ------------- + ---------- + ----------------- + ------------- +**

**| chips | 3 | side | Lex Mex |**

**| salsa | 2 | side | Lex Mex |**

**| guacamole | 3 | side | Lex Mex |**

**| refried beans | 4 | side | Lex Mex |**

**| rice | 3 | side | Lex Mex |**

**| water | 1 | drink | Lex Mex |**

**| coca-cola | 2 | drink | Lex Mex |**

**| vanilla ice cream | 3 | dessert | Lex Mex |**

**| chocolate ice cream | 3 | dessert | Lex Mex |**

**| garlic bread | 4 | side | Mac Mart |**

**| coca-cola | 2 | drink | Mac Mart |**

**| water | 1 | drink | Mac Mart |**

**| vanilla ice cream | 3 | dessert | Mac Mart |**

**| chocolate ice cream | 3 | dessert | Mac Mart |**

**| coleslaw | 4 | side | Oink-Moo BBQ |**

**| baked beans | 4 | side | Oink-Moo BBQ |**

**| water | 1 | drink | Oink-Moo BBQ |**

**| coca-cola | 2 | drink | Oink-Moo BBQ |**

**| vanilla ice cream | 3 | dessert | Oink-Moo BBQ |**

**| chocolate ice cream | 3 | dessert | Oink-Moo BBQ |**

**+ ------------- + ---------- + ----------------- + ------------- +**

**SQL Matrix:**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Query1 | Q2 | Q3 | Q4 | Q5 | Q6 | Q7 | Q8 | Q9 | Q10 |
| 1. Subquery |  |  |  |  | X | X |  |  |  | X |
| 2. GROUP BY |  |  |  | X |  |  |  |  | X |  |
| 3. HAVING |  |  |  | X |  |  |  |  |  |  |
| 4. ORDER BY |  | X |  |  |  | X | X |  |  |  |
| 5. IN/NOT IN |  | X |  |  |  |  |  |  |  |  |
| 6. Aggregate function(s) |  |  | X | X | X | X |  |  | X | X |
| 7. REGEXP |  |  | X |  |  |  |  |  |  |  |
| 8. Date function(s) |  |  |  |  |  |  |  | X |  |  |
| 9. IS NULL | X |  |  |  |  |  |  |  |  |  |