**References for Administrator and user**

1. Administrator Part:
2. The Database Design:

Requirements analysis:

For this Hotel Management system, it needs two major parts including room booking or cancelling part and meal booking or cancelling part. Meanwhile, it has two branches, which are guest branch and staff branch.

The guest should be able to register their account at first, then they can use this account to login the system for room booking/cancelling and meal booking/cancelling. Guests must provide personal information to register account: 1. Username, 2. Their Real Name, 3. Passport ID, 4. Telephone Number, 5. Email Address. The information updating is optional in the requirements. To login account of guests, they need to type in their username and password correctly. Afterwards, if they login successful, (Use correct username and password) they will have some options such as “Book Room”, “Book Meal”, “Cancel Room”, “Cancel Meal”, and “Log out”. (Important options) When guest booking a room, they need to specify the check-in date, check-out date, and room type. If there are more than one available room are consistent with their specified condition, then the system would allow them to choose the specific room they want to stay.

For the food service of guests, they can book the meal two days in advance. The chefs only work in certain days in each week. Most of the meals are different in their meal lists, but two meals could be cooked by three of them. These meals are “fried egg” and “curry rice”. Meanwhile, the meal is only available between 7:00AM and 10:00PM.

For the staff part, their account should be associated with “Username”, “Password”, “Telephone Number”, and “Staff ID”. Staff should be able to check the room booking situation at least.

Table Design in Database:

Table Design:

I have created 10 tables to store all information, the name of these tables listed as below:

(Due to the Server connection collation (Character Set) is “utf8mb4\_unicode\_ci” (case-insensitive), the actual name in the database schema is a little different from the presented name in this list. For instance, “RoomType” shown as “roomtype” in the database.)

1. Guest
2. Staff
3. Room
4. RoomType
5. BookedRoom
6. Chef
7. Meal
8. Schedule (This name is a key word in SQL, you need to add “ ` ” such as `Schedule` to avoid accidents)
9. OneWeek
10. BookedMeal
11. Guest Table has 8 attributes in it.

These attributes include “userID”, “username”, “password”, “realName”, “passportID”, “telephoneNumber”, “email”, and “operationTime”.

The `userID` is an **INT** type attribute, it has a **Primary Key (UNSIGNED AUTO\_INCREMENT)** of this table, and it will not be presented to guests. This is because the `userID` is only used for record and prevent duplicates. Furthermore, the `userID` would be utilised as a communication code with different tables or function blocks in the Java program.

The `username` attribute could store the specified username of each guest. It has a **UNIQUE** Key (and **NOT NULL**) to prevent username duplicates and it is a **VARCHAR (255)** attribute in this table.

The `password` (**VARCHAR (255) NOT NULL**) is similar to the `username` attribute. However, it does not have a UNIQUE key restriction because the passwords from different guests could be the same.

The `realName` (**VARCHAR (255) NOT NULL**) is an attribute to store the Real Name of the guest. They cannot choose to skip filling in the blank of this attribute when they are registering an account.

`passportID` (**VARCHAR (9) NOT NULL**) is an attribute to store the Passport ID of each guest. Due to the international standards, **the length of Passport ID cannot exceed 9 characters and digits combination**.

`telephoneNumber` (**BIGINT UNSIGNED NOT NULL**) is an attribute to store the Phone Number of guests. The Phone Number format in this system is the same as the guests’ Phone Number **excluding blank, underscore, or dashed line (“ + ”, “ ”, “ \_ ”, “ - ”)**. For example, “+86 12332112356” should be written as “8612332112356”.

`email` (**VARCHAR (255) DEFAULT ‘’**) attribute is used to store the email of guests. It is an optional attribute because there is a possibility that the guest does not have an email. Thus, it could choose to skip the filling step on this attribute.

`operationTime` (**TIMESTAMP NOT NULL ON UPDATE CURRENT\_TIMESTAMP**) is an attribute to capture the latest information modified date and time. It would be initialised by “NOW()” function in the Java program. It could also be utilised to promote this system with mechanism of optimistic locking to support multiple threads in the future.

An improvement suggestion of this table: The `username` and `password` are always used when guests log into their accounts. Meanwhile, the DBMS will scan through the entire table to find corresponding data. Thus, a large-scale Guest table which includes detailed information of each guest would reduce the running efficiency. One possible solution is to separate the guest table into two tables. The `Login` table only has userID (Primary Key), username and password, but another `Information` table has the detailed information of each guest. This `Information` table also has the `userID` attribute and reference the Primary Key of `Login` table. This solution will not only enhance the running efficiency of DBMS, but also provide other possibilities which means that users could add multiple personal information as the alternative information in the database. The reason why I do not choose to apply the above solution into this table is that the data set is not extremely large at the initial stage. Although the solution provided above has two advantages, the Cartesian Product will decrease the running efficiency when guests want to update their personal information. Therefore, the Cartesian Product is a more noticeable problem compared with scanning efficiency reduction problem when the data of guests is rare. Consequently, I chose to use the large `Guest` table in this program. In the future, when the data of guests are numerous, I could consider alter the table structure as the same one as the solution stated above.

1. Staff Table has 5 attributes.

`staffID` (**INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT**) is used to distinguish the different staff in this hotel.

`username` (**VARCHAR (255) NOT NULL**) is an attribute to store the staff name or alias in this table.

`password` (**VARCHAR (255) NOT NULL**) is used for staff login operation.

`telephoneNumber` (**BIGINT UNSIGNED NOT NULL**) is an attribute to store the phone number of staff. It has the same format rule as guests phone number input in the Java program.

`operationTime` (**TIMESTAMP NULL DEFAULT NULL ON UPDATE CURRENT\_TIMESTAMP**) could record the latest update time on staff information. (Staff could only change their password in this system.) This attribute is also set for future function extension, it can cooperate with optimistic locking to implement a multiple threads operation.

Notice: The new Staff could only register their account by submitting their application to the Administrator of this database. Afterwards, the Administrator would tell him or her the `username` and default `password` in this hotel management system (HMS). They are recommended to change their `password` as soon as possible after they get their own account.

The improvement suggestion of this table: (Almost the same as the Guest Table improvement suggestion. However, this alteration method is not very recommended for this table, because staff do not have complex personal information compared with guests’ information.)

1. Room Table has 2 attributes.

`roomID` (**INT UNSIGNED PRIMARY KEY**) is used to store the Room ID of each floor, it was formatted as “XY” (X stands for the number of floors, Y stands for the room number in each floor.) For example, the room number 6 in the 6th floor is “606”.

`roomTypeID` (**TINYINT UNSIGNED NOT NULL**) is specify the four different types of room in the hotel. It references the `roomTypeID` attribute in `RoomType` Table.

1. RoomType Table has 2 attributes.

`roomTypeID` (**TINYINT UNSIGNED PRIMARY KEY AUTO\_INCREMENT**) is used as the index to represent 4 types of room in this hotel.

`roomType` (**VARCHAR (30) NOT NULL**) stored the actual name of these four types of room.

1. BookedRoom Table has 6 attributes.

`bookedRoom\_ID` (**INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT**) is an order code of booked room. It can separate different booked room orders with different ID code. This design could help guests to modify their booked room with only order codes specified in program. It is more convenient than input other information to specify a room order.

`userID` (**INT UNSIGNED NOT NULL**) is used to specify who want to live in the hotel. It has referenced `userID` attribute in Guest Table.

`roomID` (**INT UNSIGNED NOT NULL**) is used to specify which room that the guest wants to live in. It has referenced `roomID` attribute in Room Table.

`checkInDate` (**DATE NOT NULL**) stores the guest’s check-in date.

`checkOutDate` (**DATE NOT NULL**) stores the guest’s check-out date.

(Combined two attributes above determine the time interval that a specific guest live in the hotel.)

`operationTime` (**TIMESTAMP NOT NULL ON UPDATE CURRENT\_TIMESTAMP**) records the latest modify time of a booked room order. It would be initialised by “NOW()” function in the Java program. This attribute could cooperate with optimistic locking to implement a multiple threads operation.

1. Chef Table has 2 attributes.

`chefID` (**TINYINT UNSIGNED PRIMARY KEY AUTO\_INCREMENT**) stands for the ID of different chefs. It will work as a communication code in the Java program. Due to the number of chefs are not extremely large, the `chefID` was set to TINYINT data type to save the storage (or memory) space. It could be changed to INT if the number of chefs is exceeded 127.

`chefName` (**VARCHAR (20) NOT NULL**) stores the Real Name of chefs. Because of the longest name in these chefs are not exceed 20 characters, the VARCHAR was set to 20 length for storage (or memory) saving.

1. Meal Table has 4 attributes.

`dishesType\_ID` (**INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT**) is the ID of different combination between dishes and chefs. It shows all available meal in the hotel.

`chefID` (**TINYINT UNSIGNED NOT NULL**) refers to the chef of each meal. It has referenced the attribute `chefID` in Chef Table.

`dishes` (**VARCHAR (60) NOT NULL**) stores the name of each dish. Each dish corresponds to a chef in this table. For the aim of storage or memory space saving, the VARCHAR is set to 60 characters. It can be changed if new added name of dish exceeds this length.

`price` (**FLOAT UNSIGNED NOT NULL**) stores the price of per dish. It could be a float type data, but in this table all price data are integer. However, the new added dish might have float point number of prices.

An improvement suggestion of this table: This table could be separated into two tables, because there exists a condition that one meal corresponds to three chefs. According to Normalisation, it should be separated into two tables consistent with the third Normal form. However, there are only two meals that have this type of relationship with chefs. Meanwhile, the final result of these two separated tables should be recombined together to display a valid relation. It is noticeable that there exists Cartesian Product that will reduce the running efficiency. Therefore, I stayed it as an entire Meal Table at this initial stage. If the meal and chefs have increased to a large number and the “1:m” relationship has become regular, I will choose to separate this table to consistent with the third normal form.

1. `Schedule` Table has 3 attributes.

`weekday\_ID` (**INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT**) stores the different ID to separate different chefs and their workday.

`chefID` (**TINYINT UNSIGNED NOT NULL**) stores the chefs’ ID. It has referenced the `chefID` attribute in Chef Table.

`day\_ID` (**TINYINT UNSIGNED NOT NULL**) stores the corresponding workday for different chefs. It has referenced `day\_ID` attribute in OneWeek Table.

1. OneWeek Table has 2 attributes.

`day\_ID` (**TINYINT UNSIGNED PRIMARY KEY AUTO\_INCREMENT**) stores the corresponding day number for one week. It is reasonable that one week could be represented by 7 different numbers. Therefore, it does not need to declare a large data type for this attribute. The TINYINT type is the appropriate minimum data type for this range.

`day\_Name` (**VARCHAR (10) NOT NULL**) stores the full text name of each day in one week. Because of the name which has the maximum length in one week is not exceed 10 characters, the VARCHAR set as 10 characters for storage or memory space saving.

1. BookedMeal Table has 8 attributes.

`bookedMeal\_ID` (**INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT**) stores the order code of meal. It can help guests to modify the meal in convenient approach.

`userID` (**INT UNSIGNED NOT NULL**) stores the guest who book the meal. It has referenced `userID` attribute inGuest Table.

`bookedRoom` (**BIT NOT NULL**) verify that whether the customer living in a room when he or she enjoying the meal in the hotel. “0” stands for false (Not live in the hotel) and “1” stands for true (Live in the hotel).

`dishesType\_ID` (**INT UNSIGNED NOT NULL**) stores the meal type and chef. This attribute represents most of meal information in each tuple in this table. It has referenced `dishesType\_ID` attribute in Meal Table.

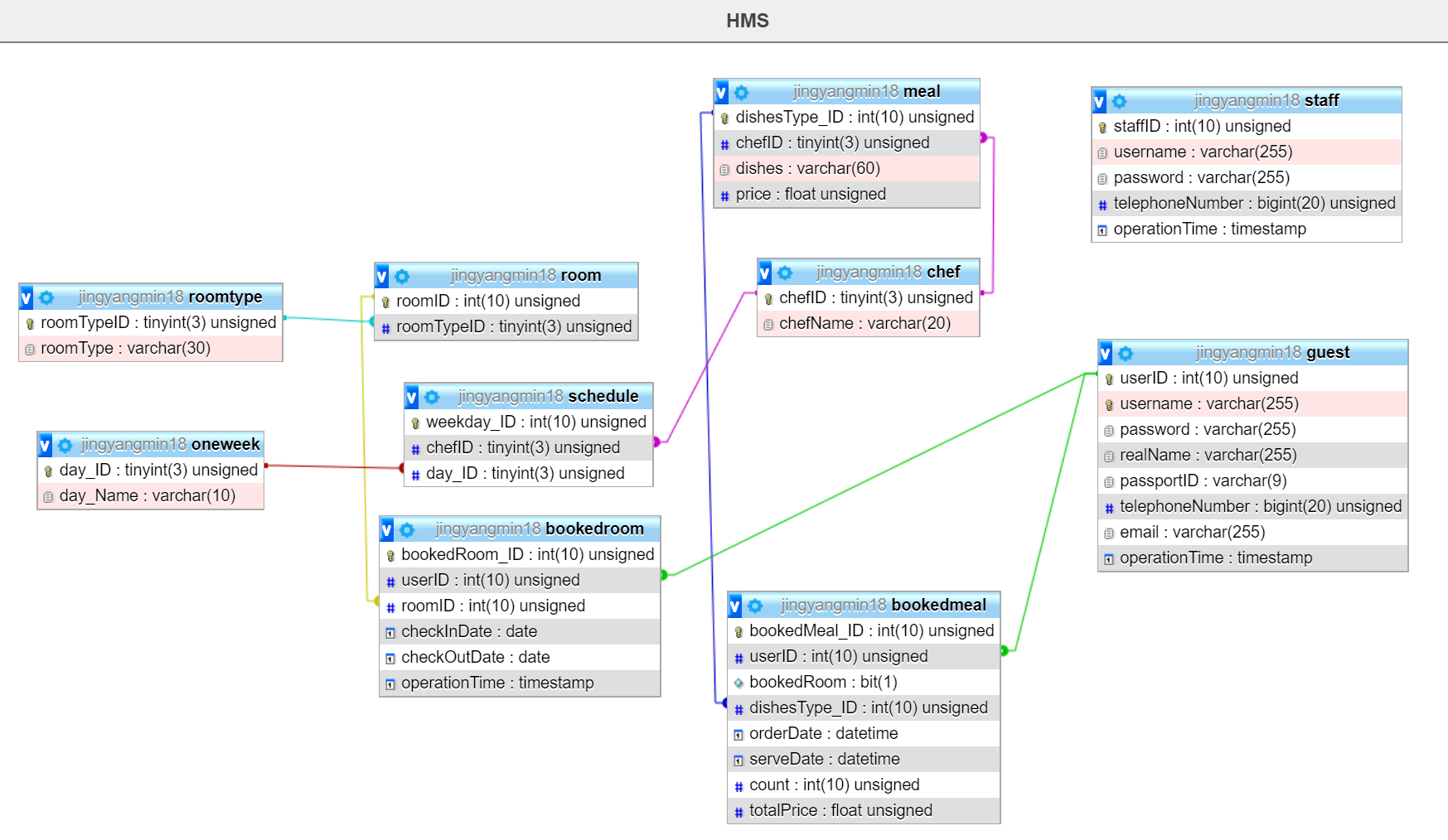
`orderDate` (**DATETIME NOT NULL**) stores the order date and time of guests.

`serveDate` (**DATETIME NOT NULL**) stores the service date and time for guests to enjoy the meal.

`count` (**INT UNSIGNED NOT NULL DEFAULT 1**) stores the specified count for each meal by guests. The default count for each dish is 1.

`totalPrice` (**FLOAT UNSIGNED NOT NULL**) stores the total price of each meal order. If the guest book the meal two days in advance and living in hotel on the service date, they will get 20% discount on the total price.

The Entity-Relationship (E/R) Diagram:



**Figure 1:** The E/R Diagram of all relations in Hotel Management System

1. The Program Design:

Although the MySQL DBMS has provided numbers of approaches to implement data storage issue, the table design is also important because it will affect program and DBMS running efficiency. In this program, I have not chosen to utilise the absolutely OOP concept when I was designing it. It is feasible that directly insert an object into one table, but it is not necessary. This is because the user inputs should be checked before insert these data into a table, and create an object is still need these processes. It is noticeable that create an object for “Guest” or “Booked Room” would need intermediate steps. Thus, I just verify these data at first and then insert them into a table directly.

In this program, to establish the connection with the specified database is one of the most important tasks. In order to get a connection with the XJTLU mariaDB database schema which has been named as **jingyangmin18**, (The URL is **jdbc:mysql://csse-mysql.xjtlu.edu.cn:3306/jingyangmin18**) we need to configure the Connector/J package (JDBC) into the library of our project in the IDE. I used **jdk1.8.0\_251** and **mariadb-java-client-2.6.0.jar** in my **IntelliJ IDEA project**. To optimise the configuration efficiency, I used **java.util.ResourceBundle** to directly configure the URL, Username, Password in it. You can also find it in (Path: project/Hotel\_Management\_System/src/information.properties). When you open this properties file, you will find 3 configuration contents including: **1. URL, 2. administrator, 3. password**. You can modify the contents shown on the right side of equal sign (“=”) to connect the program to your own database. The prepared contents in this file are the configuration of my database schema, you can also unchanged it and directly connect to my database.

All the function in this program have been divided into 8 classes. The project includes one main class named as “HMS”, and a package named as “utils”. Almost all the function was implemented in this utils package. This utils package includes: “Guest”, “Staff”, “Booking”, “StaffOperation”, “DB\_Utility”, “TablePrinter”, and “ShowWindow”. The last two class is used to display the table in GUI format. It involves java swing to implement this function.

This program could be descripted as a robust system, it supports not only number codes interaction, but also supports full text name interaction, message verification (with regular expression), transaction, and a little function about concurrency supporting. This program also provided large extension space for following developers to create new function in its body. Because of my design concept is to directly insert data into table, so most of classes in the program is not support object instantiation. To prevent it happened, except the “ShowWindow” class, the constructors of other classes have been already set to private and prevent object instantiation from out side of their class body.

For the different panel display, I used “byte code” to implement the communication among different function blocks. For example, 0 stands for welcome page, 1 stands for Guest welcome page, 2 for Staff welcome page, -1 for quit the system. There are 23 different codes in the program, each of them has different function. They could be separated into different function blocks, but all of them combined together to form an entire HMS program. Therefore, it has a high portability because of different part is not very tightly connected together.

The program also shown accurate “exception” message to user, these messages are surrounded or upper bounded by “=======” sign to help users identify what error they made in the past time typing contents.

For more detailed information of this program, you can also refer to the Javadoc comments in these source codes.