University of Regina

DEPARTMENT OF COMPUTER SCIENCE

CS 875 –Database Systems

Project Report

Supervised by

Dr. Cory Butz

Submitted by

Kiranmai Arva

200416487

KAD373@uregina.ca

Contents

1. Introduction

Most of the restaurants use paper-based approach to communicate between the customer orders and chefs in the Kitchen via waiter. While following this type of approach there are high chances of miscommunication between the customer and chef and this may lead to the customer dissatisfaction. Apart from the above problem, the other problem is at the kitchen level chefs may get confused between the orders if there are huge number of orders. The information like number of orders received, number of orders processed by chef and which chef, Customer order information, customer order history, information is not maintained when paper-based approach is followed.

1.1 Objective

The main objective of this problem is to eliminate the paper-based communication. A tablet is placed on every table with restaurant application. Customers can login to the application view the menu and place orders also write some notes to chef while placing orders. All the active orders are sent to the chef. The chefs can select the order and start processing it. Once the order is done, he can update the order status as done. Now the waiter can serve the food to the customers. The admin or store manager is responsible for updating the daily menu, check the table status like if the table is free or occupied, can add new employee details, can check the number of orders processed by each chef in a day. All the tasks mentioned above can be achieved by building and maintaining a Relational Database. The main objective is to build a relational database which stores all the information.

1.2 Benefits of Relational Database for Restaurants

The following are benefits of building a relational database for restaurant management

Information storage: Each and every piece of information is stored logically in a relational database. All the information about customers, orders, payment, chef info, feedback from customers are all stored logically in a database. By storing the information logically one can analyse the stored information and can identify the patterns like customer behaviour, most liked food items and customer opinion on restaurants.

Easy access to data

As the information is stored in a database, the information can be accessed very easily just by one click. This way there will be no delayed or wrong communication between the chef and customers. Also, the restaurant manager can check the total number of orders processed in a day, Update the menu items and add new chefs to the system just by one click.

Data Integrity

Data Integrity is one crucial factor to be considered while building a Relational database. By maintaining validity checks and referential integrity constraints can prevent the orphaned records or incomplete records from the database.

Improved customer experience

By ruling out the paper-based communication, we can effectively improve the customer experiences by reducing the miscommunications and by providing the live status of the order.

2. Requirement Analysis

To build a database for restaurant management system, the actors should be identified first. In this scenario there are three main actors namely Customers, Chef, Manager. The developed database is used by these three users via front end application. Now functionalities of each actor have to be defined.

2.1 Functionalities

Customers

- Can login to the system
- Check the menu items and place an order with some notes to the chef if required
- Provide feedback, Make payment

- Can check previous order history, if returning customer

Chef

- Check the Active orders and process them
- Update the order status like processing, completed.
- Can check all the orders

Administrator/Manager

- Add new chefs to the system
- Update the menu items every day
- Check the number of tables occupied and empty

2.2 Software Requirements

Database Engine: SQL Server

GUI Tool: SQL Server Management Studio, Visual Studio

Programming Language: C#

Project type: Individual

3. Entity-Relationship Diagram

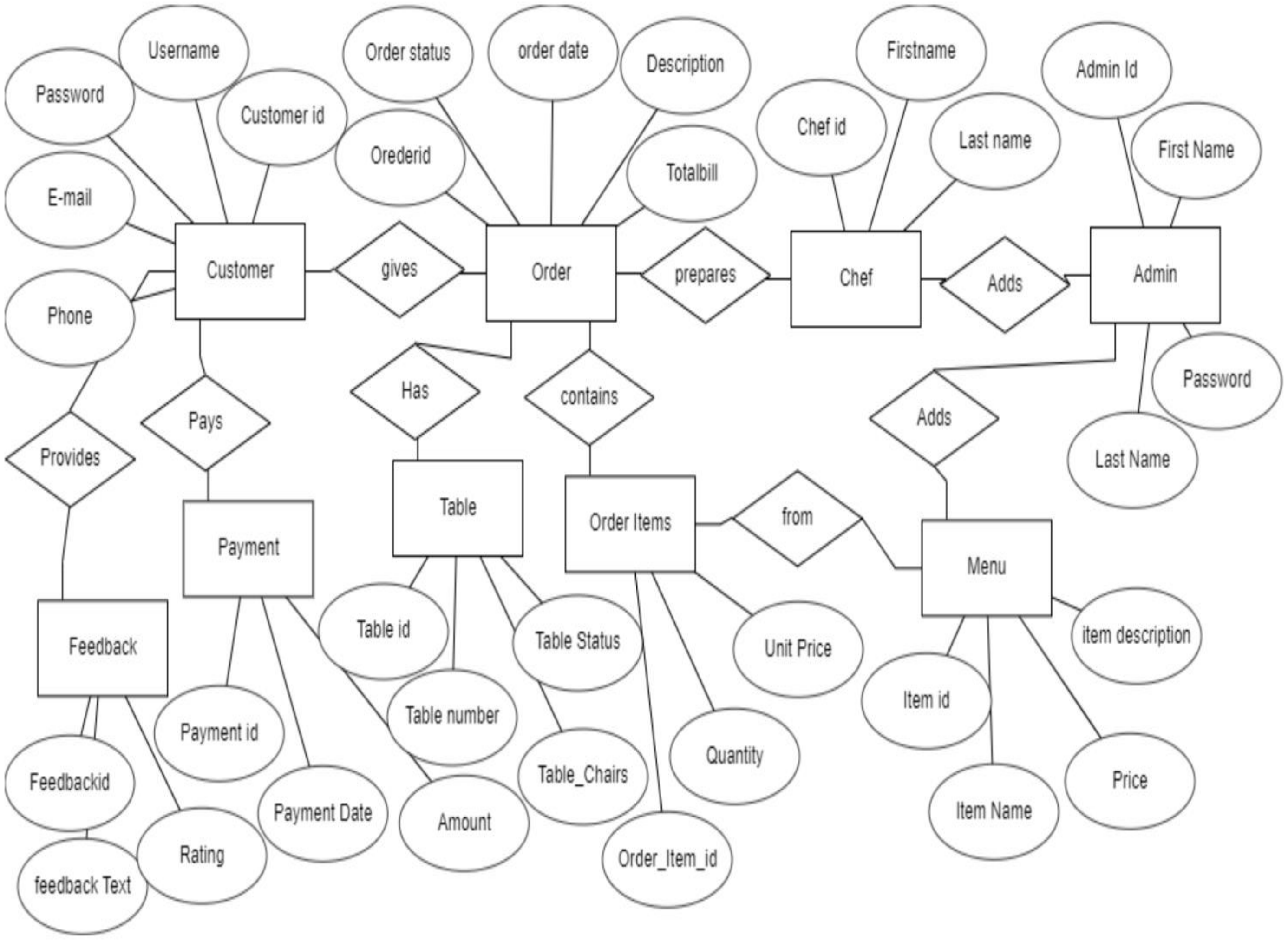


Fig 3:ERD Diagram

4. Data Dictionary

Data Dictionary contains the metadata about the database. It holds information about the database access permission, security which are handled by the administrator, Table level information like columns datatypes. Table constraints like Primary key constraint, Foreign key constraints.

Constraints used while creating the table in this project are

Primary Key Constraint: Primary key column, uniquely identifies each tuple in a table. The primary key columns can act as foreign key in other tables. Null Values are not accepted for primary key column.

Foreign Key Constraint: In relational database the relations between the tables are maintained by foreign key constraint. This key helps to prevent the invalid data inserted into the foreign key column. If a value needs to be inserted in the foreign key column that values needs to be present in the parent table.

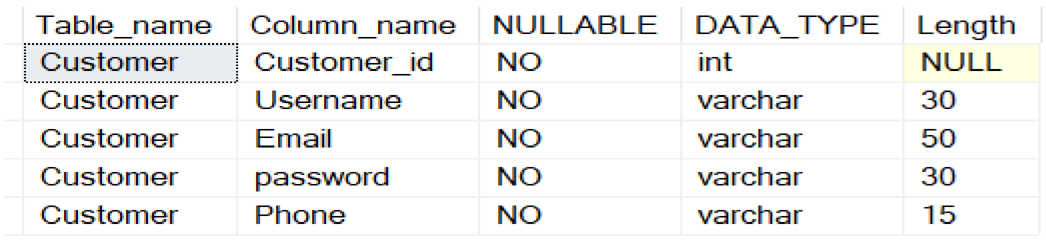
Unique Key Constraint: Unique key constraints also identifies each tuple in a table as unique. Many unique keys can be present in a table and they can accept the null values.

In this project the defined schema name is ‘rms’ and the Database Name is ‘Restaurant_Management’. In Restaurant_Management database following tables are defined.

4.1 Customer

SQL Query:

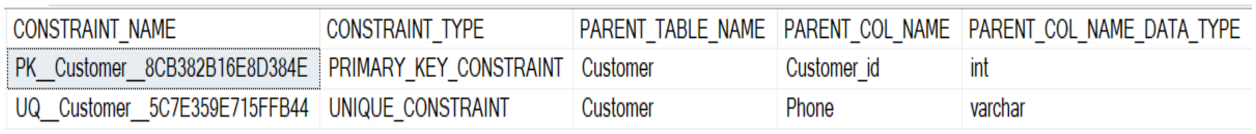
```
CREATE TABLE rms.Customer
( Customer_id int IDENTITY(1,1) NOT NULL PRIMARY KEY,
  Username VARCHAR(30) NOT NULL,
  Email VARCHAR(50) NOT NULL,
  password VARCHAR(30) NOT NULL,
  Phone VARCHAR(15) unique NOT NULL);
```



Table_name	Column_name	NULLABLE	DATA_TYPE	Length
Customer	Customer_id	NO	int	NULL
Customer	Username	NO	varchar	30
Customer	Email	NO	varchar	50
Customer	password	NO	varchar	30
Customer	Phone	NO	varchar	15

Table 4.1: Customer table information

Constraints on Customer Table



CONSTRAINT_NAME	CONSTRAINT_TYPE	PARENT_TABLE_NAME	PARENT_COL_NAME	PARENT_COL_NAME_DATA_TYPE
PK__Customer__8CB382B16E8D384E	PRIMARY_KEY_CONSTRAINT	Customer	Customer_id	int
UQ__Customer__5C7E359E715FFB44	UNIQUE_CONSTRAINT	Customer	Phone	varchar

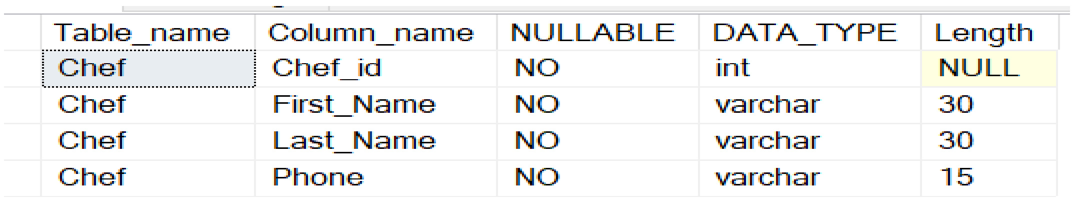
Table 4.2: Customer table constraints

In the Customer table, the primary key is on 'Customer_id' Moreover, Customer_id is Identity key and Unique key constraint is on 'Phone'. If the table definition is observed the primary key column is Identity. In identity column the values increase automatically and user cannot insert values into the identity column.

4.2 Chef

SQL Query:

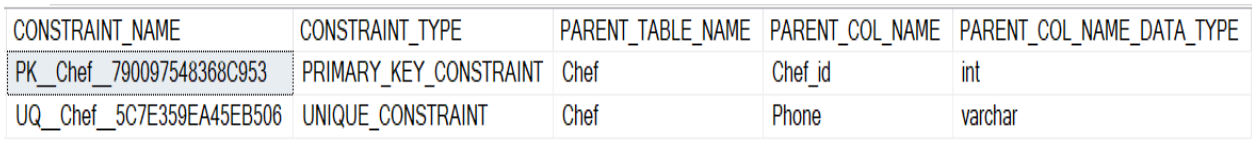
```
CREATE TABLE rms.Chef
( Chef_id int IDENTITY(1,1) NOT NULL PRIMARY KEY,
  First_Name VARCHAR(30) NOT NULL,
  Last_Name VARCHAR(30) NOT NULL,
  Phone VARCHAR(15) unique NOT NULL
);
```



Table_name	Column_name	NULLABLE	DATA_TYPE	Length
Chef	Chef_id	NO	int	NULL
Chef	First_Name	NO	varchar	30
Chef	Last_Name	NO	varchar	30
Chef	Phone	NO	varchar	15

Table 4.3: Chef Table Information

Constraints on Chef Table



CONSTRAINT_NAME	CONSTRAINT_TYPE	PARENT_TABLE_NAME	PARENT_COL_NAME	PARENT_COL_NAME_DATA_TYPE
PK__Chef__790097548368C953	PRIMARY_KEY_CONSTRAINT	Chef	Chef_id	int
UQ__Chef__5C7E359EA45EB506	UNIQUE_CONSTRAINT	Chef	Phone	varchar

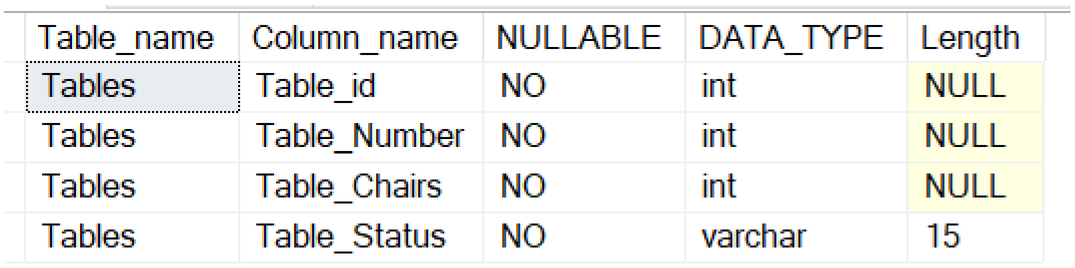
Table 4.4: Chef table constraints

In a Chef table the Primary key is ‘Chef_id’ and Unique key is on ‘Phone’

4.3 Tables

SQL Query:

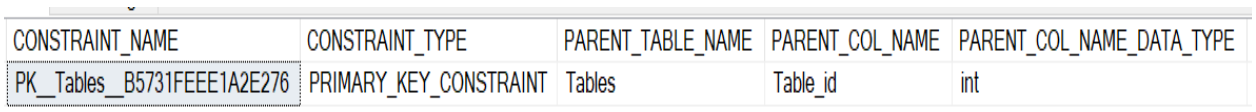
```
CREATE TABLE rms.Tables
( Table_id int IDENTITY(1,1) NOT NULL PRIMARY KEY,
  Table_Number int NOT NULL,
  Table_Chairs int NOT NULL,
  Table_Status VARCHAR(15)  NOT NULL
);
```



Table_name	Column_name	NULLABLE	DATA_TYPE	Length
Tables	Table_id	NO	int	NULL
Tables	Table_Number	NO	int	NULL
Tables	Table_Chairs	NO	int	NULL
Tables	Table_Status	NO	varchar	15

Table 4.5: Tables table Information

Constraints on Tables Table



CONSTRAINT_NAME	CONSTRAINT_TYPE	PARENT_TABLE_NAME	PARENT_COL_NAME	PARENT_COL_NAME_DATA_TYPE
PK__Tables__B5731FEEE1A2E276	PRIMARY_KEY_CONSTRAINT	Tables	Table_id	int

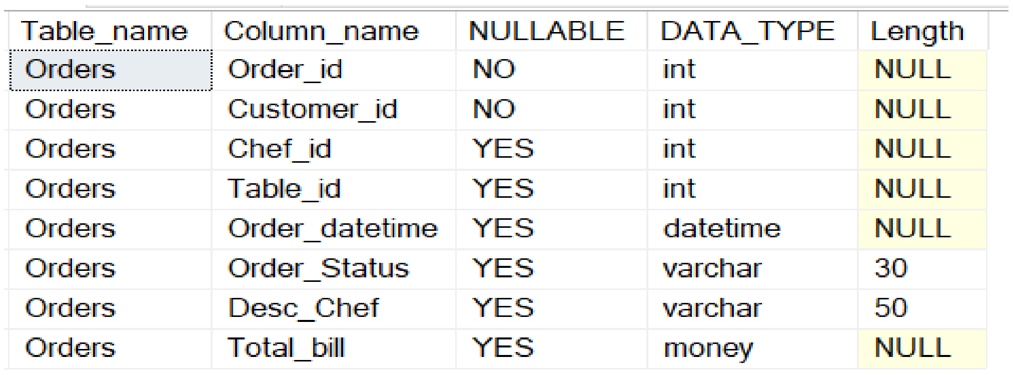
Table 4.6: Tables table constraints

In Tables table has only the primary key ‘Table_id’

4.4 Orders

SQL Query:

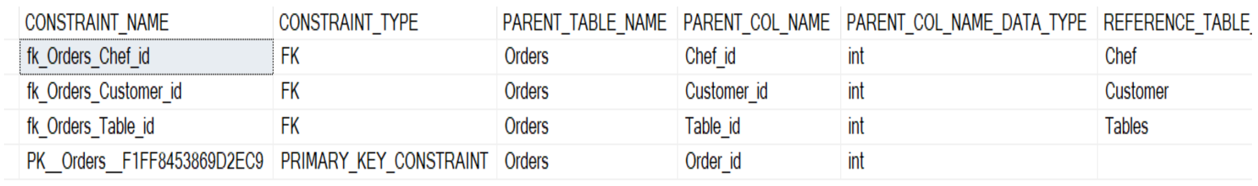
```
CREATE TABLE rms.Orders
( Order_id int IDENTITY(1,1) NOT NULL PRIMARY KEY,
Customer_id int NOT NULL ,
Chef_id int ,
Table_id int  ,
Order_datetime datetime,
Order_Status VARCHAR(30) NULL,
Desc_Chef VARCHAR(50) NULL,
Total_bill Money NULL,
CONSTRAINT fk_Orders_Customer_id FOREIGN KEY (Customer_id) REFERENCES
rms.Customer(Customer_id) ON UPDATE CASCADE ON DELETE NO ACTION,
CONSTRAINT fk_Orders_Chef_id FOREIGN KEY (Chef_id) REFERENCES
rms.Chef(Chef_id) ON UPDATE CASCADE ON DELETE SET NULL,
CONSTRAINT fk_Orders_Table_id FOREIGN KEY (Table_id) REFERENCES
rms.Tables(Table_id) ON UPDATE CASCADE ON DELETE SET NULL
);
```



Table_name	Column_name	NULLABLE	DATA_TYPE	Length
Orders	Order_id	NO	int	NULL
Orders	Customer_id	NO	int	NULL
Orders	Chef_id	YES	int	NULL
Orders	Table_id	YES	int	NULL
Orders	Order_datetime	YES	datetime	NULL
Orders	Order_Status	YES	varchar	30
Orders	Desc_Chef	YES	varchar	50
Orders	Total_bill	YES	money	NULL

Table 4.7: Orders table Information

Constraints on Orders Table



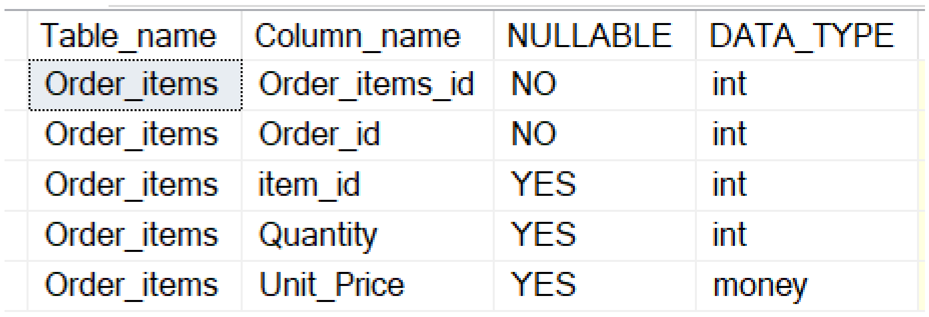
CONSTRAINT_NAME	CONSTRAINT_TYPE	PARENT_TABLE_NAME	PARENT_COL_NAME	PARENT_COL_NAME_DATA_TYPE	REFERENCE_TABLE_
fk_Orders_Chef_id	FK	Orders	Chef_id	int	Chef
fk_Orders_Customer_id	FK	Orders	Customer_id	int	Customer
fk_Orders_Table_id	FK	Orders	Table_id	int	Tables
PK__Orders__F1FF8453869D2EC9	PRIMARY_KEY_CONSTRAINT	Orders	Order_id	int	

Table 4.8: Orders table constraints

4.5 Order_Items

SQL Query:

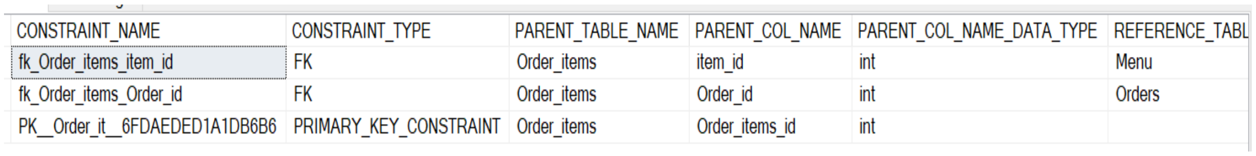
```
CREATE TABLE rms.Order_items
( Order_items_id int IDENTITY(1,1) NOT NULL PRIMARY KEY,
Order_id int NOT NULL ,
item_id int  ,
Quantity  int ,
Unit_Price Money NULL,
CONSTRAINT fk_Order_items_Order_id FOREIGN KEY (Order_id) REFERENCES
rms.Orders(Order_id) ON UPDATE CASCADE ON DELETE NO ACTION,
CONSTRAINT fk_Order_items_item_id FOREIGN KEY (Item_id) REFERENCES
rms.Menu(Item_id) ON UPDATE CASCADE ON DELETE SET NULL,
);
```



Table_name	Column_name	NULLABLE	DATA_TYPE
Order_items	Order_items_id	NO	int
Order_items	Order_id	NO	int
Order_items	item_id	YES	int
Order_items	Quantity	YES	int
Order_items	Unit_Price	YES	money

Table 4.9: Order_items table information

Constraints on Order_items Table



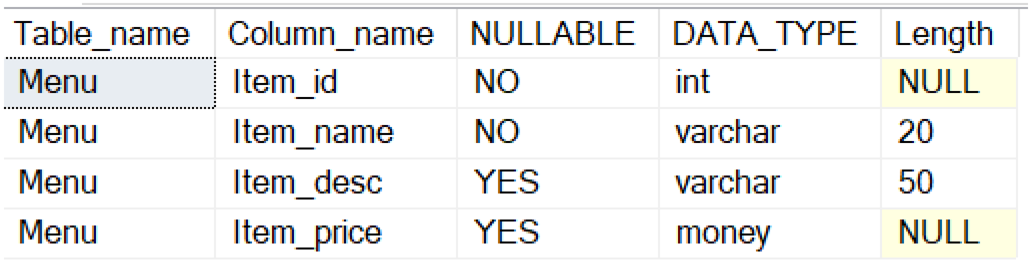
CONSTRAINT_NAME	CONSTRAINT_TYPE	PARENT_TABLE_NAME	PARENT_COL_NAME	PARENT_COL_NAME_DATA_TYPE	REFERENCE_TABL
fk_Order_items_item_id	FK	Order_items	item_id	int	Menu
fk_Order_items_Order_id	FK	Order_items	Order_id	int	Orders
PK__Order_it__6FDAEDED1A1DB6B6	PRIMARY_KEY_CONSTRAINT	Order_items	Order_items_id	int	

Table 4.10: Order_items table constraints

4.6 Menu

SQL Query:

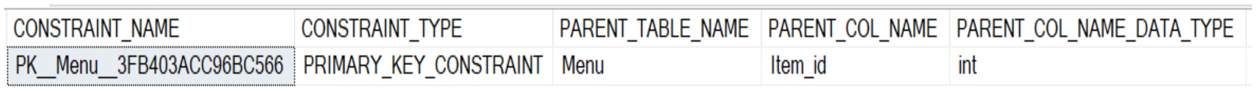
```
CREATE TABLE rms.Menu
( Item_id int IDENTITY(1,1) NOT NULL PRIMARY KEY,
Item_name VARCHAR(20) NOT NULL,
Item_desc VARCHAR(50) NULL,
Item_price Money NULL,
);
```



Table_name	Column_name	NULLABLE	DATA_TYPE	Length
Menu	Item_id	NO	int	NULL
Menu	Item_name	NO	varchar	20
Menu	Item_desc	YES	varchar	50
Menu	Item_price	YES	money	NULL

Table 4.11: Menu table information

Constraints on Menu Table



CONSTRAINT_NAME	CONSTRAINT_TYPE	PARENT_TABLE_NAME	PARENT_COL_NAME	PARENT_COL_NAME_DATA_TYPE
PK__Menu__3FB403ACC96BC566	PRIMARY_KEY_CONSTRAINT	Menu	Item_id	int

Table 4.12: Menu table constraints

4.7 Feedback

SQL Query

```
CREATE TABLE rms.Feedback
```

```
( Feedback_id int IDENTITY(1,1) NOT NULL PRIMARY KEY,
Customer_id int NOT NULL ,
Feedback_text nvarchar(50),
Feedback_Rating int,
CONSTRAINT fk_Feedback_Customer_id FOREIGN KEY (Customer_id) REFERENCES
rms.Customer(Customer_id) ON UPDATE CASCADE ON DELETE NO ACTION
);
```

4.8 Payment

SQL Query:

```
CREATE TABLE rms.Payment
( Payment_id int IDENTITY(1,1) NOT NULL PRIMARY KEY,
Order_id int NOT NULL ,
Payment_Date Datetime  ,
Total_bill Money NULL,
CONSTRAINT fk_Payment_ Order _id FOREIGN KEY (Order_id) REFERENCES
rms.Orders(Customer_id) ON UPDATE CASCADE ON DELETE NO ACTION

);
```

5. Integrity Rules

The database designed above enforces certain Integrity Rules. By following these integrity constraints, the data stored in the database is valid and consistent. Integrity constraints are applied to ensure the changes like Updation, Insertion, Deletion does not lead to the loss of data. There are two important integrity rules they are Entity Integrity Rule and Referential Integrity Rule.

***Entity Integrity Rule*:** No Component of primary key on the base relation is allowed to be null.

***Referential Integrity Rule*:** The database should not contain any unmatched foreign key values.

The designed database for Restaurant Management system follows the Entity Integrity Rule constraints. If we notice the table definitions of all the table discussed above, the primary key column for all the tables have ‘NOT NULL’ enforced. The primary key column cannot be null, from this we can conclude that the designed database follows Entity Integrity Rules.

If the ‘Order’ table is considered, it contains three foreign key references on Customer_id, Chef_id, Tables_id. If the table definition is noticed certain rules are enforced on Updation and Deletion.

‘CONSTRAINT fk_Orders_Customer_id FOREIGN KEY (Customer_id) REFERENCES rms.Customer(Customer_id) ON UPDATE CASCADE ON DELETE NO ACTION’

The above foreign key constraint is from the Orders table where on UPDATE CASCADE means if a record from parent table is Updated then it is Updated in the child table. On DELETE NO ACTION means if a user wants to delete the record in the parent table and it’s related record exist in the child table then it shows an error message on that the column that it is foreign key referenced.

‘CONSTRAINT fk_Orders_Chef_id FOREIGN KEY (Chef_id) REFERENCES rms.Chef(Chef_id) ON UPDATE CASCADE ON DELETE SET NULL’

The above foreign key constraint is from the Orders table on Chefid column. Here if a record is updated in the parent table then it is updated on the child as well. ON DELETE SET NULL means if a record is deleted from the chef table then in the order table it should set the NULL value because the chef may not be the active participant in the restaurant but might hold some orders and orders are related to the customers so it is not recommended to delete those records.

Likewise, Referential Integrity is enforced on all the foreign keys for each table one can notice the constraints on the table definitions given above. The database implemented in this project for restaurant management follows the Integrity Rules.

6. Normalization

Normalization is a database design technique, it divides the larger table into smaller ones by providing the relationships between the tables Moreover, it helps in eliminating the Update, Insert, delete anomalies and reduce redundancy. The normal forms are based on the Candidate keys and Functional dependencies.

Candidate key*:* Let r be any relation of schema R let k is a subset of R. if the following two conditions holds then the key is candidate key.

Uniqueness Property: The no two tuples in r have the same value for k

Irreducibility property: No proper subset of k has the uniqueness property.

Functional Dependencies*:* Given a relation r on Schema R with x,y subset of R, Relation r satisfies the functional dependency X→Y if for every X value in r , πy(σX=x(r)) has precisely one tuple.

The three main normal forms are 1NF, 2NF,3NF.

Before discussing the Normal forms lets discuss the Prime and Nonprime attribute.

On a relational schema R an attribute A in R and a set of FD's over R. Attribute A is prime in R with respect to F if A is contained in Candidate key of R. If A is not contained in Candidate key of R then it is Non-Prime Attribute.

First Normal Form: A relation schema R is in 1NF if the values in the domain are atomic for every attribute in relational schema

Second Normal Form: A relation schema R is in 2NF, the relation should be in 1NF and every non prime attribute is fully dependent on candidate key of a Relation.

Third Normal Form: A relation schema R is in 3NF if no nonprime attribute in R is transitively dependent upon Candidate key of R.

All the tables in this project are in 1NF, 2NF, 3NF. Let's check each table.

The table 'Orders' is in 1NF because the values in the domain are atomic for every attribute in the schema. For example, every order id is related to one customer. This table is in 2NF because every non prime attribute is fully dependent on candidate key. In this table the candidate key is order_id which is a prime attribute is fully dependent on the non prime attribute columns like order_date, order status. This table is in 3NF because, for a given set of fd's form the relation no nonprime attribute is transitively dependent on the Candidate key order_id.

The table 'Customer' is in 1NF because the values in the domain are atomic for every attribute in the schema. For example, every customer_id there exist only one record in the customer table. This table is in 2NF because every non prime attribute is fully dependent on candidate key. In this table the candidate key is Customer_id which is a prime attribute is fully dependent on the non prime attribute columns like Username, E-mail. This table is in 3NF because, for a given set of fd's which are derived from the table there is no nonprime attribute is transitively dependent on the Candidate key Customer_id.

The table 'Order items' is in 1NF because the values in the domain are atomic for every attribute in the schema. For example, every order_item_id there exist only one record in the order item table. This table is in 2NF because every non prime attribute is fully dependent on candidate key. In this table the candidate key is order_item_id which is a prime attribute is fully dependent on the non prime attribute columns like Quantity, Unit price. This table is in 3NF because, for a given set of fd’s which are derived from the table there is no nonprime attribute is transitively dependent on the Candidate key order_item_id.

The table ‘Menu’ is in 1NF because the values in the domain are atomic for every attribute in the schema. For example, every item_id there exist only one item in the Menu table. This table is in 2NF because every non prime attribute is fully dependent on candidate key. In this table the candidate key is item_id which is a prime attribute is fully dependent on the non prime attribute columns like item name, item price. This table is in 3NF because, for a given set of fd’s which are derived from the table there is no nonprime attribute is transitively dependent on the Candidate key item_id.

The table ‘Chef’ is in 1NF because the values in the domain are atomic for every attribute in the schema. For example, every chef_id there exist only one chef in the Chef table. This table is in 2NF because every non prime attribute is fully dependent on candidate key. In this table the candidate key is chef_id which is a prime attribute is fully dependent on the non prime attribute columns like name, phone. This table is in 3NF because, from the table there is no nonprime attribute is transitively dependent on the Candidate key Chef_id.

The table ‘Payment’ is in 1NF because the values in the domain are atomic for every attribute in the schema. For example, every payment_id there exist only one record in the payment table. This table is in 2NF because every non prime attribute is fully dependent on candidate key. In this table the candidate key is payment_id which is a prime attribute is fully dependent on the non prime attribute columns payment date, total bill. This table is in 3NF because, from the table there is no nonprime attribute is transitively dependent on the Candidate key Payement_id.

The table ‘feedback’ is in 1NF because the values in the domain are atomic for every attribute in the schema. For example, every feedback_id there exist only one record in the feedback table. This table is in 2NF because every non prime attribute is fully dependent on candidate key. In this table the candidate key is Feedback_id which is a prime attribute is fully dependent on the non prime attribute columns feedback text, Ratings. This table is in 3NF because, from the table there is no nonprime attribute is transitively dependent on the Candidate key Feedback_id.

The table ‘Tables’ is in 1NF because the values in the domain are atomic for every attribute in the schema. For example, every Table_id there exist only one record in the Tables table. This table is in 2NF because every non prime attribute is fully dependent on candidate key. In this table the candidate key is Table_id which is a prime attribute is fully dependent on the non prime attribute columns table status, table chairs. This table is in 3NF because, from the table there is no nonprime attribute is transitively dependent on the Candidate key Table_id.

7. Indexes

When Indexes are applied on the tables or views it seep up the retrieval process. Generally, Indexes are used by queries for faster retrieval of data this is very useful when our database contains huge number of records. To understand the indexes one can relate this topic with the indexes present in the book for example In order to go to a chapter in a book one first refers the index page check the page number and then directly jump to that chapter. The Indexes work in a similar way.

If Indexes are not applied, for a query to obtain the desired results has to scan the each and every record in the table and this decreases the query performance. So its always good idea to design a database which include indexes.

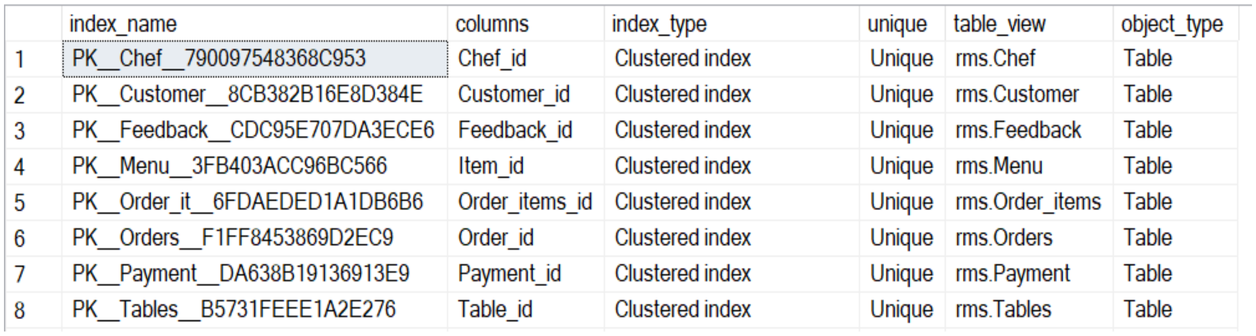
Now let’s discuss how to choose index columns or when to apply indexes in a table. Indexes are applied on the columns which are most commonly used in query conditions. If a referential integrity or unique key constraint exist on a column. On those columns’ indexes can be applied.

There are Many types of Indexes, here in this report the two important indexes types are discussed.

- Clustered Indexes
- Non-Clustered Indexes

Clustered Indexes: Clustered Index determines the physical order of data stored in a table, that is based on a column all the values in the table are sorted more over the values are sorted based on one condition. In Sql Server, it automatically creates clustered indexes on the Primary Key Column. There can be only one Clustered Indexes in a table.

Below fig shows the clustered indexes created on the tables with respect to columns in the Restaurant Management Database

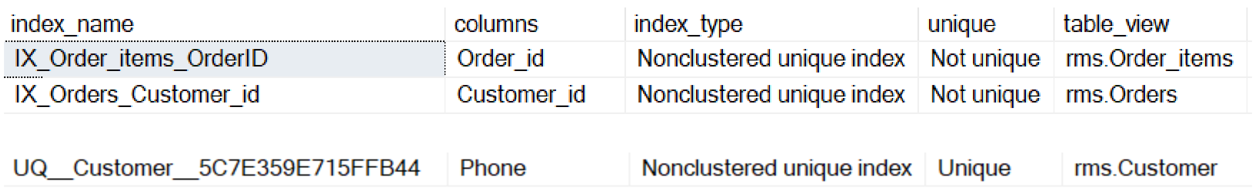


	index_name	columns	index_type	unique	table_view	object_type
1	PK__Chef__790097548368C953	Chef_id	Clustered index	Unique	rms.Chef	Table
2	PK__Customer__8CB382B16E8D384E	Customer_id	Clustered index	Unique	rms.Customer	Table
3	PK__Feedback__CDC95E707DA3ECE6	Feedback_id	Clustered index	Unique	rms.Feedback	Table
4	PK__Menu__3FB403ACC96BC566	Item_id	Clustered index	Unique	rms.Menu	Table
5	PK__Order_it__6FDAEDED1A1DB6B6	Order_items_id	Clustered index	Unique	rms.Order_items	Table
6	PK__Orders__F1FF8453869D2EC9	Order_id	Clustered index	Unique	rms.Orders	Table
7	PK__Payment__DA638B19136913E9	Payment_id	Clustered index	Unique	rms.Payment	Table
8	PK__Tables__B5731FEEE1A2E276	Table_id	Clustered index	Unique	rms.Tables	Table

Table 7.1 Clustered Indexes

Non-Clustered Indexes:

In Non-Clustered Indexes the indexes are stored at one place and the data is stored at other place similar to index in a book. The indexes use the pointers to the location of the data. Unlike clustered indexes there can be many Non-Clustered Indexes. When a query is executed based on the column that has the non clustered indexes, the database first check the indexes and get the address where the row is stored in the table and fetches the record. The following Non clustered Indexes are created in the created Restaurant Management database.



index_name	columns	index_type	unique	table_view
IX_Order_items_OrderID	Order_id	Nonclustered unique index	Not unique	rms.Order_items
IX_Orders_Customer_id	Customer_id	Nonclustered unique index	Not unique	rms.Orders
UQ__Customer__5C7E359E715FFB44	Phone	Nonclustered unique index	Unique	rms.Customer

Table 7.2: Non Clustered indexes

The Orderid is the primary key in the Orders table but it is foreign key in many others table but In Order_items table on the column Order_id Non clustered index is created because the column holds a referential integrity constraint and more over this column is most frequently used while fetching the order details.

Likewise, In the Orders tables on the Customer_id column a non clustered index is created because this column is also most frequently queried column in fetching the orders of customers.

8. Triggers

Triggers are executed automatically when a specified action is performed on a table. There are two types of triggers they are.

- Data Manipulation Language (DML)
- Data Definition Language (DDL)

Data Definition Language: These triggers are invoked when Create, Drop, Alter commands are issued on the table.

Data Manipulation Language: These triggers are invoked automatically when Insert, Delete, Update operations are performed on the table.

DML triggers are further classified as

- After or For trigger
- Instead of trigger

After trigger executes after the Insert, Update, Delete actions are performed on the table. All the referential constraints specified need to be achieved before it triggers a trigger.

Instead of triggers, when Insert, Delete, Update commands are issued on a table instead of performing the action on the given table it performs on the table mentioned in the trigger.

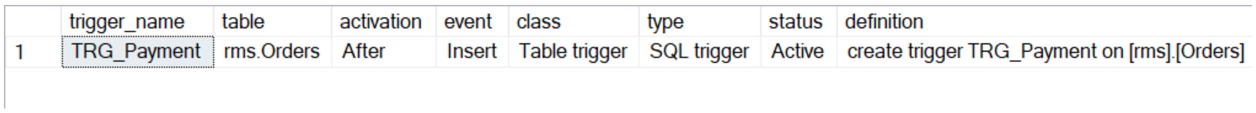
In the context of triggers there are two tables used in the triggers they are ‘Inserted’, ’Deleted’. In simpler terms in the Inserted table the new records are available while in the deleted old records are available.

To demonstrate the triggers practically, Let’s apply them on the Restaurant Management system.

Scenario: When an order is placed in to the system, then a payment id for that order needs to be generated automatically.

For the above scenario, we require two tables Orders and Payment. When a record is inserted in the 'Order' table then its Payment information needs to be generated in the payment table. But as soon as the order is placed there may be chance that the payment is done by the customer only after the order is served and enjoyed by the customer. To handle this in the Payment table there is a 'Payment status' column. By default the payment status is 'Not completed'. When a transaction is completed the admin can update the payment status or we can automate the process. But here I am trying to use the concept of triggers for payment id generation as soon as the order placed.

```
create trigger TRG_Payment on [rms].[Orders]
After Insert
AS
BEGIN
declare @Order_id int;
declare @Order_datetime datetime;
declare @Total_bill money;
select @Order_id = Order_id from inserted;
select @Order_datetime  = Order_datetime from inserted;
select @Total_bill = Total_bill from inserted;
Insert into [rms].[Payment]
(Payment_Date,Total_bill,Payment_Status,Order_id)
values(@Order_datetime,@Total_bill,'Not Done',@Order_id)
END
GO
```



	trigger_name	table	activation	event	class	type	status	definition
1	TRG_Payment	rms.Orders	After	Insert	Table trigger	SQL trigger	Active	create trigger TRG_Payment on [rms].[Orders]

Table 8: Trigger Information

9. Stored Procedures

If a SQL Query is most often used in the database then creation of stored procedure helps. Stored procedure holds certain number of SQL statements it can be reused multiple times. If user has restrictions on the table but there is a scenario where he needs to use the table. So a stored procedure is created with the SQL statement and the user can use the Stored proc instead of table.

While building a front end application and integrating the database it is always good idea to build stored procedures instead of giving direct access to the front end developers. It also preserves the data integrity.

In this project, While building the front end application direct tables are never used instead created a stored procedure and used that proc in the frond end. Many stored procedures are created for each module for selection, updation, insertion and deletion.

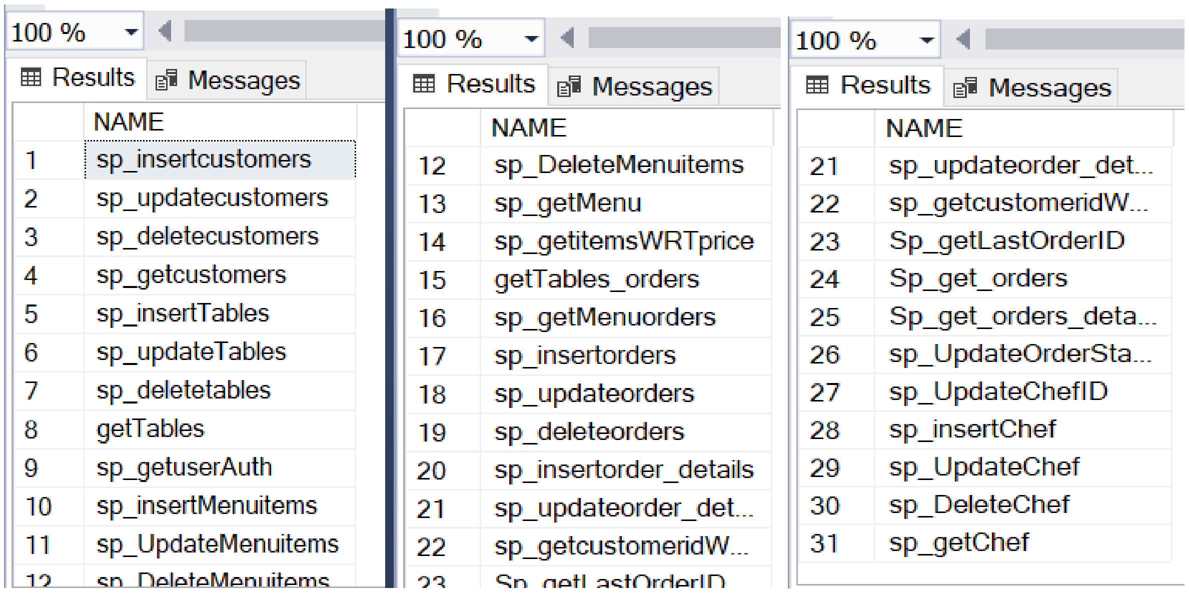
Scenario1: When a customer selects an item from the menu the price must be automatically reflected in the menu. For this a stored procedure is created and used while developing the front end.

```
create procedure rms.sp_getitemsWRTprice
@Item_id int
as
select [Item_price] as 'Price' from [rms].[Menu] where [Item_id] = @Item_id
```

Scenario2: When a chef is done with the order the chef has to update the order status in the orders table from pending to served for this a stored procedure is created.

```
Create procedure rms.sp_UpdateOrderStatus
@status varchar(30),
@OrderID int
as
update [rms].[Orders]
set
[Order_Status]=@status
where
[Order_id] = @OrderID
```

Likewise, Many stored procedures are created in the ‘Restaurant Management’ System. Below is the list of stored procedure created.

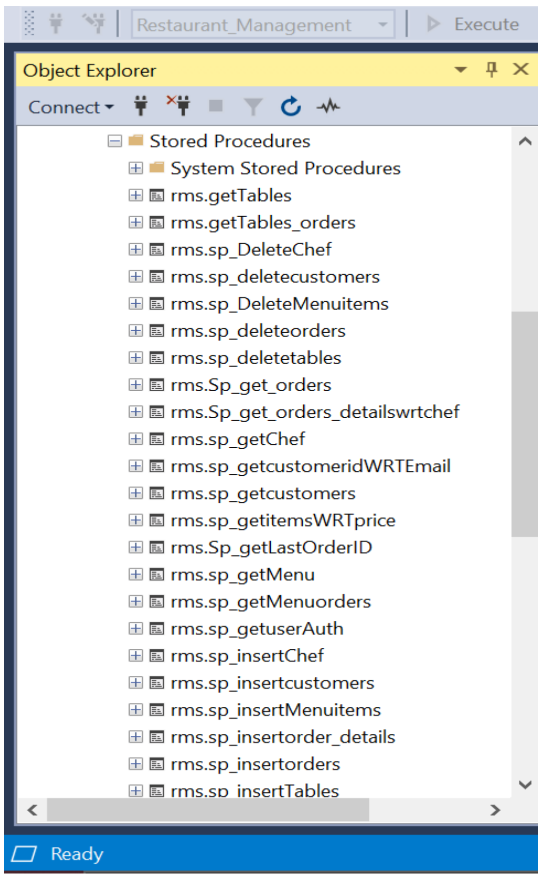


	NAME
1	sp_insertcustomers
2	sp_updatecustomers
3	sp_deletecustomers
4	sp_getcustomers
5	sp_insertTables
6	sp_updateTables
7	sp_deletetables
8	getTables
9	sp_getuserAuth
10	sp_insertMenuitems
11	sp_UpdateMenuitems
12	sp_DeleteMenuitems

	NAME
12	sp_DeleteMenuitems
13	sp_getMenu
14	sp_getitemsWRTprice
15	getTables_orders
16	sp_getMenuorders
17	sp_insertorders
18	sp_updateorders
19	sp_deleteorders
20	sp_insertorder_details
21	sp_updateorder_det...
22	sp_getcustomeridW...
23	Sp_getLastOrderID

	NAME
21	sp_updateorder_det...
22	sp_getcustomeridW...
23	Sp_getLastOrderID
24	Sp_get_orders
25	Sp_get_orders_deta...
26	sp_UpdateOrderSta...
27	sp_UpdateChefID
28	sp_insertChef
29	sp_UpdateChef
30	sp_DeleteChef
31	sp_getChef

Fig 9.1: Stored Procedures



Stored Procedures
- System Stored Procedures
- rms.getTables
- rms.getTables_orders
- rms.sp_DeleteChef
- rms.sp_deletecustomers
- rms.sp_DeleteMenuitems
- rms.sp_deleteorders
- rms.sp_deletetables
- rms.Sp_get_orders
- rms.Sp_get_orders_detailswrtchef
- rms.sp_getChef
- rms.sp_getcustomeridWRTEmail
- rms.sp_getcustomers
- rms.sp_getitemsWRTprice
- rms.Sp_getLastOrderID
- rms.sp_getMenu
- rms.sp_getMenuorders
- rms.sp_getuserAuth
- rms.sp_insertChef
- rms.sp_insertcustomers
- rms.sp_insertMenuitems
- rms.sp_insertorder_details
- rms.sp_insertorders
- rms.sp_insertTables

Fig 9.2: Stored Procedures

10. Designing Front-end

The frontend is designed using the programming language C# in the Visual Studio 2019.

Login Window: The below screenshot shows the login window of the customer. If a customer registers into the system then can login into the system.

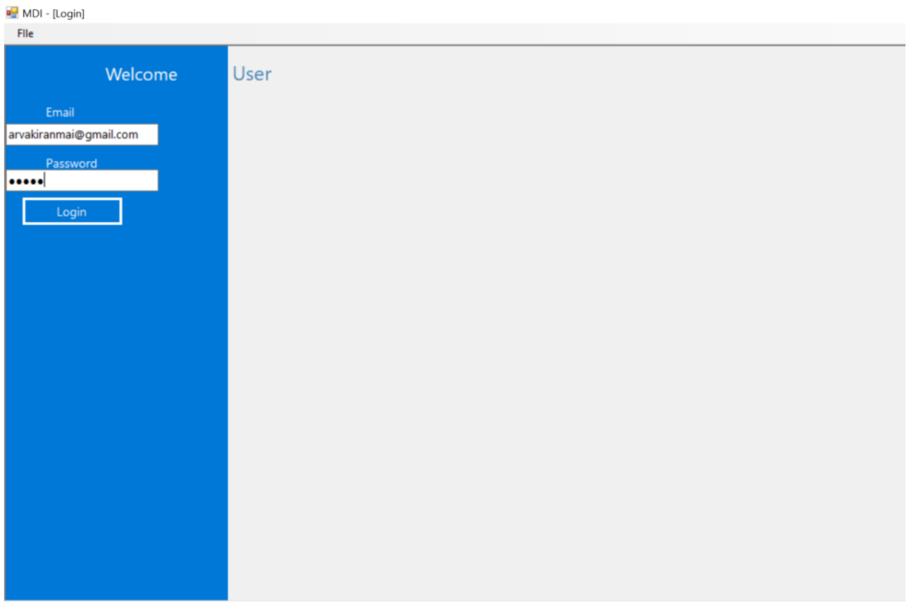
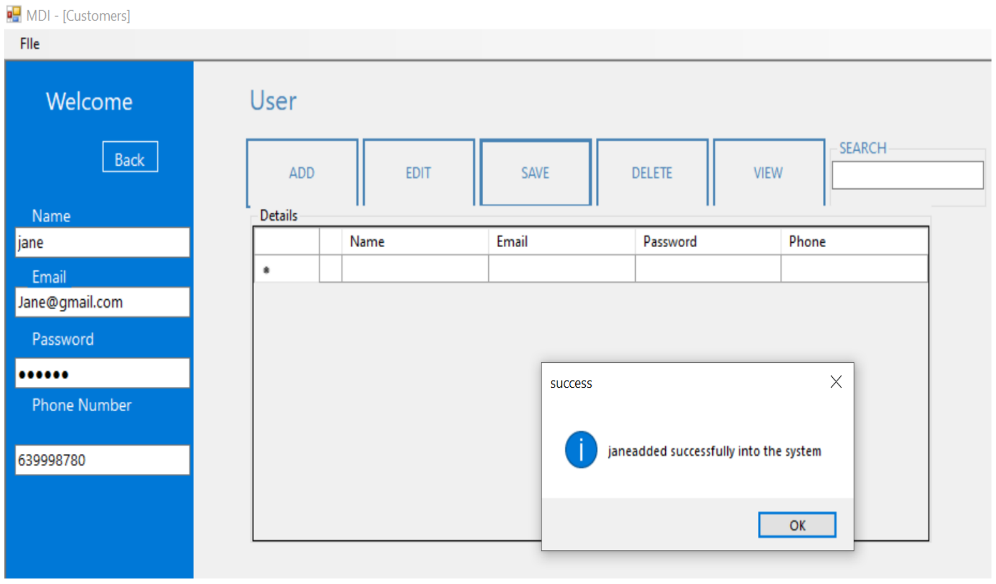


Fig 10.1: Login window

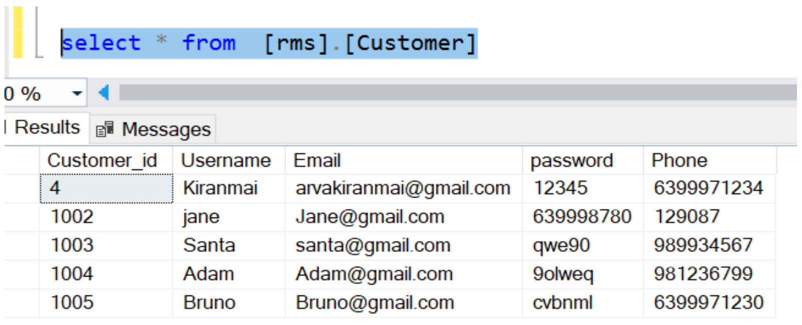
Customer Registration Window: The user can register into the system using this window



janeadded successfully into the system

Fig 10.2: Customer Registration Window

When a customer registers at the front end the data is added into the Customers table in the database.



```
select * from [rms].[Customer]
```

	Customer_id	Username	Email	password	Phone
	4	Kiranmai	arvakiranmai@gmail.com	12345	6399971234
	1002	jane	Jane@gmail.com	639998780	129087
	1003	Santa	santa@gmail.com	qwe90	989934567
	1004	Adam	Adam@gmail.com	9olweq	981236799
	1005	Bruno	Bruno@gmail.com	cvbnml	6399971230

Fig 10.3: Data in the customers table

Admin Window: When the Admin logins to the system, the home screen of admin looks like below screen shot with various buttons like Menu, Add Chef, Table. By clicking on the button lands on the respective page.

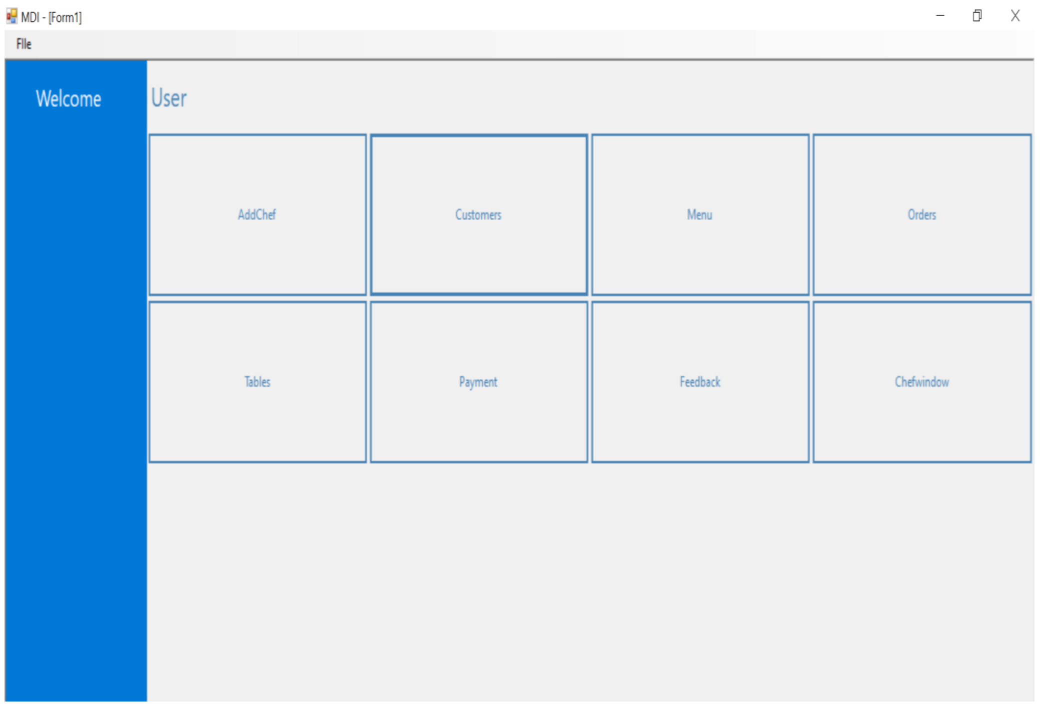
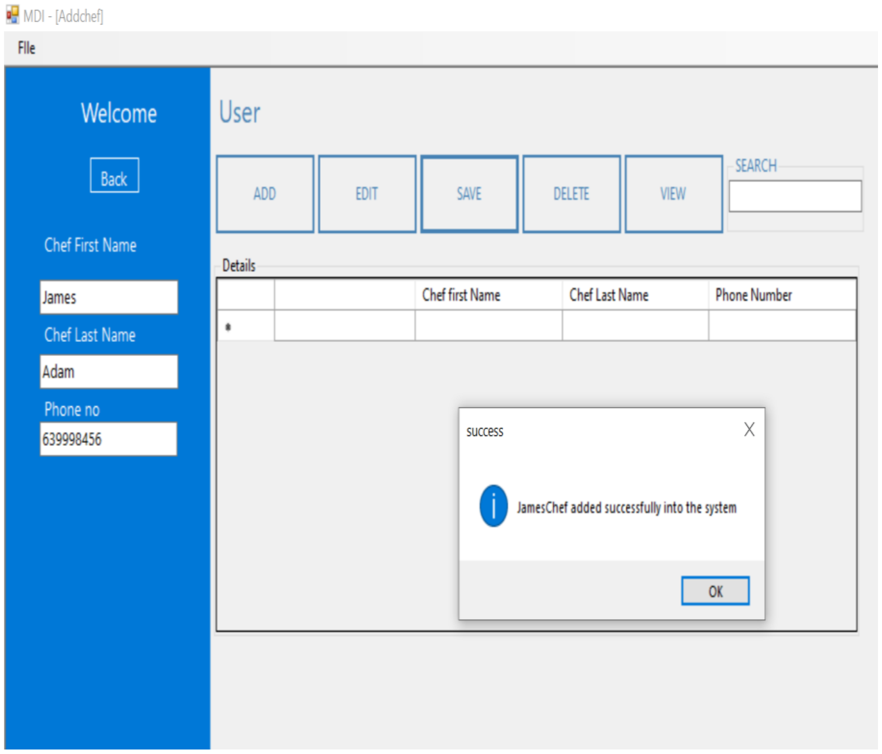


Fig 10.4: Admin window

Functionalities of Admin:

- Can Add new chef information to the system.
- Can view the chef information



Chef First Name: James

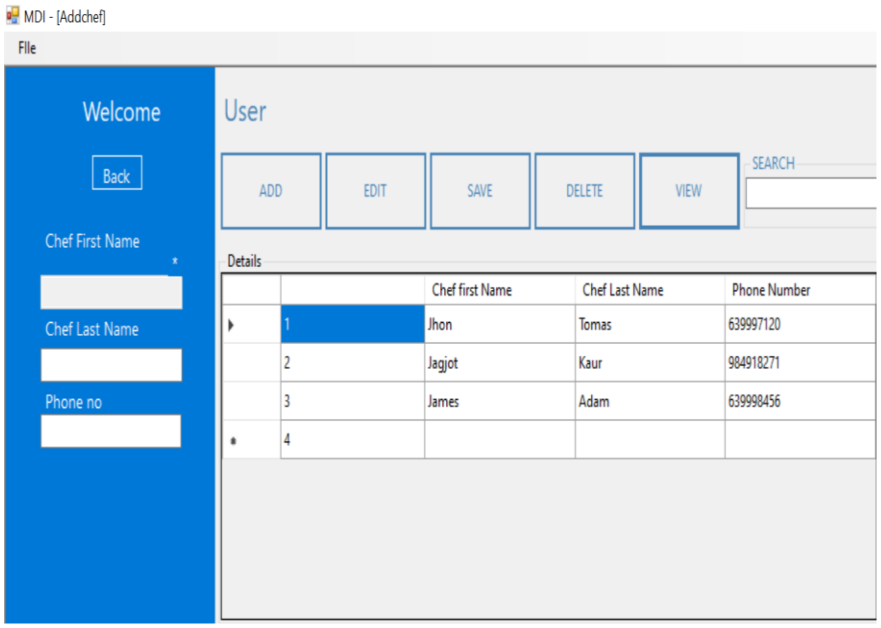
Chef Last Name: Adam

Phone no: 639998456

		Chef first Name	Chef Last Name	Phone Number
*				

JamesChef added successfully into the system

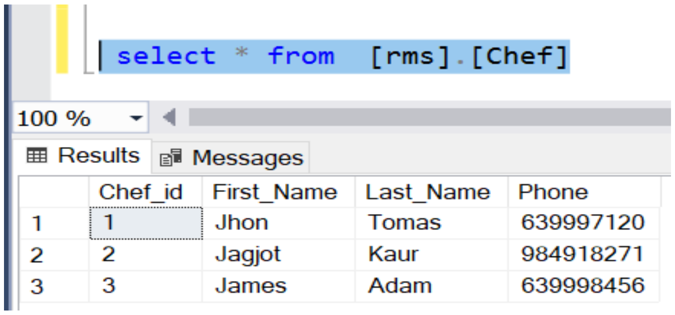
Fig 10.5: Adding chef info window



		Chef first Name	Chef Last Name	Phone Number
▶	1	Jhon	Tomas	639997120
	2	Jagjot	Kaur	984918271
	3	James	Adam	639998456
*	4			

Fig 10.6: View information

When Admin adds the chef information, it is stored inside the Chef table in the database.

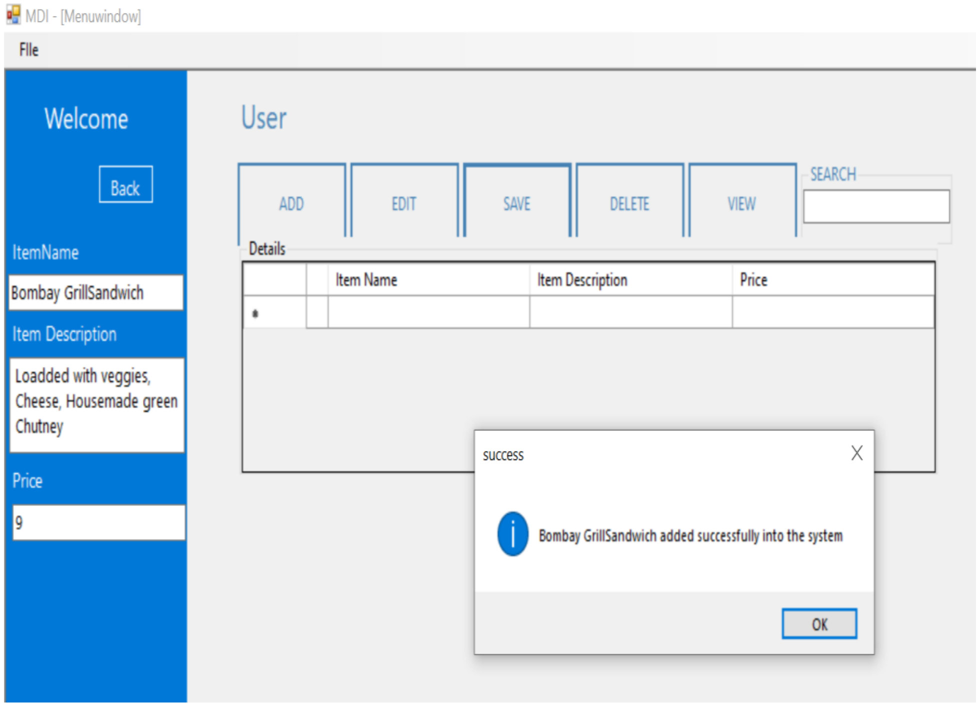


```
select * from [rms].[Chef]
```

	Chef_id	First_Name	Last_Name	Phone
1	1	Jhon	Tomas	639997120
2	2	Jagjot	Kaur	984918271
3	3	James	Adam	639998456

Fig 10.7: Data in the chef table

- Admin can Add, Edit, Update, Delete the Menu Items as shown below.



ItemName: Bombay GrillSandwich

Item Description: Loadded with veggies, Cheese, Housemade green Chutney

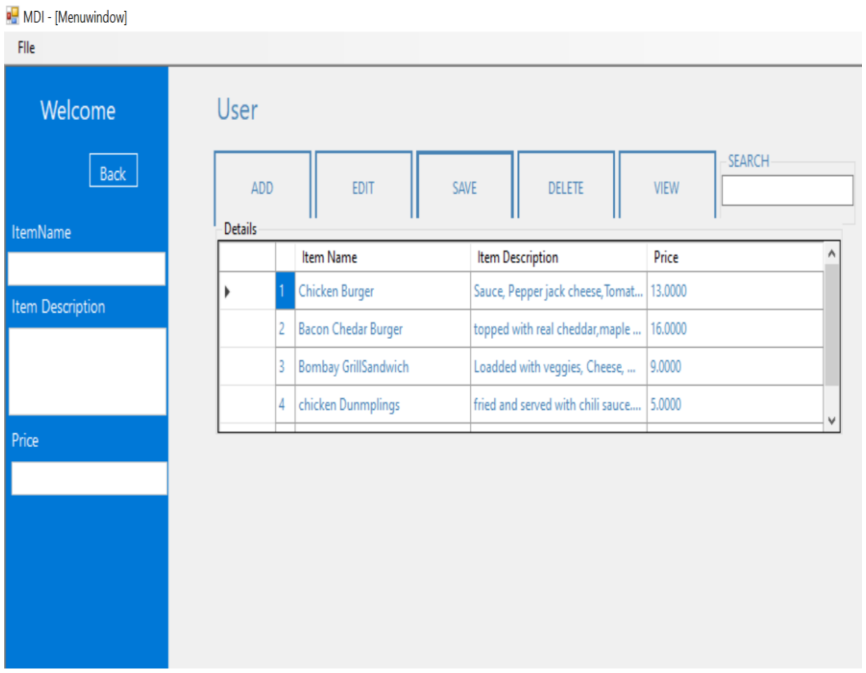
Price: 9

	Item Name	Item Description	Price
*			

success

Bombay GrillSandwich added successfully into the system

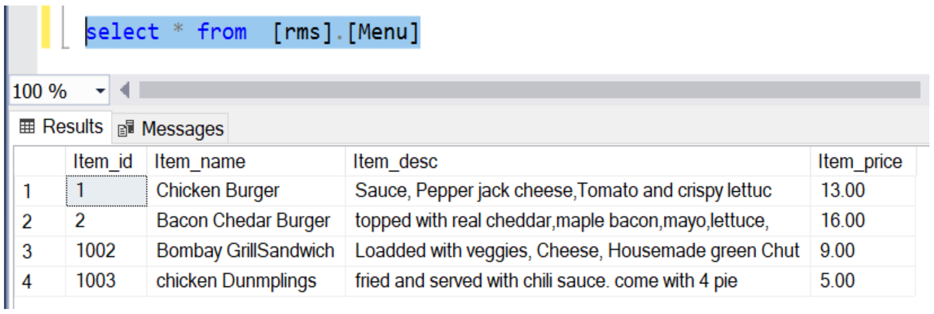
Fig 10.8: Add menus window



		Item Name	Item Description	Price
▸	1	Chicken Burger	Sauce, Pepper jack cheese, Tomat...	13.0000
	2	Bacon Chedar Burger	topped with real cheddar,maple ...	16.0000
	3	Bombay GrillSandwich	Loadded with veggies, Cheese, ...	9.0000
	4	chicken Dunmplings	fried and served with chili sauce....	5.0000

Fig 10.9: View Menu Information

When Items are added by the admin, the menu details are stored in the Menu Table



```
select * from [rms].[Menu]
```

	Item_id	Item_name	Item_desc	Item_price
1	1	Chicken Burger	Sauce, Pepper jack cheese,Tomato and crispy lettuc	13.00
2	2	Bacon Chedar Burger	topped with real cheddar,maple bacon,mayo,lettuce,	16.00
3	1002	Bombay GrillSandwich	Loadded with veggies, Cheese, Housemade green Chut	9.00
4	1003	chicken Dunmplings	fried and served with chili sauce. come with 4 pie	5.00

Fig 10.10: Menu data

- Admin can Add, Update, Delete the table information. Moreover, table status can be updated here.

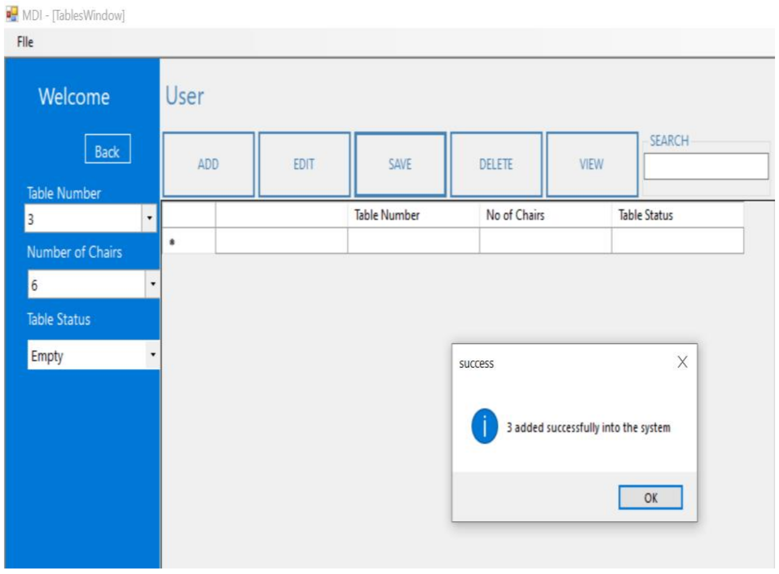


Table Number: 3

Number of Chairs: 6

Table Status: Empty

success

3 added successfully into the system

OK

Fig 10.11: Tables window

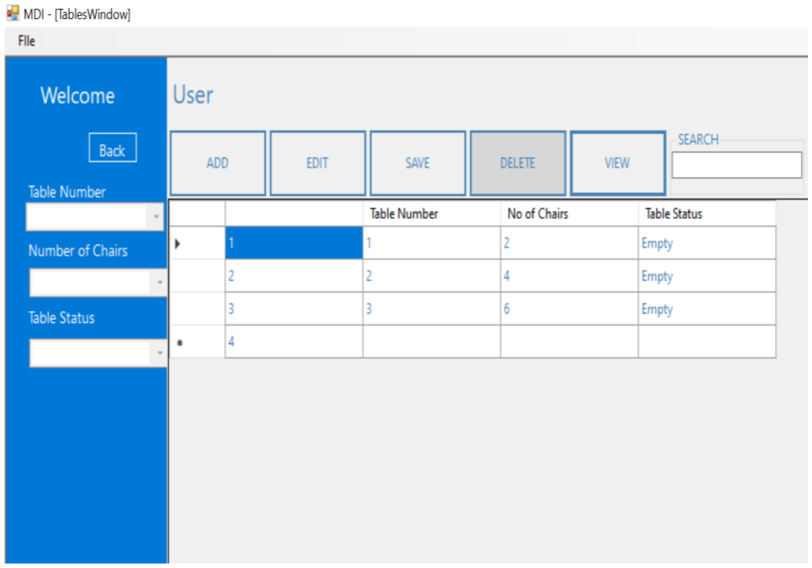
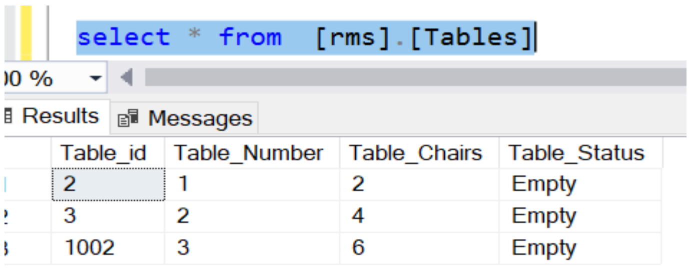


	Table Number	No of Chairs	Table Status
1	1	2	Empty
2	2	4	Empty
3	3	6	Empty
4			

Fig 10.12: Add table window.

The table information is added in the 'Tables' table in the database

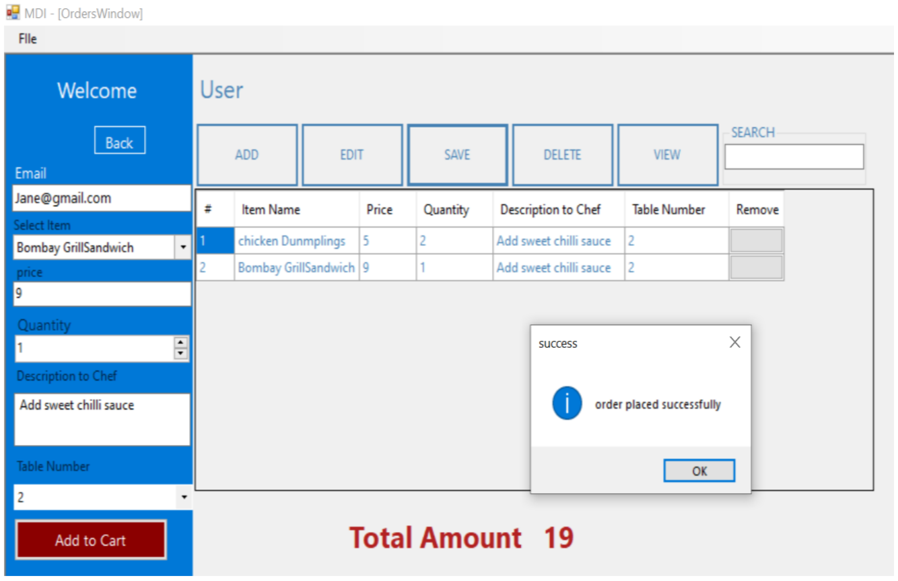


```
select * from  [rms].[Tables]
```

Table_id	Table_Number	Table_Chairs	Table_Status
2	1	2	Empty
3	2	4	Empty
1002	3	6	Empty

Fig 10.13: Data in the Tables data

Orders Window: The customer after Login can view the orders window, where on the left hand side the can view the menu and add items to the cart. When the order is saved it displays the 'order Placed Successfully' Message.



Email: Jane@gmail.com

Select Item: Bombay GrillSandwich

price: 9

Quantity: 1

Description to Chef: Add sweet chilli sauce

Table Number: 2

#	Item Name	Price	Quantity	Description to Chef	Table Number	Remove
1	chicken Dunmplings	5	2	Add sweet chilli sauce	2	
2	Bombay GrillSandwich	9	1	Add sweet chilli sauce	2	

success

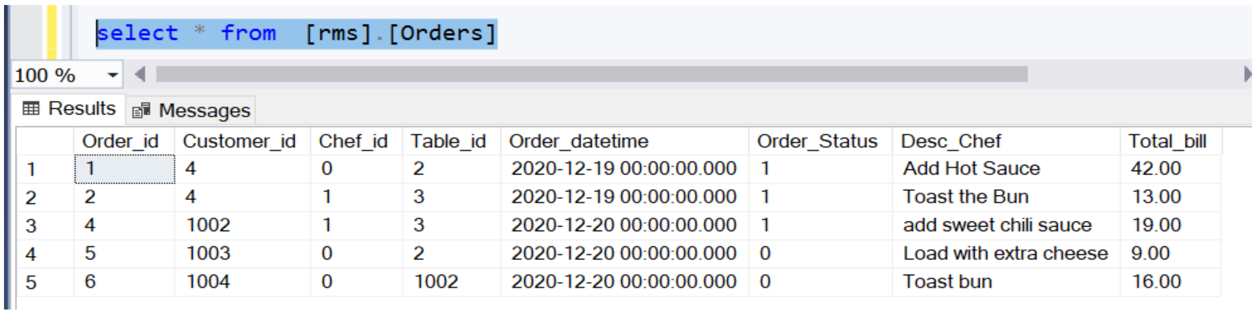
order placed successfully

OK

Total Amount 19

Fig 10.14: Orders window

When the customer adds the order, the order is stored in the ‘Orders Window’

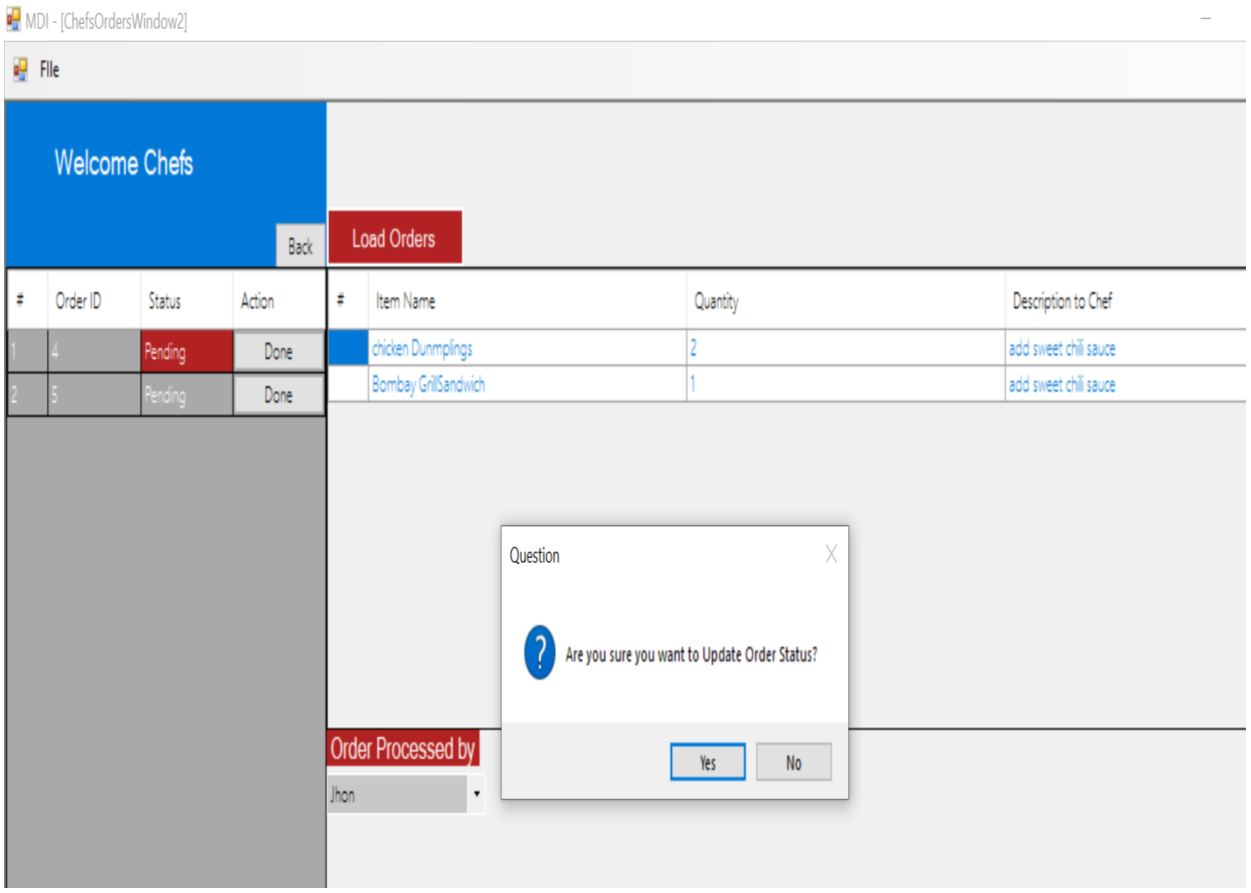


```
select * from [rms].[Orders]
```

	Order_id	Customer_id	Chef_id	Table_id	Order_datetime	Order_Status	Desc_Chef	Total_bill
1	1	4	0	2	2020-12-19 00:00:00.000	1	Add Hot Sauce	42.00
2	2	4	1	3	2020-12-19 00:00:00.000	1	Toast the Bun	13.00
3	4	1002	1	3	2020-12-20 00:00:00.000	1	add sweet chili sauce	19.00
4	5	1003	0	2	2020-12-20 00:00:00.000	0	Load with extra cheese	9.00
5	6	1004	0	1002	2020-12-20 00:00:00.000	0	Toast bun	16.00

Fig 10.15: Data in the orders window

Chef Window: Chef can view the orders by clicking on the Load button present on the top. All the orders will be loaded on the left side. By clicking on each order, the order details will be displayed. Once the chef is done with the order can select his name from the drop-down menu available at the bottom ‘Order Processed by’. After selecting the name by clicking on the done button next to the order can update the ‘Order status’ from pending to served. After the status is updated that order is removed from this window.



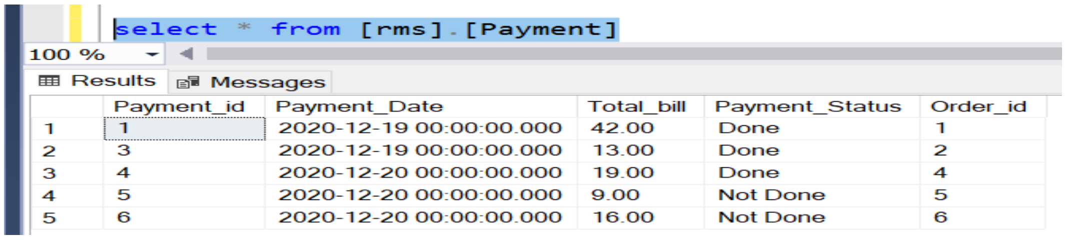
#	Order ID	Status	Action
1	4	Pending	Done
2	5	Pending	Done

#	Item Name	Quantity	Description to Chef
	chicken Dunmplings	2	add sweet chili sauce
	Bombay GrillSandwich	1	add sweet chili sauce

Are you sure you want to Update Order Status?

Fig 10.16: Chef window

Payment Table

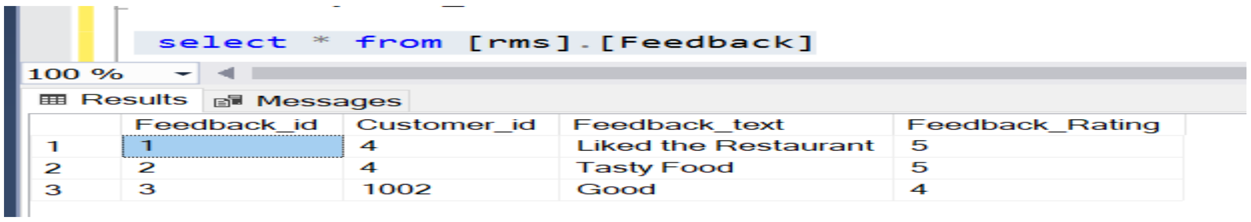


```
select * from [rms].[Payment]
```

	Payment_id	Payment_Date	Total_bill	Payment_Status	Order_id
1	1	2020-12-19 00:00:00.000	42.00	Done	1
2	3	2020-12-19 00:00:00.000	13.00	Done	2
3	4	2020-12-20 00:00:00.000	19.00	Done	4
4	5	2020-12-20 00:00:00.000	9.00	Not Done	5
5	6	2020-12-20 00:00:00.000	16.00	Not Done	6

Fig 10.17: Data in the payment table

Feedback Table



```
select * from [rms].[Feedback]
```

	Feedback_id	Customer_id	Feedback_text	Feedback_Rating
1	1	4	Liked the Restaurant	5
2	2	4	Tasty Food	5
3	3	1002	Good	4

Fig 10.18: Data in the feedback table

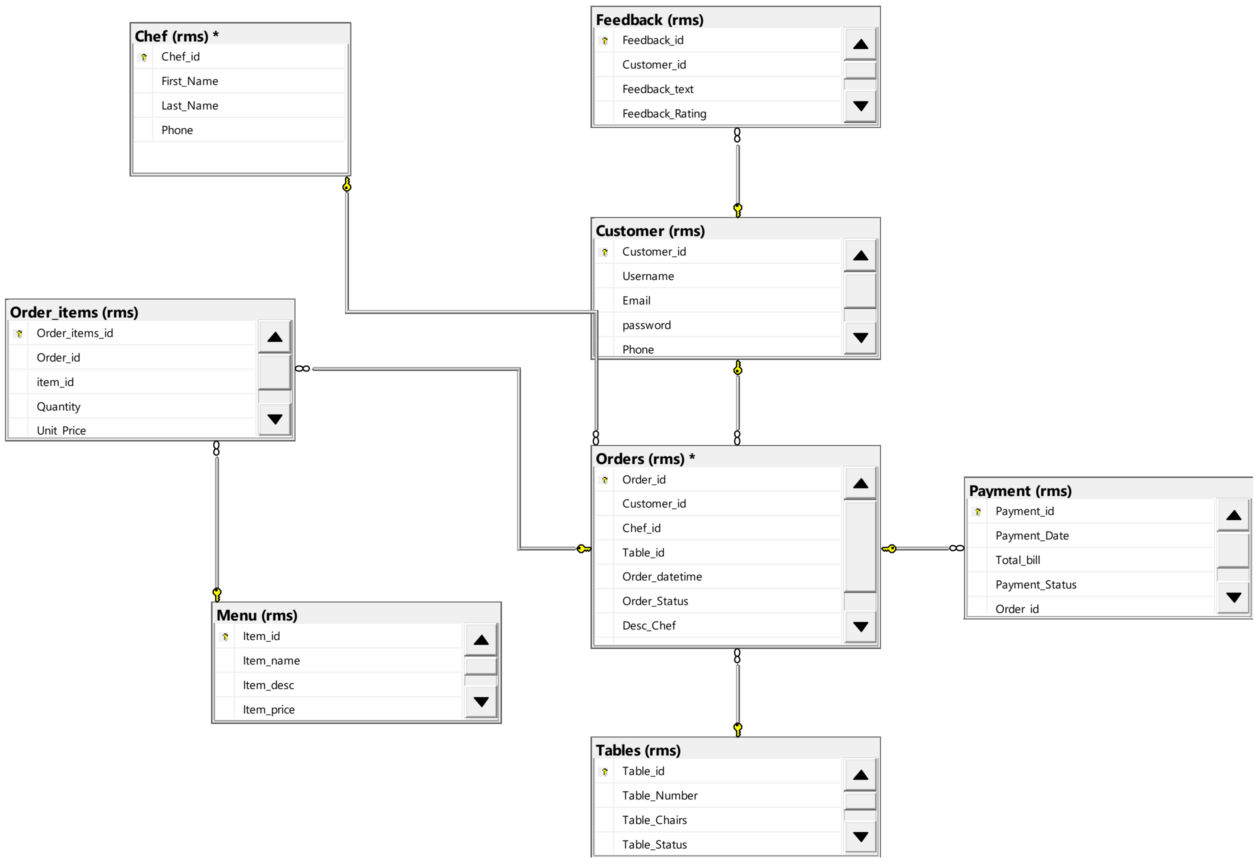
11. Conclusion

To conclude, A database design and Implementation is done for the ‘Restaurant Management system’. The main aim of this project is to avoid the paper-based communication between the customers and chefs. The customers can login to the system and order the items. The orders are directly reflected in the chef window. Chef can process the order and update the status of the order. This way the customer need not wait for the waiters to take the order, whenever the customers are ready, they can directly place the order in the system.

The designed data holds all the Integrity rules, Normalizations. Indexes are also added to the database for faster retrieval of data when the data becomes huge in future. A fort is developed using the programming language C# in the visual studio and integrated the developed database with the frontend. Stored procedures are created instead of using the tables directly in the front end. The usage of triggers is demonstrated with an example in the restaurant management. The designed database is free from anomalies.

12. Appendix

ERD Diagram in SQL Server



13. References

[1]"An Essential Guide to SQL Server Indexes", *SQL Server Tutorial*, 2020. [Online]. Available: https://www.sqlservertutorial.net/sql-server-indexes/.

[2]"Database Indexes Explained - Essential SQL", *Essential SQL*, 2020. [Online]. Available: https://www.essentialsql.com/what-is-a-database-index/.

[3] Sardar Ali, A. (2019). Restaurant Management System [Video]. Retrieved from https://www.youtube.com/watch?v=7-aog3said0&list=PLzG4EZfLrZ31M-Iv9gosC7IrjIP08O4rQ&ab_channel=AliRazaSardarAli

[4]"SQL Trigger | Student Database - GeeksforGeeks", *GeeksforGeeks*, 2016. [Online]. Available: https://www.geeksforgeeks.org/sql-trigger-student-database/.

[5]"Entity Relationship Diagram (ERD) - What is an ER Diagram?", *Smartdraw.com*, 2017. [Online]. Available: https://www.smartdraw.com/entity-relationship-diagram/.