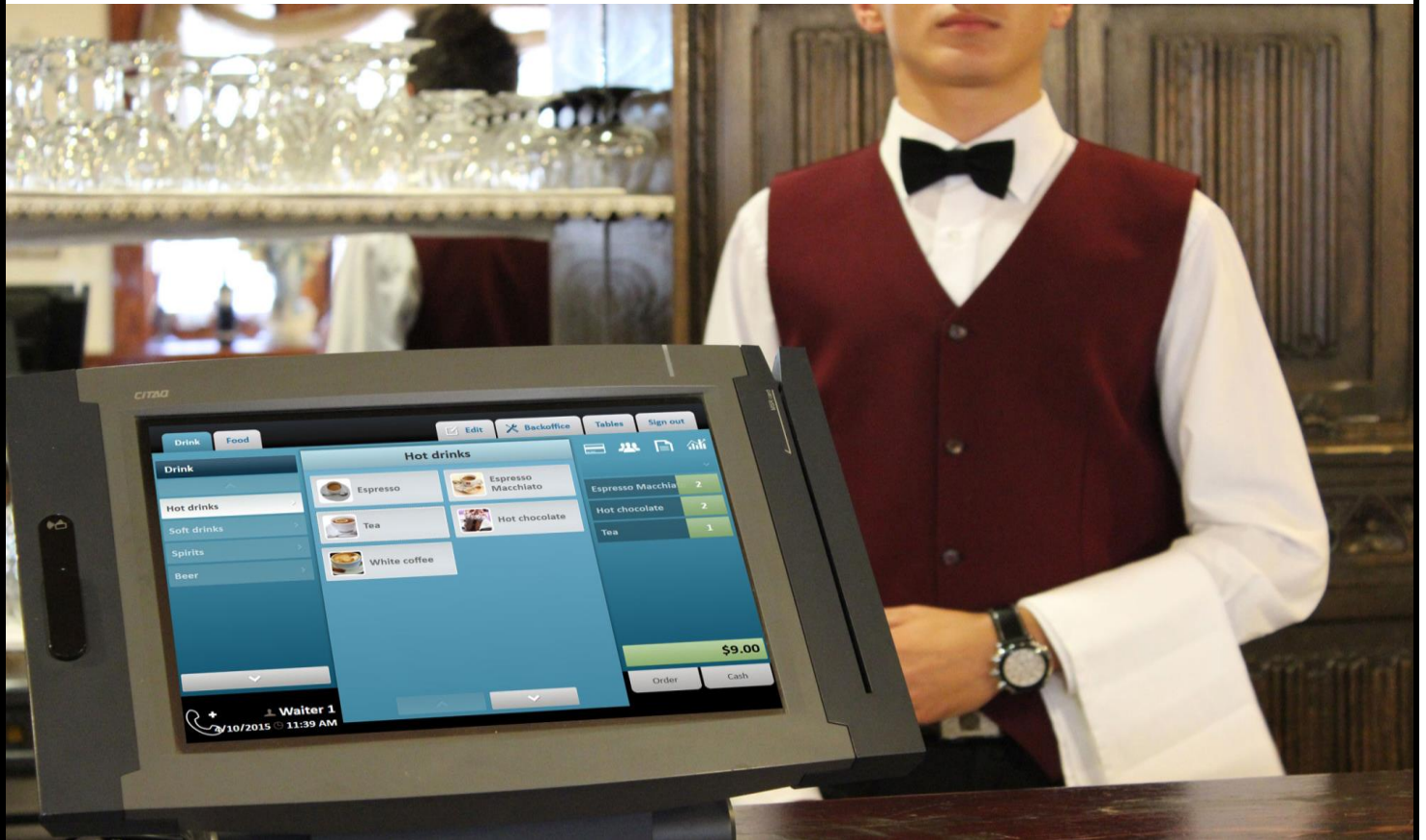


PROJECT REPORT

Restaurant Management System



By-

Jai Kumar Soni

Northeastern University

NUID: 001822913

INFO 6210- Sec 05- Data Management
and Database Design

I have always found it remarkable when I order food at a restaurant and my order through a Tab rather than a piece of paper and how the order is been transferred to the kitchen automatically followed by the timely delivery of the order back to the table. To dig on this more and apply the concepts learned in class, I have chosen to create a database model for the same.

The database requirement of this system involves storing of data right from the customer checking the menu and placing the order to the chef preparing it and the waiter who delivers the order.

Triggers are be used to track the modification of data in tables wherever applicable and stored procedures are used to complete routine tasks like placing orders etc.

This Project shows the approach for:

- Creating the model
- Defining Relationships/Cardinality between entities
- Defining Use Cases
- Creating View's, Stored Procedures, and Triggers to achieve Use Cases and analysis

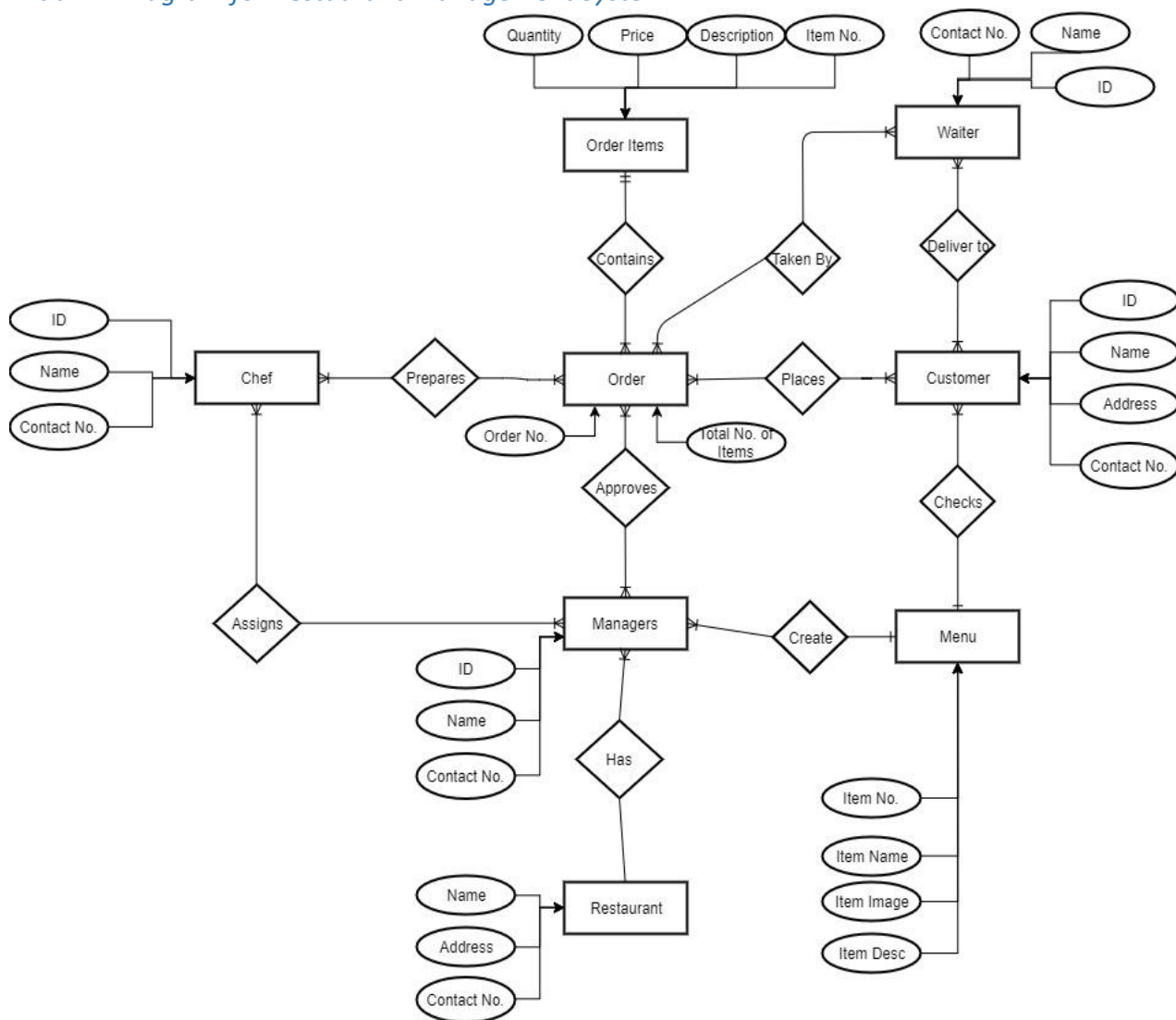
Initial Approach:

Initially designed tables for the model proposed:

1. Order (contains Order no., Order Items, Comments, quantity, Status etc.)
2. Order Items(Item no, Description, quantity, Price)
3. Menu (ID, Item Name, Item Category, item description, item image, cost)
4. Customer (ID, Name, Address, Contact No.)
5. Manager (ID, Name, Contact no.)
6. Restaurant (Name, Address, Contact no.)
7. Chef (ID, Name, Contact no.)
8. Waiter (ID, Name, Contact no.)

Defining Use cases:

1. Customer may place an order
2. Order must contain at least one order Item
3. Analyze Revenue earned per month by the restaurant
4. Analyze Total Revenue earned by restaurant for all the orders
5. Analyze top performing Staffs like Managers, Waiters and Chefs
6. Get the Customer Order History
7. Get Customer's most ordered/ favorite item
8. Get the most ordered items/ famous items of restaurant and revenue earned by each item

Initial ER Diagram for Restaurant Management System

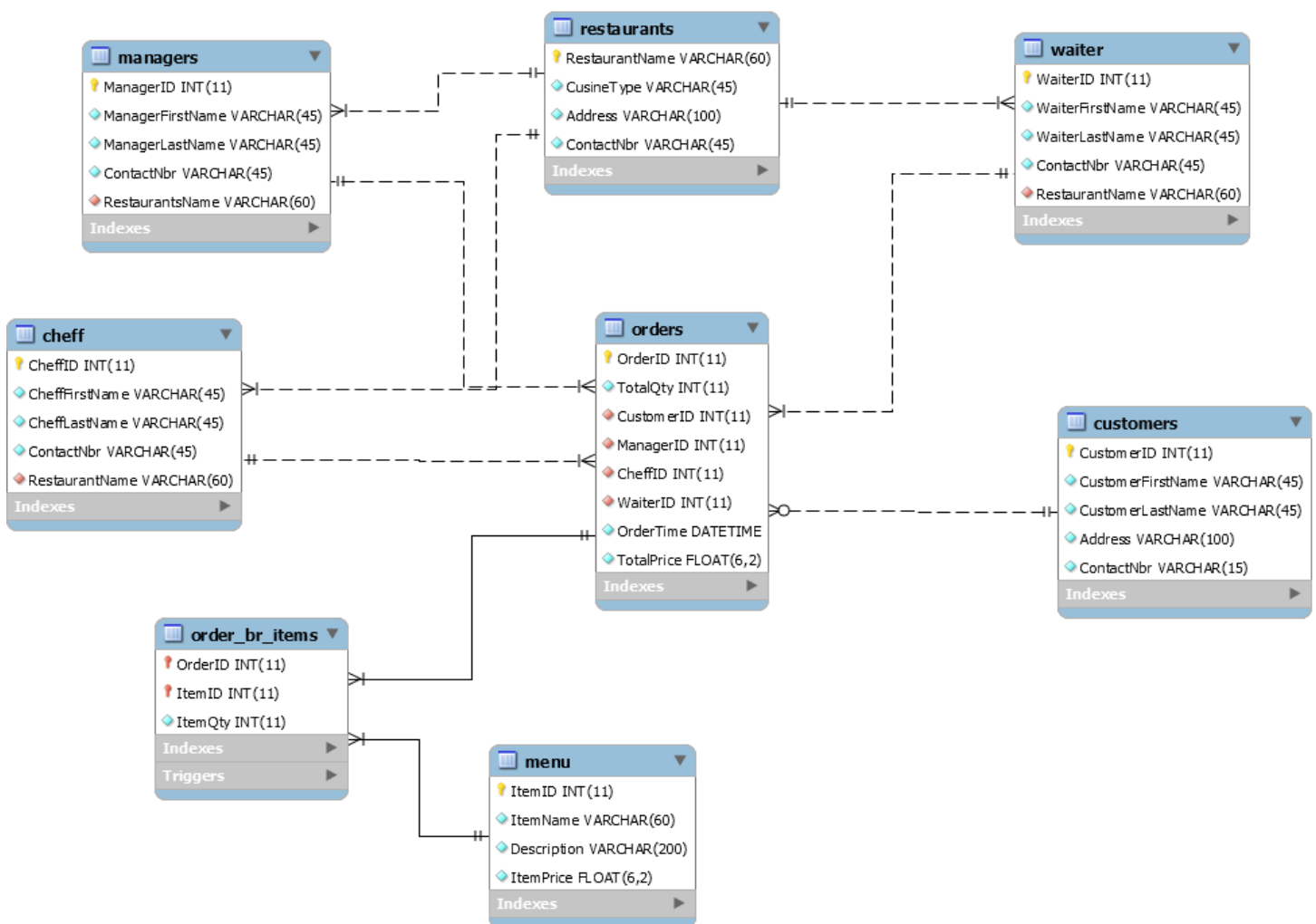
To achieve the proposed use cases and after normalizing the tables, the table model are as follows:

1. Restaurant (Name, Address, Contact no.)
2. Customer (ID, Name, Address, Contact No.)
3. Manager (ID, Name, Contact no.)
4. Chef (ID, Name, Contact no.)
5. Waiter (ID, Name, Contact no.)
6. Menu (Item ID, Item Name, Item Description, Item Price)
7. Order(Order ID, Customer ID, Manager ID, Chef ID, Waiter ID, Total Qty, Total Price, Order Time)
8. Order Items(Order ID, Item ID, Item Qty)

Cardinality of the tables:

- One to Many relationship between Customer and Order- A Customer can create many orders
- One to Many relationship between An Order can have many Order Items
- One to Many relationship between Order Items and Menu, an Item can be present only once in a menu, but there can be many menu items in Order Items table
- One to Many relationship between a manager and a restaurant, a restaurant has at least one manager and a manager cannot work in multiple restaurants
- One to Many relationship between a chef and a restaurant, a restaurant has at least one chef and a chef cannot work in multiple restaurants
- One to Many relationship between a waiter and a restaurant, a restaurant has at least one waiter and a waiter cannot work in multiple restaurants

Final EER diagram after table creation



Use Cases:

1. Customer may place an order

There is a 1: n relationship between CUSTOMER and ORDERS table, a customer may not place any order, or he can order several times.

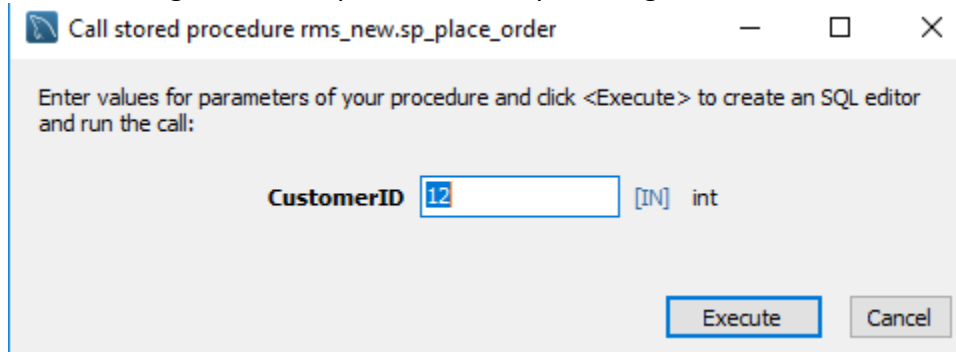
After Checking the menu, manager can place the order on behalf of the person, to place an order, Stored procedure 'sp_Place_Order' is used which requires Customer ID as input parameter, the order is placed in the Orders table, the order is automatically assigned a Manager, Chef and a waiter.

'sp_Place_Order' is defined as follows:

```
CREATE DEFINER='root'@'localhost' PROCEDURE `sp_place_order`(IN CustomerID int)
BEGIN
declare c int default 1;
declare m int default 1;
declare w int default 1;
declare oID int;
set m=(select FLOOR(1 + rand() * (select count(*) from managers)));
set c=(select FLOOR(1 + rand() * (select count(*) from cheff)));
set w=(select FLOOR(1 + rand() * (select count(*) from waiter)));
set oID= (Select Max(OrderID)+1 from orders);

INSERT INTO `rms_new`.`orders` (`OrderID`, `CustomerID`, `ManagerID`, `CheffID`, `WaiterID`) VALUES (oID, CustomerID, m, c, w);
END
```

On calling this stored procedure and providing Customer ID - 12



Call stored procedure rms_new.sp_place_order

Enter values for parameters of your procedure and click <Execute> to create an SQL editor and run the call:

CustomerID [IN] int

Execute Cancel

A record is entered in Orders table as:

	OrderID	TotalQty	CustomerID	ManagerID	CheffID	WaiterID	OrderTime	TotalPrice
	11	15	13	4	5	3	2017-07-08 08:47:35	152.85
	12	3	8	4	1	2	2017-08-22 14:59:37	16.23
	13	7	3	4	2	3	2017-09-04 13:31:14	96.93
	14	10	16	5	2	4	2017-09-15 21:32:40	66.60
	15	16	15	3	2	5	2017-09-22 14:59:37	84.35
	16	5	3	1	5	1	2017-10-11 11:12:05	41.25
	17	1	16	1	4	1	2017-11-22 14:59:37	5.60
	18	4	19	1	5	1	2017-12-03 17:54:05	27.96
	19	22	13	5	5	5	2017-12-11 17:52:19	140.04
	20	5	13	3	4	5	2017-12-11 17:58:08	41.25
	21	7	16	4	1	1	2017-12-13 04:14:52	67.23
	22	8	14	3	4	5	2017-12-13 05:06:58	127.92
	23	0	12	5	3	3	2017-12-13 16:32:41	0.00

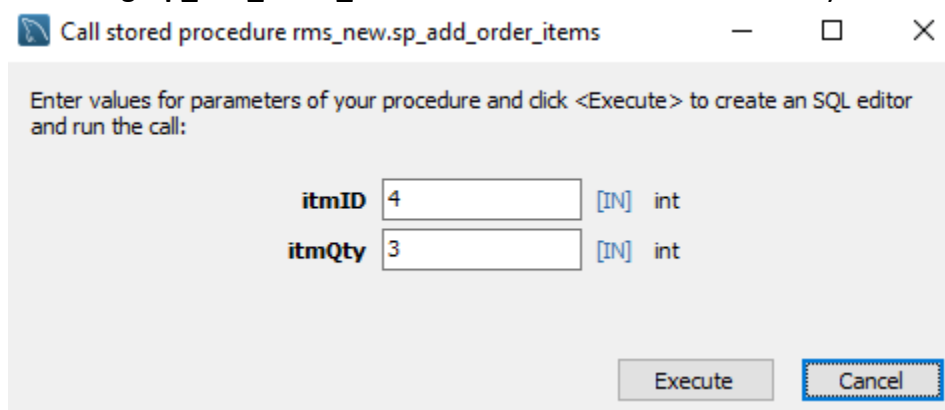
2. Order must contain at least one order Item

After the Values are placed in the Orders Table through 'sp_Place_Order', Items are added in the Order Items table through stored procedure 'sp_add_order_items' which takes Item ID and Item Qty as an input. 'sp_add_order_items' automatically adds the input Item ID and Item Qty to the Order created through 'sp_Place_Order'. Since the order can contain many order items, 'sp_add_order_items' can be called multiple times as per number of items required for that particular order.

'sp_add_order_items' is defined as follows:

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `sp_add_order_items`(IN itmID int,IN itmQty int)
BEGIN
declare oID int ;
set oID=(Select Max(OrderID) from orders);
INSERT INTO `rms_new`.`order_br_items` (`OrderID`, `ItemID`, `ItemQty`) VALUES (oID, itmID, itmQty);
END
```

On calling 'sp_add_order_items' with Item ID as 4 and Item Qty as 3



Call stored procedure rms_new.sp_add_order_items

Enter values for parameters of your procedure and click <Execute> to create an SQL editor and run the call:

itmID [IN] int

itmQty [IN] int

Execute Cancel

Order Items table is updated as:

OrderID	ItemID	ItemQty
16	3	5
17	2	1
18	6	4
19	2	6
19	3	5
19	5	2
19	6	4
19	7	5
20	3	5
21	3	5
21	5	2
22	4	8
23	4	3

After entering values in order items table, the fields Total Qty and Total price are updated in Orders table with the help of triggers. Also if the data in the order items table is modified for a particular

order, like removing an order item from the table, the fields Total Qty and Total price in the Orders table.

The triggers added to achieve the above are as follows:

`tr_OrderQty_Price_AFTER_INSERT`

```
CREATE DEFINER=`root`@`localhost` TRIGGER `rms_new`.`tr_OrderQty_Price_AFTER_INSERT`
AFTER INSERT ON `order_br_items` FOR EACH ROW
] BEGIN
  update orders
  set TotalQty= (select sum(ItemQty)from order_br_items where OrderID=new.OrderID)
  where OrderID=new.OrderID;
  call sp_TotalPrice(new.OrderID);
- END
```

`tr_OrderQty_Price_AFTER_UPDATE`:

```
CREATE DEFINER=`root`@`localhost` TRIGGER `rms_new`.`tr_OrderQty_Price_AFTER_UPDATE`
AFTER UPDATE ON `order_br_items` FOR EACH ROW
] BEGIN
  update orders
  set TotalQty= (select sum(ItemQty)from order_br_items where OrderID=new.OrderID)
  where OrderID=new.OrderID;
  call sp_TotalPrice(new.OrderID);
- END
```

`tr_OrderQty_Price_AFTER_DELETE`:

```
CREATE DEFINER=`root`@`localhost` TRIGGER `rms_new`.`tr_OrderQty_Price_AFTER_DELETE`
AFTER DELETE ON `order_br_items` FOR EACH ROW
] BEGIN
  update orders
  set TotalQty= (select sum(ItemQty)from order_br_items where OrderID=old.OrderID)
  where OrderID=old.OrderID;
- call sp_TotalPrice(old.OrderID);
  END
```

The above triggers uses a stored procedure 'sp_TotalPrice' to update the Total Price of the order in the orders table

`sp_TotalPrice`:

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `sp_TotalPrice`(IN OrderID int)
] BEGIN
  update orders
  set TotalPrice= (select OrderPrice from vw_order_price where vw_order_price.OrderID=OrderID)
  where orders.OrderID=OrderID;
- END
```

'sp_TotalPrice' uses a view 'vw_order_price' to update the Total Price in Orders table

`vw_order_price` uses joins to get the Item price in the menu table and sums the items based on Item ID and Item Qty from the Order Items table, so the final output of the view is Order ID and Order Price

`vw_order_price`:

```
CREATE
  ALGORITHM = UNDEFINED
  DEFINER = `root`@`localhost`
  SQL SECURITY DEFINER
VIEW `vw_order_price` AS
  SELECT
    `order_br_items`.`OrderID` AS `OrderID`,
    SUM((`order_br_items`.`ItemQty` * `menu`.`ItemPrice`)) AS `OrderPrice`
  FROM
    (`order_br_items`
    JOIN `menu` ON ((`order_br_items`.`ItemID` = `menu`.`ItemID`)))
  GROUP BY `order_br_items`.`OrderID`
```

Which contains values like:

OrderID	OrderPrice
1	28.05
2	13.98
3	23.49
4	28.58
5	12.35
6	23.22
7	105.93
8	88.70
9	20.90
10	104.70
11	152.85
12	16.23

Once the Item ID and item Qty is added in Order Items table, Order table is updated as follows:

OrderID	TotalQty	CustomerID	ManagerID	CheffID	WaiterID	OrderTime	TotalPrice
11	15	13	4	5	3	2017-07-08 08:47:35	152.85
12	3	8	4	1	2	2017-08-22 14:59:37	16.23
13	7	3	4	2	3	2017-09-04 13:31:14	96.93
14	10	16	5	2	4	2017-09-15 21:32:40	66.60
15	16	15	3	2	5	2017-09-22 14:59:37	84.35
16	5	3	1	5	1	2017-10-11 11:12:05	41.25
17	1	16	1	4	1	2017-11-22 14:59:37	5.60
18	4	19	1	5	1	2017-12-03 17:54:05	27.96
19	22	13	5	5	5	2017-12-11 17:52:19	140.04
20	5	13	3	4	5	2017-12-11 17:58:08	41.25
21	7	16	4	1	1	2017-12-13 04:14:52	67.23
22	8	14	3	4	5	2017-12-13 05:06:58	127.92
23	3	12	5	3	3	2017-12-13 16:32:41	47.97

3. Analyze Revenue earned per month by the restaurant

A view `vw_restaurantrevenue_analysis` is created to achieve the monthly revenue of the restaurant. `vw_restaurantrevenue_analysis` view is defined as follows:

```
CREATE
  ALGORITHM = UNDEFINED
  DEFINER = `root`@`localhost`
  SQL SECURITY DEFINER
VIEW `vw_restaurantrevenue_analysis` AS
  SELECT
    MONTH(`orders`.`OrderTime`) AS `Month`,
    YEAR(`orders`.`OrderTime`) AS `Year`,
    COUNT(`orders`.`OrderID`) AS `OrdersPerMonth`,
    SUM(`orders`.`TotalPrice`) AS `RevenuePerMonth`
  FROM
    `orders`
  GROUP BY `Month` , `Year`
  ORDER BY `Year` , `Month`|
```

The values are displayed as follows:

Month	Year	OrdersPerMonth	RevenuePerMonth
1	2017	2	42.03
2	2017	1	23.49
3	2017	2	40.93
4	2017	1	23.22
5	2017	1	105.93
6	2017	3	214.30
7	2017	1	152.85
8	2017	1	16.23
9	2017	3	247.88
10	2017	1	41.25
11	2017	1	5.60
12	2017	5	404.40

4. Analyze Total Revenue earned by restaurant for all the orders

A view `vw_total_revenue` is created to calculate the total revenue of the restaurant based on the orders taken.

`vw_total_revenue`:

```
CREATE
  ALGORITHM = UNDEFINED
  DEFINER = `root`@`localhost`
  SQL SECURITY DEFINER
VIEW `vw_total_revenue` AS
  SELECT
    SUM(`orders`.`TotalPrice`) AS `TotalRevenue`
  FROM
    `orders`
```

The value is displayed as:

TotalRevenue
1318.11

5. Analyze top performing Staffs like Managers, Waiters and Chefs

Views are created to measure the performance of Managers, Chefs and Waiters based on the number of orders they served

`vw_cheff_analysis`:

```
CREATE
  ALGORITHM = UNDEFINED
  DEFINER = `root`@`localhost`
  SQL SECURITY DEFINER
VIEW `vw_cheff_analysis` AS
  SELECT
    `orders`.`CheffID` AS `CheffID`,
    CONCAT_WS(' ',
      `cheff`.`CheffFirstName`,
      `cheff`.`CheffLastName`) AS `CheffName`,
    COUNT(`orders`.`CheffID`) AS `NbrOfOrdersPrepared`
  FROM
    (`orders`
    LEFT JOIN `cheff` ON ((`orders`.`CheffID` = `cheff`.`CheffID`)))
  GROUP BY `orders`.`CheffID`
  ORDER BY `NbrOfOrdersPrepared` DESC
```

Top chefs are displayed as:

CheffID	CheffName	NbrOfOrdersPrepared
5	Mattie Huffev	7
2	Sherilvn Kibel	6
1	Pearl Vasser	4
4	Antoni Woolmore	4
3	Ginnv MacFadden	1

`vw_manager_analysis`:

```
CREATE
  ALGORITHM = UNDEFINED
  DEFINER = `root`@`localhost`
  SQL SECURITY DEFINER
VIEW `vw_manager_analysis` AS
  SELECT
    `orders`.`ManagerID` AS `ManagerID`,
    CONCAT_WS(' ',
      `managers`.`ManagerFirstName`,
      `managers`.`ManagerLastName`) AS `ManagerName`,
    COUNT(`orders`.`ManagerID`) AS `NbrOfOrdersTaken`
  FROM
    (`orders`
    LEFT JOIN `managers` ON ((`orders`.`ManagerID` = `managers`.`ManagerID`)))
  GROUP BY `orders`.`ManagerID`
  ORDER BY `NbrOfOrdersTaken` DESC
```

Top Managers are displayed as:

ManagerID	ManagerName	NbrOfOrdersTaken
4	Verena Neroer	7
1	Herta Treend	6
3	Saraann Lupton	5
5	Charlotta Menarv	3
2	Marvs Cullinaford	1

`vw_waiter_analysis`:

```
CREATE
  ALGORITHM = UNDEFINED
  DEFINER = `root`@`localhost`
  SQL SECURITY DEFINER
VIEW `vw_waiter_analysis` AS
  SELECT
    `orders`.`WaiterID` AS `WaiterID`,
    CONCAT_WS(' ',
      `waiter`.`WaiterFirstName`,
      `waiter`.`WaiterLastName`) AS `WaiterName`,
    COUNT(`orders`.`WaiterID`) AS `NbrOfOrdersDelivered`
  FROM
    (`orders`
    LEFT JOIN `waiter` ON ((`orders`.`WaiterID` = `waiter`.`WaiterID`)))
  GROUP BY `orders`.`WaiterID`
  ORDER BY `NbrOfOrdersDelivered` DESC
```

Top Waiters are displayed as:

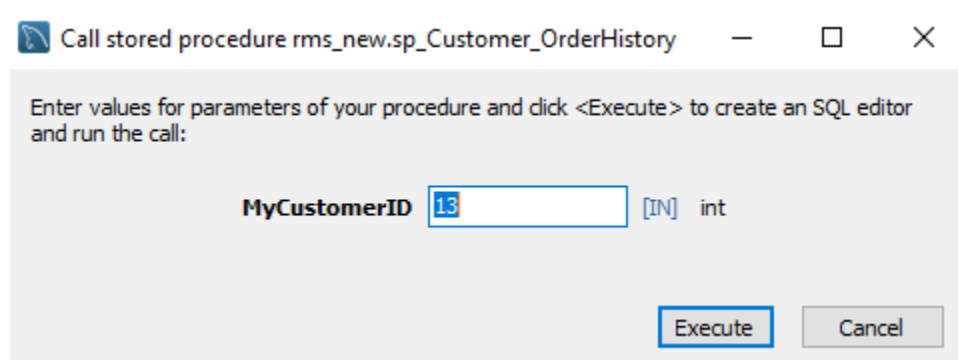
WaiterID	WaiterName	NbrOfOrdersDelivered
1	Philipa Wrack	9
5	Claudina Tancock	4
2	Irwin Roscamp	3
3	Ford Edvson	3
4	Donall Gravlev	3

6. Get the Customer Order History

A stored procedure `sp_Customer_OrderHistory` with an input parameter of Customer ID is created to display the Customer Order History

`sp_Customer_OrderHistory`:

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `sp_Customer_OrderHistory` (IN MyCustomerID int)
] BEGIN
select orders.OrderID, order_br_items.ItemID,
menu.ItemName,order_br_items.ItemQty,
menu.ItemPrice*order_br_items.ItemQty as TotalItemPrice,
orders.TotalPrice, orders.OrderTime
from orders inner join customers on
orders.CustomerID=Customers.customerID
left join order_br_items on
order_br_items.OrderID=orders.OrderID
left join menu on
menu.ItemID=order_br_items.ItemID
where orders.CustomerID=MyCustomerID;
END
```



Call stored procedure rms_new.sp_Customer_OrderHistory

Enter values for parameters of your procedure and click <Execute> to create an SQL editor and run the call:

MyCustomerID [IN] int

The execution of the above code gives the following result:

OrderID	ItemID	ItemName	ItemQty	TotalItemPrice	TotalPrice	OrderTime
10	3	Daddv's Double Burger	3	24.75	104.70	2017-06-30 02:31:08
10	4	Chicken Parmesan Calzone	5	79.95	104.70	2017-06-30 02:31:08
11	5	Buffalo Chicken Calzone	8	103.92	152.85	2017-07-08 08:47:35
11	6	Buffalo Chicken Sub	7	48.93	152.85	2017-07-08 08:47:35
19	2	American Burger	6	33.60	140.04	2017-12-11 17:52:19
19	3	Daddv's Double Burger	5	41.25	140.04	2017-12-11 17:52:19
19	5	Buffalo Chicken Calzone	2	25.98	140.04	2017-12-11 17:52:19
19	6	Buffalo Chicken Sub	4	27.96	140.04	2017-12-11 17:52:19
19	7	UHOP Soda	5	11.25	140.04	2017-12-11 17:52:19
20	3	Daddv's Double Burger	5	41.25	41.25	2017-12-11 17:58:08

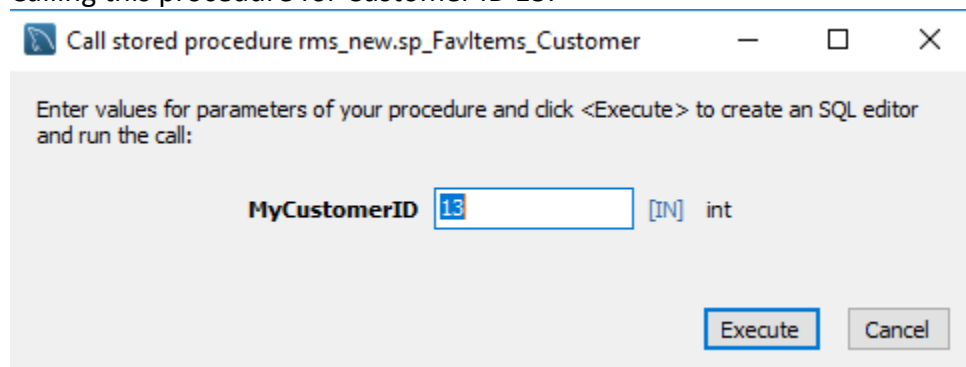
7. Get Customer's most ordered/ favorite item

A Stored procedure is `sp_FavItems_Customer` with the input parameter of Customer ID is created for to get the customers most favored items based on analysis on his order history

`sp_FavItems_Customer`:

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `sp_FavItems_Customer`(IN MyCustomerID int)
BEGIN
SELECT vw_preq_customer_analysis.ItemID,menu.ItemName as ItemName,
count(vw_preq_customer_analysis.ItemID) as NbrOfTimesOrderd
FROM rms_new.vw_preq_customer_analysis inner join
menu on vw_preq_customer_analysis.ItemID=menu.ItemID
where customerID=MyCustomerID
group by customerID, vw_preq_customer_analysis.ItemID
order by NbrOfTimesOrderd desc;
END
```

Calling this procedure for Customer ID 13:



Produces the following result:

ItemID	ItemName	NbrOfTimesOrderd
3	Daddv's Double Burger	3
5	Buffalo Chicken Calzone	2
6	Buffalo Chicken Sub	2
4	Chicken Parmesan Calzone	1
7	UHOP Soda	1
2	American Burger	1

8. Get the most ordered items/ famous items of restaurant and revenue earned by each item

A View `vw_item_analysis` is created that analyzes previous orders and gets the items present in most number of orders

`vw_item_analysis`:

```
CREATE
  ALGORITHM = UNDEFINED
  DEFINER = `root`@`localhost`
  SQL SECURITY DEFINER
VIEW `vw_item_analysis` AS
  SELECT
    `order_br_items`.`ItemID` AS `ItemID`,
    `menu`.`ItemName` AS `ItemName`,
    COUNT(`order_br_items`.`ItemID`) AS `NbrOfTimesOrdered`,
    SUM(`order_br_items`.`ItemQty`) AS `TotalQtySold`,
    (SUM(`order_br_items`.`ItemQty`) * `menu`.`ItemPrice`) AS `RevenueEarnedByItem`
  FROM
    (`order_br_items`
  LEFT JOIN `menu` ON ((`order_br_items`.`ItemID` = `menu`.`ItemID`)))
  GROUP BY `order_br_items`.`ItemID`
  ORDER BY `NbrOfTimesOrdered` DESC
```

Upon execution the data is displayed as:

ItemID	ItemName	NbrOfTimesOrdered	TotalQtySold	RevenueEarnedByItem
2	American Burder	9	36	201.60
3	Daddv's Double Burder	8	35	288.75
6	Buffalo Chicken Sub	8	24	167.76
7	UHOP Soda	5	15	33.75
5	Buffalo Chicken Calzone	5	19	246.81
4	Chicken Parmesan Calzone	5	21	335.79
1	Onion Rins	2	9	43.65