University of British Columbia, Department of Computer Science

CPSC 304

2016 Winter Term 1

Project Part 3

Group Name: g0z9a

Group Members:

Name	Student Number	Unix ID	Email Address
Jianjun Liu	40167141	h2e0b	liu31812202@hotmail.com
Le Lin	34403148	i8z9a	linle527@gmail.com
Shiyang Li (Melauria)	48753140	h0b0b	melauria0126@hotmail.com
Yumeng Chen	35365148	g0z9a	marsrabbit.meng@gmail.com

By typing our names and student numbers in the above table, we certify that the work in the attached assignment was performed solely by those whose names and student IDs are included above.

In addition, we indicate that we are fully aware of the rules and consequences of plagiarism, as set forth by the Department of Computer Science and the University of British Columbia.

Project Accomplishment:

In this project, we have accomplished to create a restaurant database system. We have created an integrated database for 2 user segments – customers and admins. Within 2 user segments are 4 separate user types, namely Customers, Servants, Chefs, and Managers. There are also a total of 9 user interfaces (UIs) in this system, and they have been grouped together by their functionalities and the user segment they serve.

The Customer User Segment –

Within the Customers user segment is the customer user type, and there are 5 UIs, namely Index UI, Register UI, Order UI, Payment UI and Consult UI.

Index UI: Each customer is welcomed by the Index UI first, which allows users to log-in. Member customers can directly login with his name and phone number, whereas non-members have the option to register or login as regular customers. Both members and non-members is directed to the Order UI after they log-in, and non-members who want to register is directed to the Register UI.

Register UI: Non-member can register in the register UI with their name and phone numbers. They are directed back to the Index UI after a successful registration.

Order UI: A menu of items available is displayed to the customer through the Order UI, and the customer is able to order directly through the system. While he does so, he is able to see a total summary of his order, including a list of the chosen items and the sum of prices of the chosen items. Once a customer confirms the order, he is directed to the to the Payment UI.

Payment UI: The Payment UI displays a summary of his orders, the time and date of this bill being produced, an option of paying with debit, credit or cash and a 'confirmation' button for him to confirm the bill. When the bill is paid and the order is confirmed, all dishes ordered are sent to the kitchen through our system and a record instance of this bill is added into our database. The customer is then directed to the Consult UI.

Consult UI: A customers checks the status (queued/processing/done) of all dishes in his order through the consult UI. He can also use the consult UI to see what others has order, and the popularity of a particular dish (number of people that have ordered that dish).

The Admin User Segment –

Within the Admin user segment are e user types – servants, chefs, and managers. There are 4 UIs, namely Admin UI, Cook UI, Servant UI,

Admin UI: For all admin users, the initial interface is the Index UI, to jump to the Admin UI an admin user needs to click on the 'Go To Admin Login' button. The admin user enter his name and 'userID', the system automatically detects whether this user is a chef, servant or manager, and then directs him towards the corresponding UI.

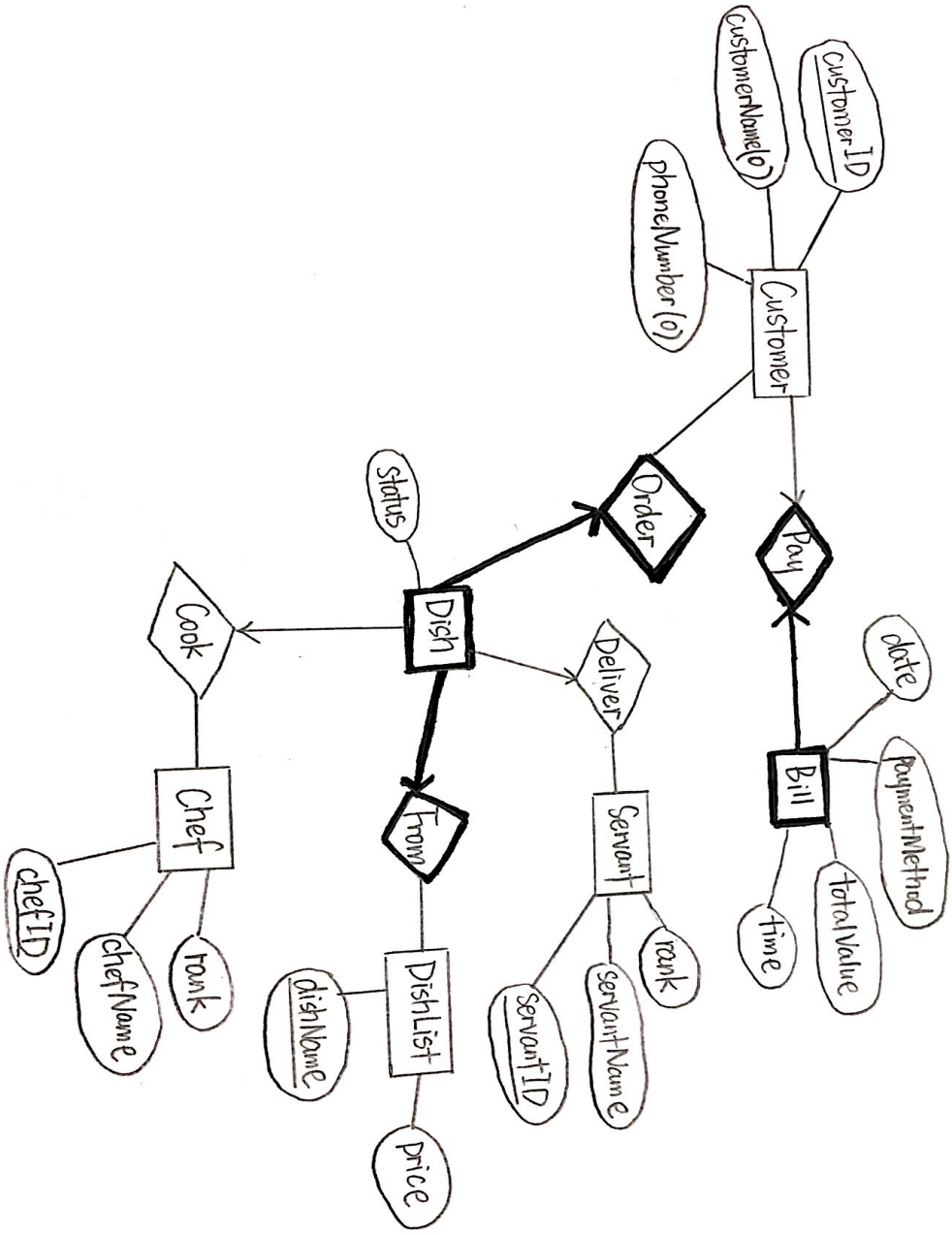
Cook UI: The Cook UI is the UI for all chefs. Chefs in the kitchen are able to monitor the complete list of dish with status 'queued'. Once a chef start making a dish, he changes the status of this particular dish to 'processing'. Once a dish's status is changed to 'processing' by any chef, the customer cannot choose to cancel this order anymore. When the chef finishes making a dish the servant come to the kitchen counter, pick up the dish and deliver it to the corresponding customer.

Servant UI: The servant UI is the UI for all servants. Servant users are able to use this system to monitor all dishes with the status 'processed'. The servant picks up a 'processed' dish from the kitchen, deliver it to the particular customer, and then change the status of this dish to 'done' with the system.
Manager UI: The manager UI is the interface for the manager. The manager can do several things in his interface, Business-wise, he can check the most popular dish of the day or the total sales of the day. Operation-wise, the manager of this restaurant can add and delete chefs and servants from the database. Furthermore, by the end of an business day of the restaurant, he needs to delete all Customer, 'Pay_Bill' and 'Ordered Dish' data of non-member customers, and all 'Pay_Bill' and Ordered Dish data of member customers from the database by the end of the day

Differences in the schema

1) We chose to remove the member schema that was in the original plan. Instead, we chose to implement the differences between member and non-member customers into the 'Customer' Schema (if the 'customerName' and 'phoneNumber' of the customer is not null, then this customer is a member).
   Reason: In the original plan, member customers was considered as the sub-class of the customer class. However, it was particularly hard to implement because we also wanted to realize the possibility of a non-member customer to become a member through self-registration.
2) We chose to remove the functionality of 'billID' and 'dishID' as a part of the primary key in the 'Pay_Bill' Schema and 'Order_From_Cook_Deliver_Dish' Schema. Our new implementation is to set 'customerID' as the only primary key in both schemas.
   Reason: With the implementation of MySQL auto-increment on 'customerID', it was enough to be a unique identifier in both schemas. Therefore, by removing other unnecessary keys, we reduce the number of MySQL auto-increment used in the code, hence we simplified the implementation of our code, while no changes to functionality was sacrificed.
3) We added an attribute called 'Rank' in the 'Chef' Schema and 'Servant' Schema. This represents the rank of a particular chef or servant.
   Reason: This change was made to demonstrate a join behavior in the demo.
4) We added a schema called 'DishList'. This 'DishList' schema is a weak-entity of the 'Dish' Schema.
   Reason: This 'DishList' schema is used to store all the available dishes in the menu, so customers must order from the dishes given inside 'DishList'. If this schema does not exist, customers would then be able to create dishes on their own.
5) We removed the 'tableID' attribute from the 'Order_From_Cook_Deliver_Dish' Shema.
   Reason: The use of 'CustomerID' as primary key of Customer instead of 'tableID' prevents the situation that if we want to keep a member's information, we have to keep a certain table for this member, and nobody else can sit on that table. This doesn't conform the reality so we cancelled 'tableID'.

Updated ER diagram:



SQL Queries used

Customer UIs:

Index:

1. SELECT * FROM Customer WHERE CustomerName=:username AND PhoneNumber=:password
2. Insert into Customer (CustomerName,PhoneNumber) values (Null,Null)
3. SELECT MAX(CustomerID) FROM Customer

Register:

1. Insert into Customer (CustomerName,PhoneNumber) values ('$username','$password')

Order:

1. INSERT INTO Order_From_Cook_Deliver_Dish VALUES ('$cid', '$dname', Null, Null, 'Onhold')
2. SELECT * FROM Order_From_Cook_Deliver_Dish WHERE CustomerID='$cid' AND DishName='$dname'
3. DELETE FROM Order_From_Cook_Deliver_Dish WHERE CustomerID='$cid' AND DishName='$dname'
4. SELECT * FROM DishList
5. SELECT DishList.DishName, Price
   FROM Order_From_Cook_Deliver_Dish, DishList
   WHERE DishList.DishName = Order_From_Cook_Deliver_Dish.DishName AND CustomerID='$cid'
6. SELECT CustomerName
   FROM Customer
   WHERE CustomerID=:cid
7. SELECT SUM(Price)
   FROM DishList, Order_From_Cook_Deliver_Dish
   WHERE DishList.DishName=Order_From_Cook_Deliver_Dish.DishName AND Order_From_Cook_Deliver_Dish.CustomerID = :cid

Payment:

1. INSERT INTO Pay_Bill VALUES ('$cid', '$time', '$date', '$paymenttype', '$actualprice')
2. UPDATE Order_From_Cook_Deliver_Dish SET Status='Queued' WHERE CustomerID='$cid'
3. SELECT SUM(Price)
   FROM DishList, Order_From_Cook_Deliver_Dish
   WHERE DishList.DishName=Order_From_Cook_Deliver_Dish.DishName AND Order_From_Cook_Deliver_Dish.CustomerID = '$cid'

Consult:

1. SELECT DishName, Status
   FROM Order_From_Cook_Deliver_Dish
   WHERE CustomerID=:cid
2. SELECT CustomerID, Sum(Price) as TOTALPRICE
   FROM Order_From_Cook_Deliver_Dish, DishList
   WHERE CustomerID<>:cid AND DishList.DishName=Order_From_Cook_Deliver_Dish.DishName

GROUP BY CustomerID;

3. SELECT CustomerID, Count(DishName) AS NumberDish
   FROM Order_From_Cook_Deliver_Dish
   WHERE CustomerID<>:cid
   GROUP BY CustomerID

Admin UIs (Chef, Servant, Manager):

Admin:

1. SELECT *
   FROM Manager
   WHERE ManagerName=:adminusername AND ManagerID=:adminpassword
2. SELECT *
   FROM Chef
   WHERE ChefName=:adminusername AND ChefID=:adminpassword
3. SELECT *
   FROM Servant
   WHERE ServantName=:adminusername AND ServantID=:adminpassword

Cook:

1. SELECT CustomerID, DishName
   FROM Order_From_Cook_Deliver_Dish
   WHERE Status = 'Queued'
2. UPDATE Order_From_Cook_Deliver_Dish SET ChefID = '$chid' WHERE CustomerID = '$cid' AND DishName = '$dname'
3. UPDATE Order_From_Cook_Deliver_Dish SET Status = 'Processing' WHERE CustomerID = '$cid' AND DishName = '$dname'

Deliver:

1. UPDATE Order_From_Cook_Deliver_Dish SET ServantID = '$sid' WHERE CustomerID = '$cid' AND DishName = '$dname'
2. UPDATE Order_From_Cook_Deliver_Dish SET Status = 'Done' WHERE CustomerID = '$cid' AND DishName = '$dname'
3. SELECT CustomerID, DishName
   FROM Order_From_Cook_Deliver_Dish
   WHERE Status = 'Processing'

Manage:

1. SELECT ServantName, ChefName
   FROM Servant, Chef
   WHERE Servant.Rank=Chef.Rank AND Servant.Rank=5
2. SELECT DISTINCT DishName
   FROM Order_From_Cook_Deliver_Dish O1
   WHERE NOT EXISTS
   (SELECT * FROM Customer
   WHERE NOT EXISTS
   (SELECT * FROM Order_From_Cook_Deliver_Dish O2
   WHERE O1.DishName=O2.DishName AND Customer.CustomerID=O2.CustomerID))

3. SELECT * FROM Customer WHERE CustomerName IS Null
4. DELETE FROM Customer WHERE CustomerName IS Null
5. SELECT * FROM Order_From_Cook_Deliver_Dish
   WHERE CustomerID IN
   (SELECT CustomerID
   FROM Customer
   WHERE CustomerName IS NOT Null)
6. DELETE FROM Pay_Bill
   Where CustomerID IN
   (SELECT CustomerID
   FROM Customer
   WHERE CustomerName IS NOT Null)
7. DELETE FROM Order_From_Cook_Deliver_Dish
   Where CustomerID IN
   (SELECT CustomerID
    FROM Customer
   WHERE CustomerName IS NOT Null)
8. UPDATE Manager SET ManagerID='$mid' WHERE ManagerName='$mname'