VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Database Management System (CC08)

Assignment

Online Restaurant Menu Management

Advisor: Dr. Tran Quang Trung
Students: Dang Cao Cuong - 1952598
Nguyen Hoang Cat - 1952044
Nguyen Thanh Chuong - 1952595

HO CHI MINH CITY, SEPTEMBER 2021

Contents

1 Requirement Description

1.1 Functional requirement

1.1.1 General

- A server shall host the system and provide system data processing and storage capability.
- The system shall be able to generate a daily and monthly report about expenditure cost and profits.
- The system shall be able to inform the customers about promotions, discounts, events.
- All system functionalities shall be accessible through computers, tablets and displays.

1.1.2 Customer

- A customer shall be able to access to the restaurant's menu by scanning the QR code, which has been placed on the table.
- A customer shall be able to navigate through the available items in their engaged menu.
- A customer shall be able to call for waiter/waitress by 1 button within the menu interface.
- A customer shall be able to see the order and bill information.
- A customer shall be able to choose the transaction method to pay the bill.
- A customer shall be able to give feedback to the restaurant.

1.1.3 Vendor

- A vendor shall be able to sell/supply item/ingredient for the restaurant.
- A vendor shall be able to review the order/bill with the buyer.
- A vendor shall be able to access the location of the restaurant.
- A vendor shall be able to choose the shipment method.

1.1.4 Employee

- An employee shall be able to log into a tablet using their assigned username and password.
- An employee shall be able to log out of a tablet.
- A waiter assigned to a table shall be alerted via their wireless tablet when a booking is placed from a table and a table has requested waiter assistance.
- A tablet shall allow a waiter to indicate the delivery of an item to its customer.
- A tablet shall allow a waiter to process a payment using cash.
- A tablet shall allow a waiter to process a payment using a bankcard.

1.2 Non-functional requirement

1.2.1 Safety

- The system shall log every state and state change of every computer, tablet and display to provision recovery from system failure.
- The system shall be capable of restoring itself to its previous state in the event of failure (e.g. a system crash or power loss).
- The system shall be able to display a menu at all times.
- The system shall utilize periodic 30-second keep-alive messages between tablets and the server to monitor tablet operational status.
- The system shall flag tablets that fail to send timely keep-alive messages as non-operational and disassociate the assigned waiter from the tablet.

1.2.2 Security

- Wireless communication throughout the system will be encrypted using SSLv3 at the application layer and WPA2-PSK at the data link layer.
- The WPA2-PSK password used for wireless communication must have a bit-strength of at least 80 bits.
- The WPA2-PSK password used for wireless communication must be changed every three months.
- A waiter password used for tablet login must have a bit-strength of at least 64 bits.
- A waiter password used for tablet login must be changed every three months.
- A waiter shall only be able to log into one tablet at any given instance of time.
- A waiter that attempts to log into a second tablet while already logged into another tablet shall be rejected and notified through both tablets.
- The system shall provide two levels of access:
  - A supervisor level for unrestricted access to system functionality.
  - A waiter level for access to waiter functionality.
- A computer shall not require a user to log in.

1.2.3 Performance

- The system shall not take over 3 seconds opening the menu, order foods, paying the bill.
- The server shall be capable of supporting no less than 200 concurrent connections from any combination of surface computers, tablets and displays.
- The server shall provide no limit on how many devices are in the system.

- The server shall be capable of supporting an unlimited number of active meals/orders, that is, no meals/orders shall be lost under any circumstances.
- The server shall be capable of supporting an unlimited number of active customer payments, that is, no payments shall be lost under any circumstances.

2 DB design and DB design tool preference

2.1 Conceptual diagram

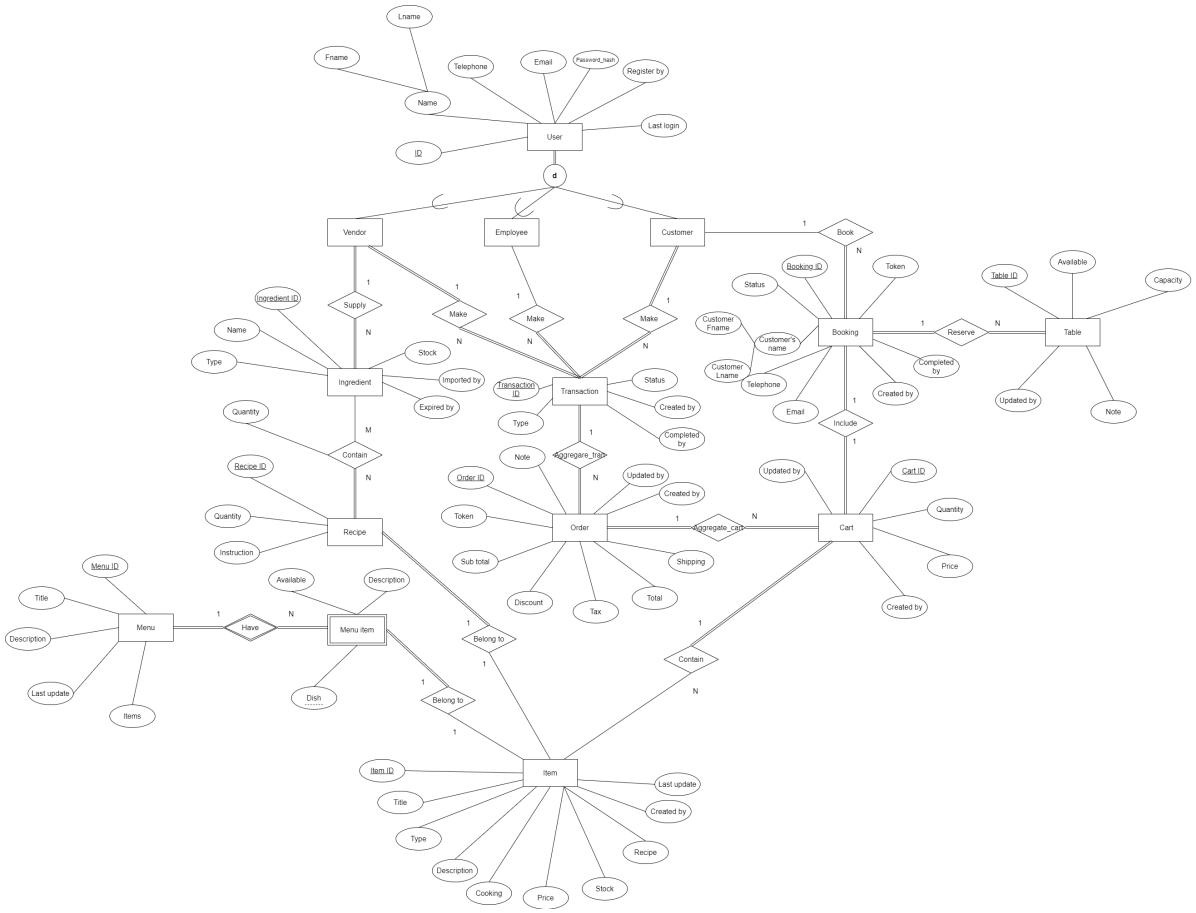


Figure 1: Conceptual diagram for Online Restaurant Menu Management

Below is the detailed description of each entity as well as each attribute belongs to the corresponding entity:

User: This entity models users of online restaurant menu management. This entity is a superclass, which means they can be either vendor, employee or customer. It contains attributes:

- ID: The unique id to identify the user.
- Name: The name of the user, including first name (Fname) and last name (Lname) of the user.
- Telephone: The telephone of the user. It can be used for login and registration purposes.
- Email: The email of the user. It can be used for login and registration purposes.
- Register by: The registration date of the user.
- Last login: It can be used to identify the last login time of the user.

Ingredient: This entity models the ingredient that a vendor supplies for the restaurant. It contains attributes:

- Ingredient ID: The unique id to identify the ingredient.
- Name: The name of the ingredient.
- Type: The type to distinguish between the different ingredient types.
- Stock: To track the ingredient inventory.
- Imported by: The date of the importation of the ingredient.
- Expired by: Expiration date of the ingredient.

Recipe: This entity is used to track the quantity of the ingredients required for an item for a single serving. It contains attributes:

- Recipe ID: The unique id to identify the recipe.
- Quantity: The quantity of the ingredient required to cook the item for a single serving.
- Instruction: The ingredient instructions required to cook the item.

Transaction: We also need a transaction entity to track the order payments made by the guests to the restaurant and restaurant to the vendors. It contains attributes:

- Transaction ID: The unique id to identify the transaction.
- Type: The type of order transaction can be either Credit or Debit.
- Status: The status of the order transaction can be New, Cancelled, Failed, Pending, Declined, Rejected, and Success.
- Created by: It stores the date and time at which the order transaction is created.
- Completed by: It stores the date and time at which the order transaction is completed.

Order: The order entity can be used to store the completed bookings and vendor orders. It contains attributes:

- Order ID: The unique id to identify the order.
- Token: The unique token associated with the order to relate it with the booking. The same token can also be passed to the Payment Gateway if required.
- Sub total: The total price of the order items.
- Discount: The total discount of the Order based on the promo code or store discount.
- Tax: The tax on the order items.
- Total: The total price of the order to be paid by the guest to the restaurant or the restaurant to the vendor.
- Shipping: The shipping charges of the order items.
- Created by: It stores the date and time at which the order is created.
- Uploaded by: It stores the date and time at which the order is updated.
- Note: Used to store the additional details of the order.

Booking: Can be used to book the restaurant tables either online or on-premises. A logged-in or existing user can also be associated with Booking. It contains attributes:

- Booking ID: The unique id to identify the booking.
- Token: The unique token associated with the booking.
- Status: The status of the booking can be New, Lounge, Active, and Complete.
- Customer's name: Including customer first name and last name.
- Telephone: The telephone number of the user.
- Email: The email of the user.
- Created by: It stores the date and time at which the booking is created.
- Completed by: It stores the date and time at which the booking is completed.

Table: This entity models the table associated with the booking. It contains attributes:

- Table ID: The unique id to identify the table.
- Available: A boolean attribute to indicate the table is available or not.
- Capacity: The capacity of the table.
- Updated by: It stores the date and time at which the table is updated.
- Note: Is used to store the additional details of the table.

Cart: The shopping cart to store the booking items. It contains attributes:

- Cart ID: The unique id to identify the cart.
- Quantity: The quantity of items the cart has.
- Price: The total price of the cart.
- Created by: It stores the date and time at which the cart is created.
- Updated by: It stores the date and time at which the cart is updated.

Menu: The online menu of the restaurant. It contains attributes:

- Menu ID: The unique id to identify the menu.
- Title: The menu title to be displayed.
- Description: The description of the menu.
- Last update: It stores the date and time at which the menu is last updated.
- Items: The items in the menu.

Menu item: Is the weak entity to be used to track the items available in the menu. It contains attributes:

- Dish: The name of the dish.
- Available: The flag to check whether the menu item is available.
- Description: The description of the menu item.

Item: This entity models the general items in the restaurant. It contains attributes:

- Item ID: The unique id to identify the item.
- Title: The item title to be displayed on the Menu.
- Type: The type to distinguish between the different item types.
- Description: The description of the item.
- Cooking: The flag to identify whether cooking is required for the item.
- Price: The selling price of either one unit or a single serving.
- Stock: To track the item inventory.
- Recipe: The instructions required to cook the item.
- Created by: It stores the date and time at which the item is created.
- Last update: It stores the date and time at which the item is last updated.

2.2 Database design tool preference

In the assignment, we will work with Diagrams.net (formerly known as draw.io) to create conceptual diagram by using Chen's notation.

Brief description about Diagrams.net: diagrams.net/draw.io is an open-source technology stack for building diagramming applications, and the world's most widely used browser-based end-user diagramming software. It supports many graphs, notations, symbols from other services, group editing, sharing with everyone, has many powerful features and have a priority for privacy.

Advantages of Diagrams.net:

- Totally free.
- Unlimited diagrams and work space.
- Support application and web-based software.
- Automatically save the backup of the data.
- Works on desktop or mobile instances.
- Versatile and easy-to-use, with both ready-made and customizable elements.
- Privacy first and secure.
- Allow sharing and importing/exporting.

Disadvantages of Diagrams.net:

- Sometimes users experience lags when working on browser for a long time.
- Not enough tools to make super complex things.
- Editing diagrams on mobile version might be difficult.
- Is not efficient for adding 3D effect on the shapes.

3 Investigate/Study a logical DB design tool

3.1 Database design tool - MYSQL Workbench

There are many DB design tools but group 1 chose MySQL Workbench due to its simplicity and efficiency in both designing and viewing. MySQL Workbench is a visual and graphic tool that can be used for the development of the database, modeling, configuring, designing, creating, and maintaining the database including backup and restoration using interactive graphics. It is a sophisticated tool that can be used by the developers, architects, and administrators of the database. It was developed by Sun Systems/Oracle and is completely cross-platform so it can be used in operating systems like MacOs, Linux/Unix, and Windows.

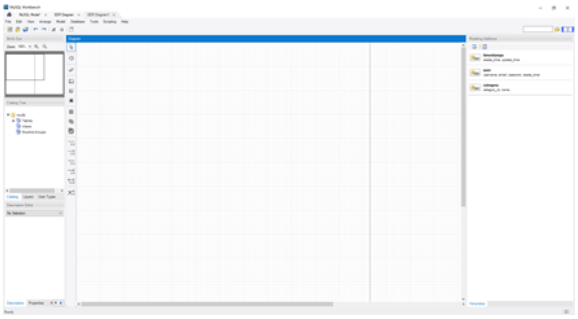


Figure 2: MySQL Workbench user interface

3.1.1 Create tables with MySQL Workbench

To start designing a data model, user can click the set of buttons on the left bar to create an empty table. After double clicking the empty table, a schema editor will pop up to allow users to edit table name and attributes.

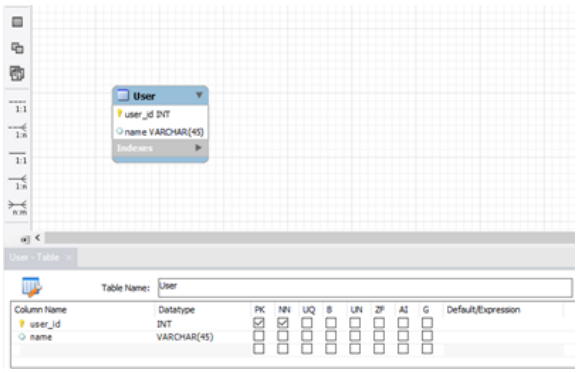


Figure 3: MySQL Workbench table creation

In this editor user can change table name, add or remove attribute names and types, set attribute constraints like primary key, nullable, unique...

3.1.2 Relationships design

Relationship between tables is the core of relational data model, MySQL Workbench allows users to create relationships by choosing the relationship type and the two tables to link them together.

A link between 2 tables confirmed that the relationship setting is successful. MySQL Workbench allows for the following type of relationships:

- Non-identifying one to one

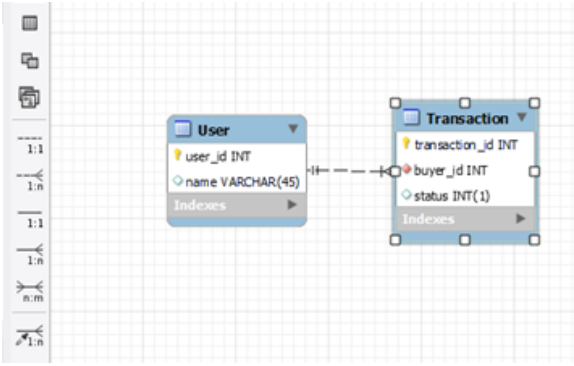


Figure 4: MySQL Workbench relationship menu

- Non-identifying one to many
- Identifying one to one
- Identifying one to many
- Identifying many to many

4 Logical design using relational Database model

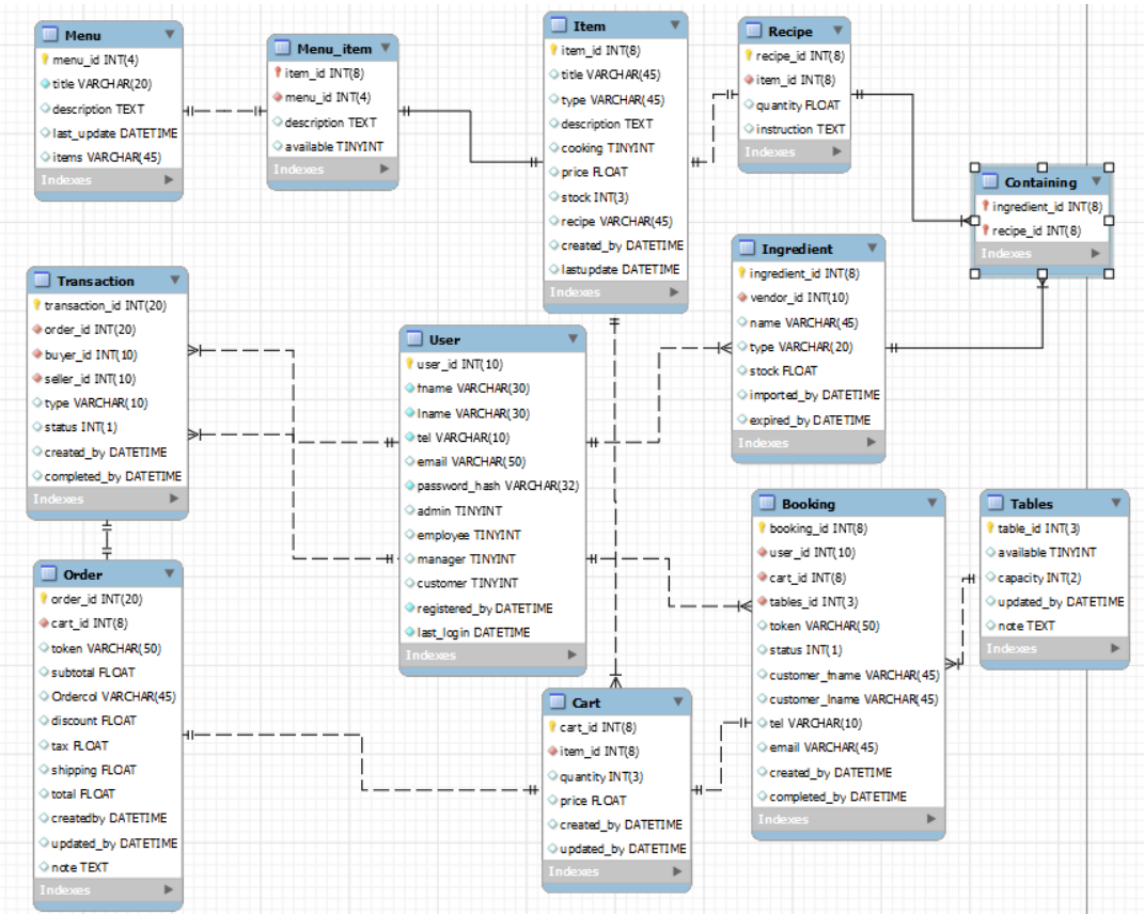


Figure 5: Final database design

4.1 User

In this relational, we use **multiple relation method** coupled with **vendor**, **employee** and **customer** type attribute to deal with **Generalization** or **Specialization**.

4.2 Transaction

This relation uses 2 **1 to N** relationships with the **User** relation for the buyer and seller identification, those attributes are foreign keys. It also has a **1 to 1** relation with **Order** as a foreign key to make sense with the transaction.

4.3 Order

This relation's primary key is **order_id**.This relation uses one **1 to N** relationship with the **Transaction** relation because every order has a transaction, one **1 to N** relationship with the **Cart** to show the order which belong to a specific cart and thus it has the **cart_id** attribute as a foreign key.

4.4 Menu_item

This relation's primary key is the primary key of **Item** with the foreign key from **Menu**, it has **1 to N** relationship **Menu** and **1 to 1** relationship with **Item** because it is a representation of an **Item** inside a **Menu**.

4.5 Item

This relation's primary key is **item_id**. This relation represents the general item in the restaurant.

4.6 Recipe

This relation's primary key is **recipe_id**. It has a **1 to N** relationship with **Item** relation to show it belongs to the **Item** relation and thus it has the **item_id** foreign key. It also participates in a **M to N** relationship with **Ingredient** relation to show that each recipe can have many ingredient.

4.7 Ingredient

This relation's primary key is **ingredient_id**. It has a **1 to N** relationship with **User** relation to show the vendor will supply the ingredient for the restaurant and thus it has the **vendor_id** foreign key. It also participates in a **M to N** relationship with **Ingredient** relation to show that each ingredient can have many recipe.

4.8 Cart

This relation's primary key is **cart_id**.It has a **1 to N** relationship with Item relation to show to the item it holds and thus it has the **item_id** attribute as a foreign key. It also has a **1 to N** relationship with **Order** relation and **1 to 1** relationship with **Booking** relation.

4.9 Containing

This relation is created to map the **M to N** relationship between entities **recipe** and **ingredient** .We use a tuple of primary key **ingredient_id** and **recipe_id**. They are also foreign keys of this relation.

4.10 Booking

This relation has a primary key of itself, 3 foreign keys to **User**, **Cart** and **Tables**. It has **1 to N** relationships with **User** and **Table**, to allow those 2 relations to have multiple version of **Booking**. It also has a **1 to 1** relationship with **Cart** to show the specific items that was booked

4.11 Table

This relation has a **1 to N** relationship with **Booking** to allow for multiple booking schedule of the same table, the foreign key is held by **Booking** so every **Booking** can only have a table

5 Selection of Database Management System

Recently, the preference of a suitable DBMS has been a problem of concern in that there is a wide of database such as: Oracle, MSSQL, MySQL, etc which each developer need to choose at least one.

MySQL is the most popular free-to-use, open-source database due to its stability, fast-processing mechanism. Moreover, an array of users give credence to this database, even Yahoo, Nokia, Google, Youtube, etc using MySQL to save time and cost for websites having huge data.

Apart from the most popularity, MySQL is also a good preference for applications deployed on Linux, Apache, etc which run on different platforms flexible for use.

Below is a summary of salient feature that we can not bypass:

- The Flexibility Of Open Source

  All the fears and worries that arise in an open source solution can be brought to an end with My SQL's round-the-clock support and enterprise indemnification. The secure processing and trusted software of MySQL combine to provide effective transactions for large volume projects. It makes maintenance, debugging and upgrades fast and easy while enhancing the end-user experience.

- Data Security

  MySQL is globally renowned for being the most secure and reliable database management system used in popular web applications like WordPress, Drupal, Joomla, Facebook and Twitter. The data security and support for transactional processing that accompany the recent version of MySQL, can greatly benefit any business especially if it is an eCommerce business that involves frequent money transfers.

- On-Demand Scalability

  MySQL offers unmatched scalability to facilitate the management of deeply embedded apps using a smaller footprint even in massive warehouses that stack terabytes of data. On-demand flexibility is the star feature of MySQL. This open source solution allows complete customization to eCommerce businesses with unique database server requirements.

- High Performance

  MySQL features a distinct storage-engine framework that facilitates system administrators to configure the MySQL database server for a flawless performance. Whether it is an eCommerce website that receives a million queries every single day or a high-speed transactional processing system, MySQL is designed to meet even the most demanding applications while ensuring optimum speed, full-text indexes and unique memory caches for enhanced performance.

- Round-The-Clock Uptime

  MySQL comes with the assurance of 24X7 uptime and offers a wide range of high availability solutions like specialized cluster servers and master/slave replication configurations.

- Comprehensive Transactional Support

  MySQL tops the list of robust transactional database engines available on the market. With features like complete atomic, consistent, isolated, durable transaction support, multi-version transaction support, and unrestricted row-level locking, it is the go-to solution for full data integrity. It guarantees instant deadlock identification through server-enforced referential integrity.

- Complete Workflow Control

  With the average download and installation time being less than 30 minutes, MySQL means usability from day one. Whether your platform is Linux, Microsoft, Macintosh or UNIX, MySQL is a comprehensive solution with self-management features that automate everything from space expansion and configuration to data design and database administration.

- Reduced Total Cost Of Ownership

  By migrating current database apps to MySQL, enterprises are enjoying significant cost savings on new projects. The dependability and ease of management that accompany MySQL save your troubleshooting time which is otherwise wasted in fixing downtime issues and performance problems.

- The Flexibility Of Open Source

  All the fears and worries that arise in an open source solution can be brought to an end with My SQL's round-the-clock support and enterprise indemnification. The secure processing and trusted software of MySQL combine to provide effective transactions for large volume projects. It makes maintenance, debugging and upgrades fast and easy while enhancing the end-user experience.

As we can see advantages just counted above, MySQL is the most suitable database system for **Online Restaurant Menu Mangement** for reasons:

- Save money for managing and maintaining data

  There is a wide range of expenses needing to cover when coordinating a restaurant in general and a menu website in particular. Besides, the cost for managing and maintaining the database is commonly so high that increase a burden for the resraurant owner. It is an open-source and free to use, MySQL, that really help the restaurant owner in saving costs for the restaurant.

- Operating regularly

  There is a plethora of use-casees using our online system anytime ranging from early morning to midnight. MySQL can meet this demand with round-the-clock availability.

- Support utilities which facilitates scalability

  A menu website has a huge numbers of users and data from customers. Therefore, scalability is of importance. Without scalability, it is impossible to expand business scale. Beside, a large volume of data can help us in analysing or classifying customers which can help us understand customers better in order to provide them the best service. Occasionally, it is necessary to modify the model of database. With the help of **MySQL Workbench**, a tool for design MySQL database server, it facilitates developer to design and generate commands of code.

- Enhance the data security

  Data security is one of the most important factors in any online system accommodating user and transaction information which is our critical assets. MySQL can ensure data protection because various well-known companies have given credence into it.

- Enhance user experience

  MySQL has a high performance with versatile functions which may bolster an user-friendly application.

- Adjust appropriately to our demand

  MySQL provides flexible options appropriate to our system.

6 Physical design on MySQL

6.1 User

- fname, lname: These attibute represents the last name and frist name of the user. Therefore this should be text datatype, specifically **VARCHAR(30)**.
- registered_by, last_login: These attribute respectively record the first time the user agent is created and the last time the user have logined. Therefore, they have datatype **DATETIME**
- email: This attribute is created to contain the email address of the user agent. This attribute has data type **VARCHAR(50)** to contain certain long email address.
- tel: The telephone number is composed of 10 digits which datatype **VARCHAR(30)** can satisfy
- password_hash: This attribute contains a password. It has datatype **VARCHAR(32)** indicating that the maximum length of the password is 32 characters

6.2 Transaction

- created_by, completed_be: These attribute represent the time that the transaction is made between user with the datatype **DATETIME**

6.3 Order

- token, note: This attributes is respectively a text describing a promotion program of the restaurant and note of the order. Datatype **VARCHAR(50)** limits the content of the token below 50 characters and **TEXT** does not limit the content.
- createdby,updated_by: These attributes represent the time. Therefore, their datatype are **DATETIME**
- subtotal, discount, tax, shipping, total: These attributes represent the amount of money. Therefore, their datatype are **FLOAT**

6.4 Menu_item

- available: This attribute just have 2 value 0 and 1 representing availability and unavailability. Therefore, datatype **TINYINT** is appropriate.
- description: This attribute has datatype **TEXT** which allows the items in the menu to be described in words.

6.5 Menu

- description: This attribute has datatype **TEXT** which allows the menu to be described in words.
- last_update: This attribute has datatype **DATETIME**
- items: This attribute has datatype **VARCHAR(45)**

6.6 Item

- item_id: This attribute represents the unique ID of an item. Therefore, it has a data type **INT(8)** which is long enough to record each item.
- title: The item title to be displayed on the Menu. Therefore, it has a data type **VARCHAR(45)** to limit the character to 45.
- type: The type to distinguish between the different item types. Therefore, it has a data type **VARCHAR(45)** to limit the character to 45.
- description: The description of the item. Therefore, it has a data type **TEXT** because it can store a large amount of data.
- cooking: The flag to identify whether cooking is required for the item. Therefore, it has a data type **TINYINT**, which is similar to boolean data type.
- price: The selling price of either one unit or a single serving. Therefore, it has a data type **FLOAT** because the price can be in decimal number.
- stock: This attribute shows the quantity of the item in stock. Therefore, it has a data type **INT(3)**.
- recipe: The instructions required to cook the item. Therefore, it has a data type **VARCHAR(45)**.
- created_by,lastupdate: These attribute respectively record the time the item is created and is last updated. Therefore, it has a datatype **DATETIME**.

6.7 Recipe

- recipe_id: This attribute represents the unique ID of a recipe. Therefore, it has a data type **INT(8)** which is long enough to record each recipe.
- item_id: Is a foreign key to show the relationship with the **Item** relation. The data type of this attribute is also the data type of **item_id** in the **Item** relation, which is **INT(8)**.
- quantity: The quantity of the ingredient required to cook the item for a single serving. Therefore, it has a data type **FLOAT** to represent the quantity in decimal number.
- instruction: The ingredient instruction required to cook the item. Therefore, this attribute has data type **TEXT** which allows the instruction to be described in words.

6.8 Cart

- cart_id: This attribute represents the unique ID of a cart. Therefore, it has a data type **INT(8)** which is long enough to record each cart.
- item_id: Is a foreign key to show the relationship with the **Item** relation. The data type of this attribute is also the data type of **item_id** in the **Item** relation, which is **INT(8)**.
- quantity: The quantity of items the cart has. Therefore, it has a datatype **INT(3)** which is long enough to store each cart's quantity.
- price: The total price of the cart. Therefore, it has a data type **FLOAT** because the price can be in decimal number.
- created_by, updated_by: These attribute respectively record the time the cart is created and is last updated. Therefore, it has a data type **DATETIME**.

6.9 Booking

- token:This attributes is a text describing a promotion program of the restaurant and note of the order. Datatype **VARCHAR(50)** limits the content of the token below 50 characters.
- customerfname, customerlname: Datatypes of these attributes are **VARCHAR(45)** which is long enough to contain the first and last name of the customer
- tel, email: These attributes respectively have datatype **VARCHAR(10)** and **VARCHAR(45)** which respectively limit the length of attribute below 10 and 45 characters.

6.10 Table

- table_id: This attribute represents the unique ID of a table, because this specific restaurant have under 1000 tables so the type is **INT(3)**
- available: This attribute shows if the table is available or not, datatype **TINYINT** is an alternative to **BOOL**

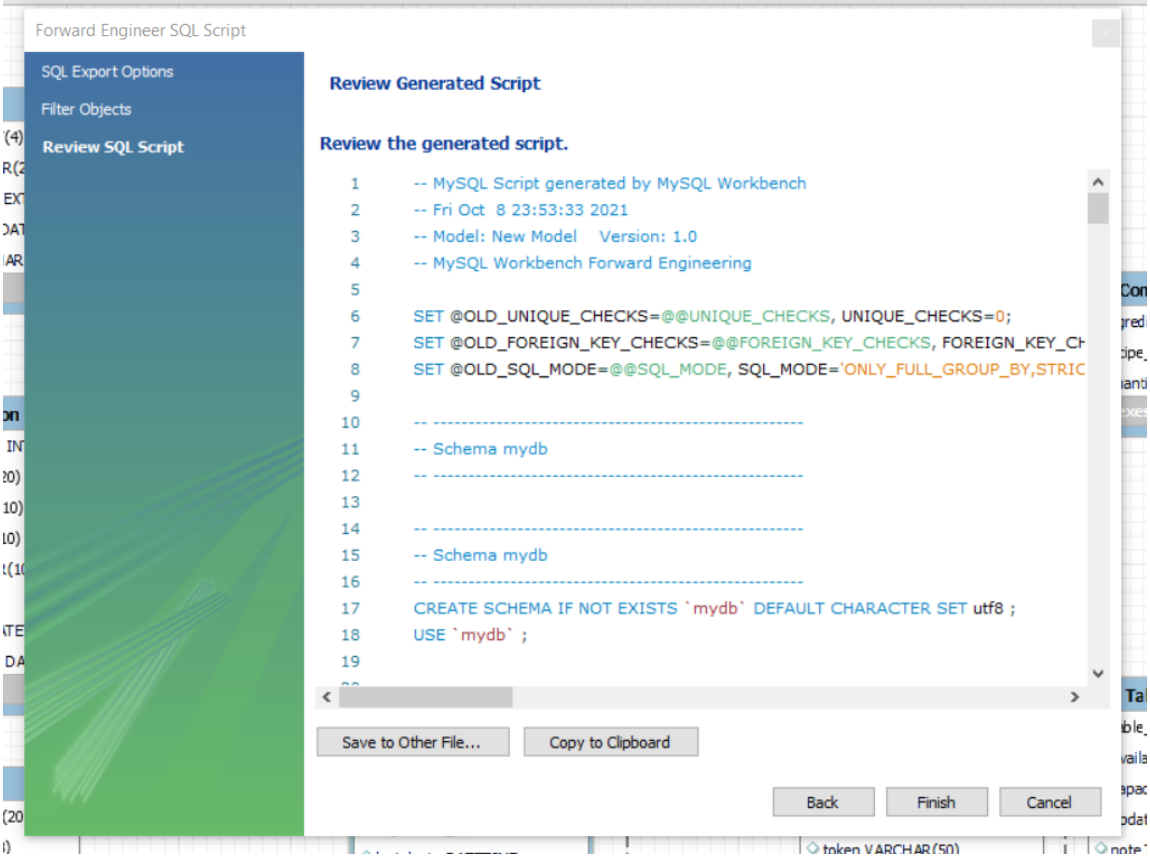
- capacity: The number of guests this table can serve, this specific restaurant has the largest table of under 100 guest so the type **INT(2)** is appropriate
- updated_by: The day this table was last modified (decoration, capacity... updates), if it was never modified, its default value is the day this table was purchased. Datatype is **DATETIME** to represent specific day and time
- note: If a customer has specific request for this table it will show here as **TEXT** because customer requests are unpredictable, they can be a few words or a whole list. Its default value is **NULL**

6.11 Ingredient

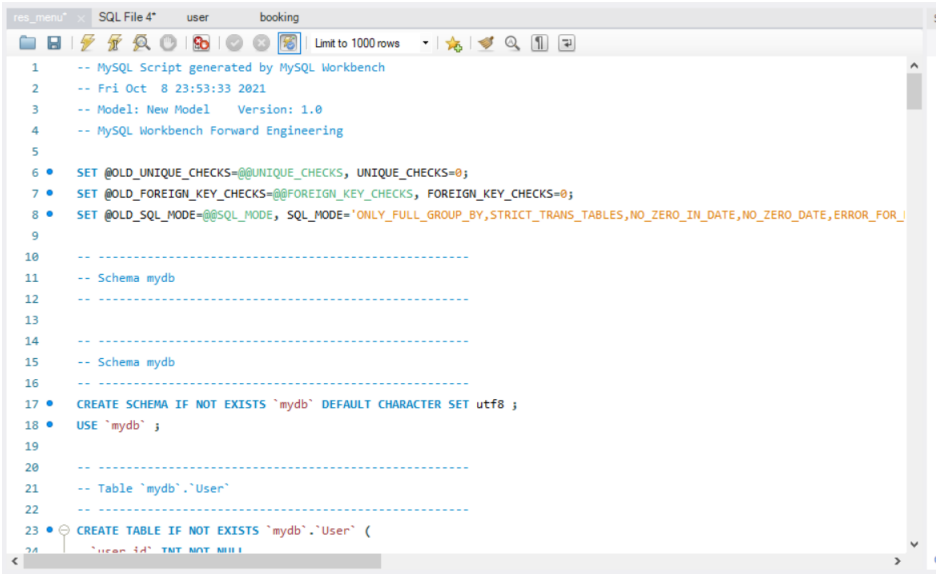
- ingredient_id: This attribute represents the unique ID of an ingredient. Therefore, it has a data type **INT(8)**, long enough to identify any kind of ingredients
- vendor_id: This attribute is the identification of the vendor who sold this ingredient, this attribute is a reference to **User.user_id**
- name: Name of the ingredient, **VARCHAR(45)** should be enough to name any ingredients
- type: Type of the ingredient (Veggie, Meat, Seafood...), a short description of the ingredient for easier querying, **VARCHAR(20)** is appropriate
- stock: This attribute shows the amount of one ingredient in Kg, so it has to be **FLOAT**
- imported_by, expired_by: These 2 attributes are the day this ingredient was purchased and the day that it will no longer usable. The type is **DATETIME** because they are both a specific date

7 DB system implementation

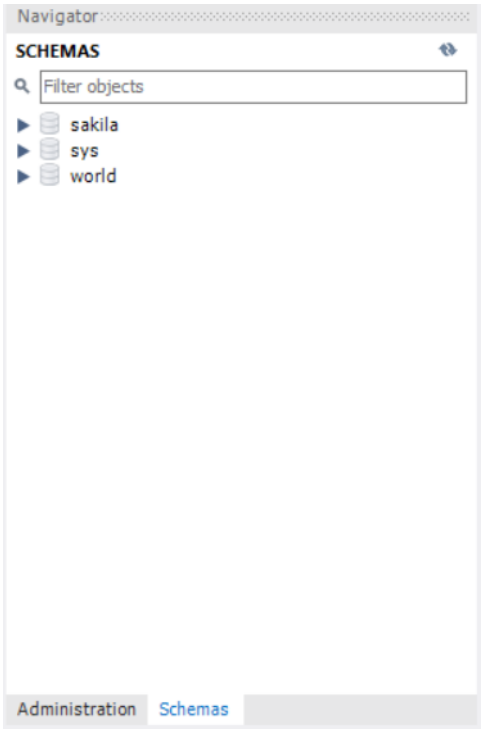
After drawing a relational diagram, we use the forward function in **MySQL** to export the code.



Then we copy it to clipboard and pass it into a script to run it.

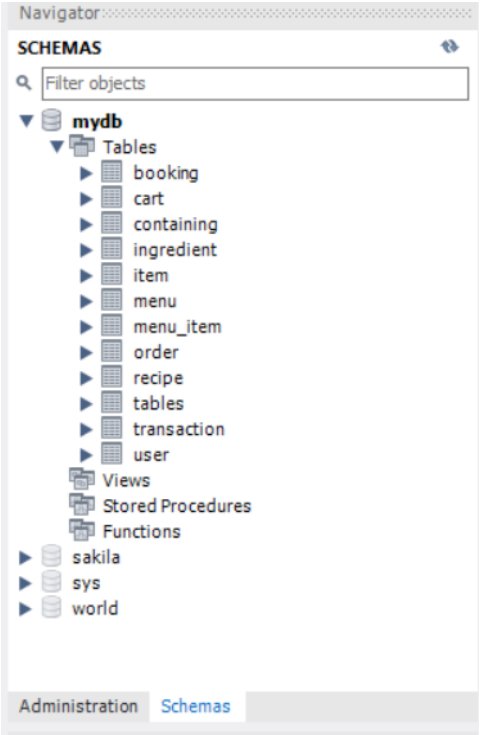


Before running the code the table had not existed.



However, after running the code, a schema **mydb** and tables are created.

STEP 1: CREATE A DATABASE WITH DEFAULT CONFIGURATION




```
CREATE SCHEMA IF NOT EXISTS `mydb` DEFAULT CHARACTER SET utf8 ;
USE `mydb` ;
```

CREATE DATABASE creates a database with the given name. To use this statement, we need the **CREATE** privilege for the database. **CREATE SCHEMA** is a synonym for **CREATE DATABASE**.

An error occurs if the database exists and we did not specify **IF NOT EXISTS**.

The CHARACTER SET option specifies the default database character set. In particular, we use **utf8**.

The USE statement tells MySQL to use the named database as the default (current) database for subsequent statements. This statement requires some privilege for the database or some object within it.

The named database remains the default until the end of the session or another USE statement is issued:

STEP 2: CREATE TABLE 'USER'

```
CREATE TABLE IF NOT EXISTS `mydb`.`User` (
  `user_id` INT NOT NULL,
  `name` VARCHAR(45) NULL,
  PRIMARY KEY (`user_id`))
ENGINE = InnoDB;
```

By default, the columns are able to hold **NULL** values. A **NOT NULL** constraint in SQL is used to prevent inserting **NULL** values into the specified column, considering it as a not accepted

value for that column. This means that you should provide a valid SQL **NOT NULL** value to that column in the **INSERT** or **UPDATE** statements, as the column will always contain data.

In this step, we create a table **USER** with two attributes **user_id** and **name** whose data type are integer and character of specified length, 45. Regarding to **user_id**, we put a constraint **NOT NULL**.

Attribute	Data Type	PK	FK	NN	UQ	B	UN	ZF	AI	G
user_id	INT(10)	✓								
fname	VARCHAR(30)									
lname	VARCHAR(30)									
tel	VARCHAR(10)									
email	VARCHAR(50)									
password_hash	VARCHAR(32)									
employee	TINYINT									
vendor	TINYINT									
customer	TINYINT									
registered_by	DATETIME									
last_login	DATETIME									

STEP 3: CREATE TABLE 'MENU'

```
CREATE TABLE IF NOT EXISTS `mydb`.`Menu` (
  `menu_id` INT(4) NOT NULL,
  `title` VARCHAR(20) NOT NULL,
  `description` TEXT NULL,
  `last_update` DATETIME NULL,
  `items` VARCHAR(45) NULL,
  PRIMARY KEY (`menu_id`))
ENGINE = InnoDB;
```

In a menu, each food is labeled with an identity number. In case that we use 4 bits to form a series of ID, we can create an array of 16 numbers which is enough for the number of food in a restaurant. The title of the menu is a short sequence of character, approximately 20 characters. The description is a long text having an unpredictable length. Therefore, we use **TEXT** as the data type for description. It is **DATETIME** that can keep **last_update**, an attribute for tracking the last time a menu is updated. Each item in the menu has its own name less than 45 characters.

Attribute	Data Type	PK	FK	NN	UQ	B	UN	ZF	AI	G
menu_id	INT(4)	✓								
title	VARCHAR(20)									
description	TEXT									
last_update	DATETIME									

Table 2 continued from previous page

Attribute	Data Type	PK	FK	NN	UQ	B	UN	ZF	AI	G
items	VARCHAR(45)									

STEP 4: CREATE TABLE 'ITEM'

```
CREATE TABLE IF NOT EXISTS `mydb`.`Item` (
  `item_id` INT(8) NOT NULL,
  `title` VARCHAR(45) NULL,
  `type` VARCHAR(45) NULL,
  `description` TEXT NULL,
  `cooking` TINYINT NULL,
  `price` FLOAT NULL,
  `stock` INT(3) NULL,
  `recipe` VARCHAR(45) NULL,
  `created_by` DATETIME NULL,
  `lastupdate` DATETIME NULL,
  PRIMARY KEY (`item_id`))
ENGINE = InnoDB;
```

Attribute	Data Type	PK	FK	NN	UQ	B	UN	ZF	AI	G
item_id	INT(8)	✓								
title	VARCHAR(45)									
type	VARCHAR(45)									
description	TEXT									
cooking	TINYINT									
price	FLOAT									
stock	INT(3)									
recipe	VARCHAR(45)									
created_by	DATETIME									
lastupdate	DATETIME									

STEP 5: CREATE TABLE 'MENU_ITEM'

```
CREATE TABLE IF NOT EXISTS `mydb`.`Menu_item` (
  `item_id` INT(8) NOT NULL,
  `menu_id` INT(4) NOT NULL,
  `description` TEXT NULL,
  `available` TINYINT NULL,
  PRIMARY KEY (`item_id`),
  INDEX `fk_Menu_item_Item1_idx` (`item_id` ASC) VISIBLE,
  INDEX `fk_Menu_item_Menu1_idx` (`menu_id` ASC) VISIBLE,
  CONSTRAINT `fk_Menu_item_Item1`
    FOREIGN KEY (`item_id`)
    REFERENCES `mydb`.`Item` (`item_id`)
    ON DELETE NO ACTION
```

```
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Menu_item_Menu1`
    FOREIGN KEY (`menu_id`)
    REFERENCES `mydb`.`Menu` (`menu_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Attribute	Data Type	PK	FK	NN	UQ	B	UN	ZF	AI	G
item_id	INT(8)	✓	✓							
menu_id	INT(4)		✓							
description	TEXT									
available	TINYINT									

STEP 6: CREATE TABLE 'RECIPE'

```
CREATE TABLE IF NOT EXISTS `mydb`.`Recipe` (
  `recipe_id` INT(8) NOT NULL,
  `item_id` INT(8) NOT NULL,
  `quantity` FLOAT NULL,
  `instruction` TEXT NULL,
  PRIMARY KEY (`recipe_id`),
  INDEX `fk_Recipe_Item1_idx` (`item_id` ASC) VISIBLE,
  CONSTRAINT `fk_Recipe_Item1`
    FOREIGN KEY (`item_id`)
    REFERENCES `mydb`.`Item` (`item_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Attribute	Data Type	PK	FK	NN	UQ	B	UN	ZF	AI	G
recipe_id	INT(8)	✓	✓							
item_id	INT(8)		✓							
quantity	FLOAT									
instruction	TEXT									

STEP 7: CREATE TABLE 'INGREDIENT'

```
CREATE TABLE IF NOT EXISTS `mydb`.`Ingredient` (
  `ingredient_id` INT(8) NOT NULL,
  `vendor_id` INT(10) NOT NULL,
  `name` VARCHAR(45) NULL,
  `type` VARCHAR(20) NULL,
  `stock` FLOAT NULL,
  `imported_by` DATETIME NULL,
  `expired_by` DATETIME NULL,
  PRIMARY KEY (`ingredient_id`),
  INDEX `fk_Ingredient_User1_idx` (`vendor_id` ASC) VISIBLE,
```

```
  CONSTRAINT `fk_Ingredient_User1`
    FOREIGN KEY (`vendor_id`)
    REFERENCES `mydb`.`User` (`user_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Attribute	Data Type	PK	FK	NN	UQ	B	UN	ZF	AI	G
ingredient_id	INT(8)	✓	✓							
vendor_id	INT(10)		✓							
name	VARCHAR(45)									
type	VARCHAR(20)									
stock	FLOAT									
imported_by	DATETIME									
expired_by	DATETIME									

STEP 8: CREATE TABLE 'CART'

```
CREATE TABLE IF NOT EXISTS `mydb`.`Cart` (
  `cart_id` INT(8) NOT NULL,
  `item_id` INT(8) NOT NULL,
  `quantity` INT(3) NULL,
  `price` FLOAT NULL,
  `created_by` DATETIME NULL,
  `updated_by` DATETIME NULL,
  PRIMARY KEY (`cart_id`),
  INDEX `fk_Cart_Item1_idx` (`item_id` ASC) VISIBLE,
  CONSTRAINT `fk_Cart_Item1`
    FOREIGN KEY (`item_id`)
    REFERENCES `mydb`.`Item` (`item_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Attribute	Data Type	PK	FK	NN	UQ	B	UN	ZF	AI	G
cart_id	INT(8)	✓	✓							
item_id	INT(8)		✓							
quantity	INT(3)									
price	FLOAT									
created_by	DATETIME									
updated_by	DATETIME									

STEP 9: CREATE TABLE 'ORDER'

```
CREATE TABLE IF NOT EXISTS `mydb`.`Order` (
  `order_id` INT(20) NOT NULL,
  `cart_id` INT(8) NOT NULL,
  `token` VARCHAR(50) NULL,
  `subtotal` FLOAT NULL,
  `Ordercol` VARCHAR(45) NULL,
  `discount` FLOAT NULL,
  `tax` FLOAT NULL,
  `shipping` FLOAT NULL,
  `total` FLOAT NULL,
  `createdby` DATETIME NULL,
  `updated_by` DATETIME NULL,
  `note` TEXT NULL,
  PRIMARY KEY (`order_id`),
  INDEX `fk_Order_Cart1_idx` (`cart_id` ASC) VISIBLE,
  CONSTRAINT `fk_Order_Cart1`
    FOREIGN KEY (`cart_id`)
    REFERENCES `mydb`.`Cart` (`cart_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Attribute	Data Type	PK	FK	NN	UQ	B	UN	ZF	AI	G
order_id	INT(20)	✓	✓							
cart_id	INT(8)		✓							
token	VARCHAR(50)									
subtotal	FLOAT									
Ordercol	VARCHAR(45)									
discount	FLOAT									
tax	FLOAT									
shipping	FLOAT									
total	FLOAT									
createdby	DATETIME									
updated_by	DATETIME									
note	TEXT									

STEP 10: CREATE TABLE 'TRANSACTION'

```
CREATE TABLE IF NOT EXISTS `mydb`.`Transaction` (
  `transaction_id` INT NOT NULL,
  `buyer_id` INT NOT NULL,
  `status` INT(1) NULL,
  PRIMARY KEY (`transaction_id`),
  INDEX `fk_Transaction_User3_idx` (`buyer_id` ASC) VISIBLE,
  CONSTRAINT `fk_Transaction_User3`
    FOREIGN KEY (`buyer_id`)
    REFERENCES `mydb`.`User` (`user_id`)
```

```
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Attribute	Data Type	PK	FK	NN	UQ	B	UN	ZF	AI	G
transaction_id	INT(20)	✓								
order_id	INT(20)		✓							
buyer_id	INT(10)		✓							
seller_id	INT(10)		✓							
type	VARCHAR(10)									
status	INT(1)									
created_by	DATETIME									
completed_by	DATETIME									

STEP 11: CREATE TABLE 'TABLES'

```
CREATE TABLE IF NOT EXISTS `mydb`.`Tables` (
  `table_id` INT(3) NOT NULL,
  `available` TINYINT NULL,
  `capacity` INT(2) NULL,
  `updated_by` DATETIME NULL,
  `note` TEXT NULL,
  PRIMARY KEY (`table_id`))
ENGINE = InnoDB;
```

Attribute	Data Type	PK	FK	NN	UQ	B	UN	ZF	AI	G
table_id	INT(3)	✓								
available	TINYINT									
capacity	INT(2)									
updated_by	DATETIME									
note	TEXT									

STEP 12: CREATE TABLE 'BOOKING'

```
CREATE TABLE IF NOT EXISTS `mydb`.`Booking` (
  `booking_id` INT(8) NOT NULL,
  `user_id` INT(10) NOT NULL,
  `cart_id` INT(8) NOT NULL,
  `tables_id` INT(3) NOT NULL,
  `token` VARCHAR(50) NULL,
  `status` INT(1) NULL,
  `customer_fname` VARCHAR(45) NULL,
  `customer_lname` VARCHAR(45) NULL,
  `tel` VARCHAR(10) NULL,
  `email` VARCHAR(45) NULL,
  `created_by` DATETIME NULL,
  `completed_by` DATETIME NULL,
  PRIMARY KEY (`booking_id`),
```

```
  INDEX `fk_Booking_User1_idx` (`user_id` ASC) VISIBLE,
  INDEX `fk_Booking_Tables1_idx` (`tables_id` ASC) VISIBLE,
  INDEX `fk_Booking_Cart1_idx` (`cart_id` ASC) VISIBLE,
  CONSTRAINT `fk_Booking_User1`
    FOREIGN KEY (`user_id`)
    REFERENCES `mydb`.`User` (`user_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Booking_Tables1`
    FOREIGN KEY (`tables_id`)
    REFERENCES `mydb`.`Tables` (`table_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Booking_Cart1`
    FOREIGN KEY (`cart_id`)
    REFERENCES `mydb`.`Cart` (`cart_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Attribute	Datatype	PK	FK	NN	UQ	B	UN	ZF	AI	G
booking_id	INT(8)	✓								
user_id	INT(10)		✓							
cart_id	INT(8)		✓							
tables_id	INT(3)		✓							
token	VARCHAR(50)									
status	INT(1)									
customer_fname	VARCHAR(45)									
customer_lname	VARCHAR(45)									
tel	VARCHAR(10)									
email	VARCHAR(45)									
created_by	DATETIME									
completed_by	DATETIME									

Table 11: Booking table

STEP 13: CREATE TABLE 'CONTAINING'

```
CREATE TABLE IF NOT EXISTS `mydb`.`Containing` (
  `ingredient_id` INT(8) NOT NULL,
  `recipe_id` INT(8) NOT NULL,
  `Quantity` FLOAT NULL,
  INDEX `fk_Containing_Ingredient1_idx` (`ingredient_id` ASC) VISIBLE,
  INDEX `fk_Containing_Recipe1_idx` (`recipe_id` ASC) VISIBLE,
  CONSTRAINT `fk_Containing_Ingredient1`
    FOREIGN KEY (`ingredient_id`)
```

```
    REFERENCES `mydb`.`Ingredient` (`ingredient_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Containing_Recipe1`
    FOREIGN KEY (`recipe_id`)
    REFERENCES `mydb`.`Recipe` (`recipe_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

This table is created to represent the relationship of ingredient and recipe; it is, therefore, not necessary to point out a primary key.

Attribute	Data Type	PK	FK	NN	UQ	B	UN	ZF	AI	G
ingredient_id	INT(8)	✓	✓							
recipe_id	INT(8)	✓	✓							
Quantity	FLOAT									