**Project Title:** Restaurant Loyalty Database

**Team Names:** Jeffrey Liu, Javier Ramirez, Joshua Parada

**Abstract**

Loyalty programs have been entering the market for many types of industry varying from fast food, gas company, online merchandise, and more. Our database will be developed to manage and maintain data for a loyalty program for a restaurant. This project will create a database system that can keep track of employee and customer information to provide better services through the restaurant’s loyalty program. The database will allow customers to set up a loyalty account to allow them access to the loyalty and reward program which can be managed by employees. It also offers information on menu items, order details, payment details, and reservation services (our database will be based on a restaurant that is reservation only, no walk-ins). To put it simply, this database system offers restaurants the ability to provide incentives for their customers to revisit because of their loyalty program while allowing access to certain data to improve work efficiency and promote customer service.

**Mission Statement**

The goal of this project is to provide a system that is able to efficiently maintain the data created and used to support the loyalty program of a restaurant, manage their customer and employee information, and promote easier access to this data.

**Mission Objectives**

To maintain (enter, update, and delete) data on Employees.

To maintain (enter, update, and delete) data on Customers.

To maintain (enter, update, and delete) data on Services.

To maintain (enter, update, and delete) data on Redemptions.

To maintain (enter, update, and delete) data on Rewards.

To maintain (enter, update, and delete) data on Orders.

To maintain (enter, update, and delete) data on Menu\_Order.

To maintain (enter, update, and delete) data on Menus.

To maintain (enter, update, and delete) data on Reservation.

To maintain (enter, update, and delete) data on TableServ.

To perform searches on Employee.

To perform searches on Customers.

To perform searches on Services.

To perform searches on Redemptions.

To perform searches on Rewards.

To perform searches on Order.

To perform searches on Menu\_Order.

To perform searches on Menus.

To perform searches on Reservations.

To perform searches on TableServes.

To report on Employee.

To report on Customers.

To report on Services.

To report on Redemptions.

To report on Rewards.

To report on Orders.

To report on Menu\_Order.

To report on Menus.

To report on Reservations.

To report on Tables.

**Major User Views**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Data** | **Access Type** | **Owner** | **Manager** | **Worker** | **Customer** |
| All Employees | Maintain |  |  |  |  |
| Query | x |  |  |  |
| Report | x |  |  |  |
| Single Employee | Maintain |  | x |  |  |
| Query |  | x | x |  |
| Report |  | x | x |  |
| All Tables | Maintain |  |  |  |  |
| Query | x |  |  |  |
| Report | x |  |  |  |
| Single Table | Maintain |  | x |  |  |
| Query |  | x | x |  |
| Report |  | x | x |  |
| All Services | Maintain |  |  |  |  |
| Query | x |  |  |  |
| Report | x |  |  |  |
| Single Service | Maintain |  | x | x |  |
| Query |  | x | x |  |
| Report |  | x | x |  |
| All Reservations | Maintain |  |  |  |  |
| Query | x |  |  |  |
| Report | x |  |  |  |
| Single Reservation | Maintain |  | x | x |  |
| Query |  | x | x |  |
| Report |  | x | x |  |
| All Customers | Maintain |  |  |  |  |
| Query | x |  |  |  |
| Report | x |  |  |  |
| Single Customer | Maintain |  | x |  | x |
| Query |  | x | x | x |
| Report |  | x | x |  |
| All Redemptions | Maintain |  |  |  |  |
| Query | x |  |  |  |
| Report | x |  |  |  |
| Single Redemption | Maintain |  | x |  |  |
| Query |  | x | x | x |
| Report |  | x | x |  |
| All Rewards | Maintain |  |  |  |  |
| Query | x |  |  |  |
| Report | x |  |  |  |
| Single Reward | Maintain |  | x |  |  |
| Query |  | x | x | x |
| Report |  | x | x |  |
| All Menus | Maintain |  |  |  |  |
| Query | x |  |  |  |
| Report | x |  |  |  |
| Single Menu | Maintain |  | x |  |  |
| Query |  | x | x |  |
| Report |  | x | x |  |
| All Menu\_Orders | Maintain |  |  |  |  |
| Query |  |  |  |  |
| Report |  |  |  |  |
| Single Menu\_Order | Maintain |  | x |  |  |
| Query |  | x | x |  |
| Report |  | x | x |  |
| All Orders | Maintain |  |  |  |  |
| Query | x |  |  |  |
| Report | x |  |  |  |
| Single Order | Maintain |  | x |  |  |
| Query |  | x | x | x |
| Report |  | x | x |  |

**Entity Relationship Diagram**

Reward

RewardId {PK}

RewardName

MenuId [FK]

PointsRequired

PriceDeducted

Available

Redemption

RedemptionId {PK}

RewardId [FK]

CustomerId [FK]

RedeemDate

RedeemTime

Menu

MenuId {PK}

MenuName

MenuCategory

MenuCost

Provides

Contains

1..\*

0..1

1..\*

0..\*

0..\*

Has

0..\*

1..1

Menu\_Order

MenuId [FK]

OrderId [FK]

Quantity

Customer

CustomerId {PK}

CustomerFirstName

CustomerLastName

CustomerPhoneNum

CustomerEmail

DateOfBirth

JoinDate

TotalPoints

1..\*

Sends

Employee

EmployeeId {PK}

EmployeeFirstName

EmployeeLastName

Position

Wage

EmployeeAddress

EmployeePhoneNum

EmployeeEmail

1..1

1..1

Sends

1..\*

Order

OrderId {PK}

CustomerId [FK]

OrderDate

OrderTime

BalanceDue

Gratuity

Discounts

Total

PointsEarned

1..\*

Submits

1..\*

0..1

Reservation

ReservationId {PK}

ReserveDate

ReserveTime

NumOfPeople

Receives

Handles

0..\*

1..\*

TableServ

TableId {PK}

EmployeeId [FK]

Seats

IsIndoor

0..5

Service

ServiceId {PK}

TableId [FK]

CustomerId [FK]

ReservationId [FK]

DateTaken

TimeTaken

IsCompleted

Has

1..\*

Offers

1..\*

1..1

**Entity Descriptions**

**Employee**

The Employee entity provides information for all the staff that works for the restaurant. It has a primary key called EmployeeId and other attributes that contain personal information. EmployeeId will be an INT that increments by 1 starting from 1 and it can also be referenced as a foreign key in an enumerated table called Table\_List where an employee manages at most 5 tables. EmployeeFirstName, EmployeeLastName, and Position will be in VARCHAR allowing only alphabetical characters and hyphen (for names only). Wage will be DECIMAL type that represents hourly pay and can only contain 2 digits after the decimal. EmployeeAddress will be VARCHAR that follows the format 12345 Street City, State Zip-code. EmployeePhoneNum will be VARCHAR type that is constrained to 10 digits exactly. EmployeeEmail will be VARCHAR of the format {email}@{email\_domain}.{domain} in which case {email} is the user’s email containing alphanumeric characters, {email\_domain} is the email domain being used that is strictly alphabetical, and {domain} is the web domain such as .com, .edu, etc that is also strictly alphabetical.

**TableServ**

The TableServ entity provides listed information of tables in the restaurant and is enumerated. The primary key for the table is TableId and is an INT type that increments starting from 1 until the maximum table the restaurant has. It references to the Service table as a foreign key to indicate which table the service is taking place. EmployeeId is a foreign key that indicates which employee is responsible for this table. The Seats attribute is an INT that indicates how many seats this table can accommodate and will range from 2 to 8. The IsIndoor attribute is a BIT type in which 0 and 1 would represent false and true, respectively. This is used in conjunction with reservation if the party prefers indoor or outdoor.

**Service**

The Service entity provides information of the customers being handled in conjunction with the Table Reservation entities. ServiceId will be the primary key of INT type that increments starting from 1. It will also contain three foreign keys, TableId, CustomerId, and ReservationId to mark which tables are occupied by a loyalty customer via reservation. Values for the two latter foreign key attributes can be NULL if the customer has not registered for the loyalty program and the customer did not make a reservation to get service for the table occupied. DateTaken and TimeTaken will be VARCHAR type of the formats ##-##-#### and ##:## (24-hour clock), respectively, whereas ‘#’ represents numerical values only. The attribute IsCompleted is a BIT type where 1 represents the service is completed, indicating that the table is available for later reservation. On the front-end side, there would be rules where customers can only be seated or reserve a table only if the most recent instance of a certain TableId has IsCompleted marked as 1 to indicate it is free.

**Reservation**

The Reservation entity provides information of the reservations the customers could make, assigning them a specific table at the restaurant with time and date. It has a primary key called ReservationId that is INT type and increments starting from 1. In order to identify the customer who made the reservation, a join query must be called with Service where the ReservationId and CustomerId acts as the foreign keys for the Service entity. There is also a ReserveDate and ReserveTime that follows the same rules as DateTaken and TimeTaken from the Service entity table mentioned previously. Finally, there is NumOfPeople attribute that is of an INT type that counts the number of people. It will accept a value between 2 to 8 to reflect upon the number of seats available for a table mentioned in Table entity above. If a customer has more than 8, another instance of a ReservationId pointing to the same CustomerId would be created instead.

**Customer**

The Customer entity provides information of the customer that signed up for the loyalty program for the restaurant. It includes personal information similar to the Employee entity with the exception that it does not contain position, wage, and address, but it does include DateOfBirth, JoinDate, and TotalPoints used for redemption purposes (refer to Redemption entity). The data type and format for most of the Customer attributes follow the one mentioned in Employee (EmployeeId for CustomerId, EmployeeFirstName for CustomerFirstName, so on and so forth). CustomerId is the primary key for this table and acts as the foreign key for the Service table, Order table, and Redemption table. DateOfBirth and JoinDate will have the DATE data type and are used for the front-end side of the Rewards entity. Customers will be offered rewards on their birthdays and on anniversaries of their join date. TotalPoints is an INT type that is at least 0 representing the number of points the customer has accumulated. These points are earned based on the amount they have spent for an order (refer to Order entity) and can be spent on rewards (refer to Reward entity).

**Redemption**

The Redemption entity provides information of the rewards that the customers have redeemed. Employees will use the information from here, joined by the information with their respective reward from the Reward table, to reduce prices on the customers’ order listed as Discounts in Order Table. The primary key is RedemptionId which is INT type that increments starting from 00000001 (8-digit format). There are also two foreign keys, RewardId and CustomerId, to indicate what reward a customer has redeemed. RedeemDate and RedeemTime has the DATE type and TIME type, respectively. The time would exclude the seconds from the TIME data type.

**Reward**

The Reward entity provides information about the list of rewards that the customers can redeem. It describes the menu item that is to be discounted and how much is discounted. The primary key is RewardId and is incremented as INT type starting from 1. RewardName is an attribute of VARCHAR type that briefly describes the reward. MenuId is a foreign key from the Menu entity so that it can mark the available discounts offered for certain menu items. PointsRequired is INT type that has a value of at least 1 and indicates how many points the customer needs to have to be able to redeem this reward. This will subtract the TotalPoints for that respective customer. PriceDeducted is a DECIMAL type with 2 digits after the decimal place and is used as Discounts for the Order entity. Available is an attribute of BIT type that simply indicates whether the reward is available at the current time.

**Menu**

The Menu entity provides information on the list of food and beverages the restaurant offers. It has the primary key MenuId which is an INT type that increments starting from 1. It acts as a foreign key in both Reward and Menu\_Order entities to indicate what item the said entity tables are referencing to. MenuName and MenuCategory are VARCHAR values that describe the food or beverage and are strictly alphanumerical including white-space characters. Examples for MenuCategory would be something like appetizer, chicken entrée, desserts, alcoholic beverages, etc. Finally, we have MenuCost that is a DECIMAL type with 2 digits after the decimal place, which simply shows the price of the item.

**Menu\_Order**

The Menu\_Order entity is an associative table that links the details between Menu and Order table. The purpose of this table is to be able to list all the menu items for an order so that the total price could be calculated later in the Order entity. It contains 2 foreign keys, MenuId and OrderId, and an attribute called Quantity that is an INT type of at least 1. The foreign keys here can be used as a candidate key as no two items on this table will have both the exact same MenuId and OrderId. OrderId cannot be used as a superkey because this table can contain different MenuId pointing to the same OrderId and MenuId cannot be a superkey because the same MenuId can point to different OrderId.

**Order**

The Order entity provides information of the order and payments the customer must provide. Its primary key is OrderId which is an INT type that increments starting from 1 and is used as a foreign key for Menu\_Order with reasons stated for that entity above. CustomerId is a foreign key to show who the order is assigned to. OrderDate and OrderTime follow the data type and format mentioned previously for dates and times. BalanceDue is DECIMAL type (2-digits after decimal) to show the initial balance due before gratuity and/or discounts are applied. Following the same data type and format, Gratuity, Discounts, and Total are attributes indicating how much the initial cost from BalanceDue could be added or subtracted to get the grand total (Total as attribute). Finally, PointsEarned is an INT type that is of at least 1 and is reflective based on BalanceDue. This should not consider gratuity and discounts applied and is strictly based on what the original cost was to show how much points can be earned by the customer which is added to the TotalPoints in the Customer entity.

**Relational Model – Table Attributes and Normalizations**

*Legends: Primary Key – {PK}, Foreign Key – [FK], Candidate Key – [CK]*

**Employee**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute** | **Data Type** | **Domain** | **Constraints** | **Default** |
| EmployeeId | INT |  | {PK} | AUTOINCREMENT |
| EmployeeFirstName | VARCHAR(30) | alphabets, hyphens | NOT NULL |  |
| EmployeeLastName | VARCHAR(30) | alphabets, hyphens | NOT NULL |  |
| Position | VARCHAR(30) | alphabets, spaces, hyphens | NOT NULL |  |
| Wage | DECIMAL |  | NOT NULL |  |
| EmployeeAddress | VARCHAR(100) | alphanumeric, spaces, hyphens | NOT NULL |  |
| EmployeePhoneNum | VARCHAR(10) | numeric | NOT NULL |  |
| EmployeeEmail | VARCHAR(100) | alphanumeric, symbols (only dot, underscore, and hyphen) | CHECK(‘%@%.%’),  NOT NULL |  |

Dependencies

EmployeeId 🡪 EmployeeFirstName, EmployeeLastName, Position, Wage, EmployeeAddress, EmployeePhoneNum, EmployeeEmail

Primary Key: EmployeeId

Normalization at 4NF: The table satisfies 1NF because all intersection contains 1 and only 1 value. It satisfies 2NF as there are no partial dependency since EmployeeId is the only candidate key and the other seven attributes directly depend on it. It is in 3NF because there is no transitive dependency (None of the seven attributes, excluding EmployeeId from the table, depend on each other). One might argue that Wage depends on Position, but the theory behind this table is that no employee of the same position are guaranteed to be paid the same wage, so Wage is actually just dependent on EmployeeId. It is BCNF as the listed dependency is a candidate key producing each of the non-prime attributes. Lastly, it is also 4NF because there are no multivalued dependencies as EmployeeId can only produce one of each of the other seven attributes (they cannot have multiple information in the other field).

**TableServ**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute** | **Data Type** | **Domain** | **Constraints** | **Default** |
| TableId | TINYINT |  | {PK} | AUTOINCREMENT |
| EmployeeId | INT |  | [FK] in Employee |  |
| Seats | TINYINT | 2 to 8 | NOT NULL |  |
| IsIndoor | BIT |  | NOT NULL |  |

Dependencies

TableId 🡪 EmployeeId, Seats, IsIndoor

Primary Key: TableId

Normalization at 4NF: The table satisfies 1NF because all intersection contains 1 and only 1 value. It satisfies 2NF as there are no partial dependency since TableId is the only candidate key and the other three attributes directly depend on it. It is in 3NF because there is no transitive dependency (EmployeeId, Seats, and IsIndoor do not depend on each other in any combination). It is BCNF as all the listed dependencies are a candidate key producing a non-prime attribute. Lastly, it is also 4NF because there are no multivalued dependencies as TableId can only produce one of each of the other three attributes.

**Service**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute** | **Data Type** | **Domain** | **Constraints** | **Default** |
| ServiceId | INT |  | {PK} | AUTOINCREMENT |
| TableId | TINYINT |  | [FK] in Table, NOT NULL |  |
| CustomerId | INT |  | [FK] in Customer |  |
| ReservationId | INT |  | [FK] in Reservation |  |
| DateTaken | DATE |  | NOT NULL |  |
| TimeTaken | TIME | 09:00 to 21:30 | CHECK (‘##:##’) |  |
| IsCompleted | BIT |  | NOT NULL |  |

Dependencies

ServiceId 🡪 TableId, CustomerId, ReservationId, DateTaken, TimeTaken, IsCompleted

Primary Key: ServiceId

Normalization at 4NF: The table satisfies 1NF because all intersection contains 1 and only 1 value. It satisfies 2NF as there are no partial dependency since ServiceId is the only candidate key and the other six attributes directly depend on it. It is in 3NF because there is no transitive dependency (All non-prime attributes, besides ServiceId, do not depend on each other). One might argue that ReservationId produces CustomerId as a reservation can only be assigned to single customer, but the Service table is already an associative table that links between Customer table and Reservation table (and Table table as well). It is BCNF as all the listed dependencies are a candidate key producing a non-prime attribute. Lastly, it is also 4NF because there are no multivalued dependencies as ServiceId can only produce one of each of the other three attributes.

**Reservation**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute** | **Data Type** | **Domain** | **Constraints** | **Default** |
| ReservationId | INT |  | {PK} | AUTOINCREMENT |
| ReserveDate | DATE |  | NOT NULL |  |
| ReserveTime | TIME | 09:00 to 21:30 | CHECK (‘##:##’), NOT NULL |  |
| NumOfPeople | TINYINT | 2 to 16 | NOT NULL |  |

Dependencies

ReservationId 🡪 ReserveDate, ReserveTime, NumOfPeople

Primary Key: ReservationId

Normalization at 4NF: The table satisfies 1NF because all intersection contains 1 and only 1 value. It satisfies 2NF as there are no partial dependency since ReservationId is the only candidate key and the other three attributes directly depend on it. It is in 3NF because there is no transitive dependency (ReservationTime, ReserveDate, and NumOfPeople do not depend on each other in any combination). It is BCNF as all the listed dependencies are a candidate key producing a non-prime attribute. Lastly, it is also 4NF because there are no multivalued dependencies as ReservationId can only produce exactly one of each of the other three attributes.

**Customer**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute** | **Data Type** | **Domain** | **Constraints** | **Default** |
| CustomerId | INT |  | {PK} | AUTOINCREMENT |
| CustomerFirstName | VARCHAR(30) | alphabets, hyphens | NOT NULL |  |
| CustomerLastName | VARCHAR(30) | alphabets, hyphens | NOT NULL |  |
| CustonerPhoneNum | VARCHAR(10) | numeric | NOT NULL |  |
| CustomerEmail | VARCHAR(100) | alphanumeric, symbols (only dot, underscore, and hyphen), 1 @ symbol | CHECK(‘%@%.%’), NOT NULL |  |
| DateOfBirth | DATE |  | NOT NULL |  |
| JoinDate | DATE |  | NOT NULL |  |
| TotalPoints | SMALLINT |  | NOT NULL | 0 |

Dependencies

CustomerId 🡪 CustomerFirstName, CustomerLastName, CustomerPhoneNum, CustomerEmail, DateOfBirth, JoinDate, TotalPoints

Primary Key: CustomerId

Normalization at 4NF: The table satisfies 1NF because all intersection contains 1 and only 1 value. It satisfies 2NF as there are no partial dependency since CustomerId is the only candidate key and the other seven attributes directly depend on it. It is in 3NF because there is no transitive dependency (None of the seven attributes, excluding CustomerId from the table, depend on each other). It is BCNF as the listed dependency is a candidate key producing each of the non-prime attributes. Lastly, it is also 4NF because there are no multivalued dependencies as CustomerId can only produce one of each of the other seven attributes (they cannot have multiple information in the other fields. Thus, there can only be one record for each CustomerId).

**Redemption**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute** | **Data Type** | **Domain** | **Constraints** | **Default** |
| RedemptionId | INT |  | {PK} |  |
| RewardId | TINYINT |  | [FK] in Reward, NOT NULL |  |
| CustomerId | INT |  | [FK] in Customer, NOT NULL |  |
| RedeemDate | DATE |  | NOT NULL |  |
| RedeemTime | TIME | 09:00 to 21:30 | CHECK (‘##:##’), NOT NULL |  |

Dependencies

RedemptionId 🡪 RewardId, CustomerId, RedeemDate, RedeemTime

Primary Key: RedemptionId

Normalization at 4NF: The table satisfies 1NF because all intersection contains 1 and only 1 value. Multiple iteration of a reward redeemed by the same one customer would be under different redemption. It satisfies 2NF as there are no partial dependency since RedemptionId is the only candidate key and the other four attributes directly depend on it. It is in 3NF because there is no transitive dependency (RewardId, CustomerId, RedeemDate, and RedeemTime do not depend on each other in any combination). It is BCNF as the listed dependency is a candidate key producing each of the non-prime attributes. Lastly, it is also 4NF because there are no multivalued dependencies as RedemptionId can only produce exactly one of each of the other four attributes.

**Reward**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute** | **Data Type** | **Domain** | **Constraints** | **Default** |
| RewardId | TINYINT |  | {PK} |  |
| RewardName | VARCHAR(50) |  | NOT NULL |  |
| MenuId | TINYINT |  | [FK] in Menu, NOT NULL |  |
| PointsRequired | SMALLINT |  | NOT NULL |  |
| PriceDeducted | SMALLINT |  | NOT NULL |  |
| Available | BIT |  | NOT NULL | 0 |

Dependencies

RewardId 🡪 RewardName, MenuId, PointsRequired, PriceDeducted, Available

Primary Key: RewardId

Normalization at 4NF: The table satisfies 1NF because all intersection contains 1 and only 1 value. It satisfies 2NF as there are no partial dependency since RewardId is the only candidate key and the other five attributes directly depend on it. It is in 3NF because there is no transitive dependency (the non-primed attributes listed on the right-hand side of the listed dependency do not depend on each other in any combination). It is BCNF as the listed dependency is a candidate key producing each of the non-prime attributes. Lastly, it is also 4NF because there are no multivalued dependencies as RewardId can only produce exactly one of each of the other five attributes.

**Menu**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute** | **Data Type** | **Domain** | **Constraints** | **Default** |
| MenuId | TINYINT |  | {PK} | AUTOINCREMENT |
| MenuName | VARCHAR(30) | alphabets and spaces | NOT NULL |  |
| MenuCategory | VARCHAR(20) | alphabets and spaces | NOT NULL |  |
| MenuCost | DECIMAL |  | NOT NULL |  |

Dependencies

MenuId 🡪 MenuName, MenuCategory, MenuCost

Primary Key: MenuId

Normalization at 4NF: The table satisfies 1NF because all intersection contains 1 and only 1 value. It satisfies 2NF as there are no partial dependency since MenuId is the only candidate key and the other three attributes directly depend on it. It is in 3NF because there is no transitive dependency (MenuName, MenuCategory, and MenuCost do not depend on each other in any combination). However, one may argue that MenuCategory can possibly depend on MenuName, but the primary key (MenuId) set for this table has to be unique, so MenuCategory can solely depend on MenuId. It is BCNF as all the listed dependencies are a candidate key producing a non-prime attribute. Lastly, it is also 4NF because there are no multivalued dependencies as MenuId can only produce exactly one of each of the other three attributes.

**Menu\_Order**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute** | **Data Type** | **Domain** | **Constraints** | **Default** |
| MenuId | TINYINT |  | [FK] in Menu, [CK] with OrderId, NOT NULL |  |
| OrderId | INT |  | [FK] in Order, [CK] with MenuId, NOT NULL |  |
| Quantity | TINYINT |  | NOT NULL | 1 |

Dependencies

MenuId, OrderId 🡪 Quantity

Primary Key: {MenuId, OrderId}

Normalization at 4NF: The table satisfies 1NF because all intersection contains 1 and only 1 value. If there are additional MenuId for a single OrderId, it would be in a separate row with the same OrderId. It satisfies 2NF as there are no partial dependency since Quantity depends on the candidate key which is the combination of MenuId and OrderId. It is in 3NF because there is no transitive dependency (Quantity is the only non-prime attribute and it depends on the candidate key {MenuId, OrderId}). It is BCNF as the chosen candidate key produces a non-prime attribute. Lastly, it is also 4NF because there are no multivalued dependencies as the candidate key, {MenuId, OrderId}, can only produce exactly one Quantity field.

**Order**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute** | **Data Type** | **Domain** | **Constraints** | **Default** |
| OrderId | INT |  | {PK} |  |
| CustomerId | INT |  | [FK] in Customer, NOT NULL |  |
| OrderDate | DATE |  | NOT NULL |  |
| OrderTime | TIME | 09:00 to 21:30 | CHECK('##:##') |  |
| BalanceDue | DECIMAL |  | NOT NULL |  |
| Gratuity | DECIMAL |  | NOT NULL |  |
| Discounts | DECIMAL |  | NOT NULL |  |
| Total | DECIMAL |  | NOT NULL |  |
| PointsEarned | SMALLINT |  | NOT NULL |  |

Dependencies

OrderId 🡪 CustomerId, OrderDate, OrderTime, BalanceDue, Gratuity, Discounts, Total

BalanceDue, Discount 🡪 PointsEarned

Primary Key: OrderId

Normalization at 2NF: The table satisfies 1NF because all intersection contains 1 and only 1 value. It satisfies 2NF as there are no partial dependency since OrderId is the only candidate key and the other seven attributes (with the exception of PointsEarned) directly depend on it. It fails to satisfy 3NF because there is a transitive dependency such that PointsEarned is directly proportional to (and dependent on) BalanceDue and Discount on a case that it is applied to the balance (disregarding gratuity) after discount is applied. It will fail BCNF and 4NF since 3NF fails for this table.

**Use Cases**

**List of Actors:** Owner, Manager, Worker, Customer

**Actor – Manager**

*Entity: Employee*

(1) Search an Employee from Restaurant

1. Manager will click “Employee” button
2. Manager will click “Search” button
3. Prompt manager to enter Employee Name or Employee ID
4. List of possible employee(s) will populate

(Search by employee ID)

**SELECT \***

**FROM Employee**

**WHERE EmployeeId = 5;**

(Search by employee name)

**SELECT \***

**FROM Employee**

**WHERE EmployeeFirstName = ‘John’ AND EmployeeLastName = ‘Doe’;**

(2) Enter a new Employee to Restaurant

1. Manager will click “Employee” button
2. Manager will click “Add New Employee” button
3. Prompt manager to input employee information
4. Manager will click “Save Entry” button

**INSERT INTO Employee**

**VALUES (EmployeeId, EmployeeFirstName, EmployeeLastName, Position, Wage, EmployeeAddress, EmployeePhoneNum, EmployeeEmail);**

(3) Update an Employee from Restaurant

1. Manager will click “Employee” button
2. Manager will click “Edit Employee” button
3. Prompt manager to enter Employee ID or Employee Name
4. Prompt manager to select employee information to make changes to
5. Manager will click “Save Entry” button

**UPDATE FROM Employee**

**SET EmployeeID = ‘’, EmployeeName= ‘’, Position= ‘’, Salary= ‘’, Address Phone = ‘’, Email = ‘’**

**WHERE EmployeeId = 5;**

(4 Remove an Employee from Restaurant

1. Manager will click “Employee” button
2. Manager will click “Remove Employee” button
3. Prompt manager to enter Employee ID or Employee Name
4. Manager will click “Delete Entry” button
5. Manager will click “Confirm Deletion” button

**DELETE FROM** **Employee**

**WHERE EmployeeId = 5;**

*Entity: Menu*

(5) Search a Menu from Restaurant

1. Manager will click “Menu” button
2. Manager will click “Search” button
3. Prompt manager to enter menu name or menu category
4. List of possible Menu(s) will populate

(Search by menu name)

**SELECT \***

**FROM Menu**

**WHERE MenuName = ‘Supreme Pizza’;**

(Search by menu category)

**SELECT \***

**FROM Menu**

**WHERE MenuCategory = ‘appetizer’;**

(6) Enter a new Menu entry

1. Manager will click “Menu” button
2. Manager will click “Add New Menu” button
3. Prompt manager to input menu information
4. Manager will click “Save Entry” button.

**INSERT INTO Menu**

**VALUES (MenuId, MenuName, MenuCategory, MenuCost);**

(7) Update a Menu entry

1. Manager will click “Menu” button
2. Manager will click “Edit Menu” button
3. Prompt manager to enter Menu ID or Menu Name
4. Prompt manager to select menu information to make changes to
5. Manager will click “Save Entry” button

**UPDATE FROM Menu**

**SET MenuName= ‘’, MenuCategory = ‘’, MenuCost = ‘’**

**WHERE MenuId = 5;**

(8) Delete a Menu entry

1. Manager will click “Menu” button
2. Manager will click “Delete Menu” button
3. Prompt manager to enter Menu ID or Menu Name
4. Manager will click “Delete Entry” button
5. Manager will click “Confirm Deletion” button

**DELETE FROM Menu**

**WHERE** **MenuId = 5;**

*Entity: Redemption*

(9) Search a Redemption from Restaurant

1. Manager will click “Redemption” button
2. Manager will click “Search” button
3. Prompt manager to enter Redemption ID or Customer ID
4. List of possible redemption(s) will populate

(Search by Customer ID)

**SELECT \***

**FROM Redemption**

**WHERE CustomerId = 5;**

(Search by Redemption ID)

**SELECT \***

**FROM Redemption**

**WHERE RedemptionId = 5;**

(10) Enter a new Redemption entry

1. Manager will click “Redemption” button
2. Manager will click “Add New Redemption” button
3. Prompt manager to input redemption information
4. Manager will click “Save Entry” button.

**INSERT INTO Redemption**

**VALUES (RedemptionId, RewardId, CustomerId, RedeemDate, RedeemTime);**

(11) Update a Redemption entry

1. Manager will click “Redemption” button
2. Manager will click “Edit Redemption” button
3. Prompt manager to enter Redemption ID or Customer ID
4. Prompt manager to select redemption information to make changes to
5. Manager will click “Save Entry” button

**UPDATE FROM Redemption**

**SET RewardId = ‘’, RedeemDate = ‘’, RedeemTime = ‘’**

**WHERE RedemptionId = 5;**

(12) Delete a Redemption entry

1. Manager will click “Redemption” button
2. Manager will click “Delete Redemption” button
3. Prompt manager to enter Redemption ID or Customer ID
4. Manager will click “Delete Entry” button
5. Manager will click “Confirm Deletion” button

**DELETE FROM Redemption**

**WHERE** **RedemptionId = 5;**

*Entity: Reward*

(13) Search a Reward from Restaurant

1. Manager will click “Reward” button
2. Manager will click “Search” button
3. Prompt manager to enter Reward ID or Reward Name
4. List of possible reward(s) will populate

(Search by reward ID)

**SELECT \***

**FROM Reward**

**WHERE RewardId = 5;**

(Search by reward name)

**SELECT \***

**FROM Reward**

**WHERE RewardName = ‘50% off French fries’;**

(14) Enter a new Reward entry

1. Manager will click “Reward” button
2. Manager will click “Add New Reward” button
3. Prompt manager to input reward information
4. Manager will click “Save Entry” button.

**INSERT INTO Reward**

**VALUES (RewardId, RewardName, MenuId, PointsRequired, PriceDeducted, Available);**

(15) Update a Reward entry

1. Manager will click “Reward” button
2. Manager will click “Edit Reward” button
3. Prompt manager to enter Reward ID or Reward Name
4. Prompt manager to select menu information to make changes to
5. Manager will click “Save Entry” button

**UPDATE FROM Reward**

**SET RewardName = ‘’, MenuId = ‘’, PointsRequired = ‘’, PriceDeducted = ‘’, Available = ‘’**

**WHERE RewardId = 5;**

(16) Delete a Reward entry

1. Manager will click “Reward” button
2. Manager will click “Delete Reward” button
3. Prompt manager to enter Reward ID or Reward Name
4. Manager will click “Delete Entry” button
5. Manager will click “Confirm Deletion” button

**DELETE FROM Reward**

**WHERE** **RewardId = 5;**

**Actor: Worker**

*Entity: Table*

(17) Search a Table from Restaurant

1. Worker will click “Table” button
2. Worker will click “Search” button
3. Prompt worker to enter table ID or number of seats
4. List of possible table(s) will populate

(Search by table ID)

**SELECT \***

**FROM Table**

**WHERE TableId = 5;**

(Search by number of seats)

**SELECT \***

**FROM Table**

**WHERE Seats = 4;**

(18) Enter a new Table entry

1. Worker will click “Table” button
2. Worker will click “Add New Table” button
3. Prompt worker to input Table information
4. Worker will click “Save Entry” button.

**INSERT INTO Table**

**VALUES (TableId, EmployeeId, Seats, IsIndoor);**

(19) Update a Table entry

1. Worker will click “Table” button
2. Worker will click “Edit Table” button
3. Prompt worker to enter Table Number
4. Prompt worker to select order information to make changes to
5. Worker will click “Save Entry” button

**UPDATE FROM Table**

**SET TableId= ‘’, EmployeeId = ‘’, Seats = ‘’, IsIndoor = ‘’**

**WHERE TableId = 5;**

(20) Delete a Table entry

1. Worker will click “Delete Table” button
2. Prompt worker to enter Table Number
3. Worker will click “Delete Entry” button
4. Worker will click “Confirm Deletion” button

**DELETE FROM Table**

**WHERE TableId = 5;**

*Entity: Customer*

(21) Search a Customer from Restaurant

1. Worker will click “Customer” button
2. Worker will click “Search” button
3. Prompt worker to enter customer ID or name
4. List of possible Customer(s) will populate

**SELECT \***

**FROM Customer**

**WHERE CustomerId= 5;**

(22) Enter a new Customer entry

1. Worker will click “Customer” button
2. Worker will click “Add New Customer” button
3. Prompt worker to input customer information
4. Worker will click “Save Entry” button.

**INSERT INTO Customer**

**VALUES (CustomerId, CustomerFirstName, CustomerLastName, CustomerPhoneNum, CustomerEmail, DateOfBirth, JoinDate, TotalPoints);**

(23) Update a Customer entry

1. Worker will click “Customer” button
2. Worker will click “Edit Customer” button
3. Prompt worker to enter Customer ID
4. Prompt worker to select order information to make changes to
5. Worker will click “Save Entry” button

**UPDATE FROM Customer**

**SET CustomerFirstName = ‘’, CustomerLastName = ‘’, CustomerPhoneNum = ‘’, CustomerEmail = ‘’, DateOfBirth = ‘’, JoinDate = ‘’, TotalPoints = ‘’**

**WHERE CustomerId = 5;**

(24) Delete a Customer entry

1. Worker will click “Customer” button
2. Worker will click “Delete Customer” button
3. Prompt worker to enter Customer ID
4. Worker will click “Delete Entry” button
5. Worker will click “Confirm Deletion” button

**DELETE FROM Customer**

**WHERE** **CustomerId = 5;**

*Entity: Reservation*

(25) Search a Reservation from Restaurant

1. Worker will click “Reservation” button
2. Worker will click “Search” button
3. Prompt worker to enter Reservation Id or customer name
4. List of possible Reservation(s) will populate

**SELECT \***

**FROM Reservation**

**WHERE ReservationId= 5;**

(26) Enter a new Reservation entry

1. Worker will click “Reservation” button
2. Worker will click “Add New Reservation” button
3. Prompt worker to input Reservation information
4. Worker will click “Save Entry” button.

**INSERT INTO Reservation**

**VALUES (ReservationId, ReserveDate, ReserveTime, NumOfPeople);**

(27) Update a Reservation entry

1. Worker will click “Reservation” button
2. Worker will click “Edit Reservation” button
3. Prompt worker to enter Reservation ID
4. Prompt worker to select reservation information to make changes to
5. Worker will click “Save Entry” button

**UPDATE FROM Reservation**

**SET ReserveDate = ‘’, ReserveTime = ‘’, NumOfPeople = ‘’**

**WHERE ReservationId = 5;**

(28) Delete a Reservation entry

1. Worker will click “Reservation” button
2. Worker will click “Delete Reservation” button
3. Prompt worker to enter Reservation ID
4. Worker will click “Delete Entry” button
5. Worker will click “Confirm Deletion” button

**DELETE FROM Reservation**

**WHERE ReservationId = 5;**

**Actor: Manager & Worker**

*Entity: Order*

(29) Search an Order from Restaurant

1. Actor will click “Order” button
2. Actor will click “Search” button
3. Prompt actor to enter customer ID or Order ID
4. List of possible Order(s) will populate

**SELECT \***

**FROM Order**

**WHERE OrderId= 5;**

(30) Enter a new Order entry

1. Actor will click “Order” button
2. Actor will click “Add New Order” button
3. Prompt actor to input order information
4. Actor will click “Save Entry” button.

**INSERT INTO Order**

**VALUES (OrderId, CustomerId, OrderDate, OrderTime, BalanceDue, Gratuity, Discounts, Total, PointsEarned);**

(31) Update an Order entry

1. Actor will click “Order” button
2. Actor will click “Edit Order” button
3. Prompt actor to enter Order ID
4. Prompt actor to select order information to make changes to
5. Actor will click “Save Entry” button

**UPDATE FROM Order**

**SET CustomerId, OrderDate, OrderTime, BalanceDue, Gratuity, Discounts, Total, PointsEarned**

**WHERE OrderId = 5;**

(32) Delete an Order entry

1. Actor will click “Order” button
2. Actor will click “Delete Order” button
3. Prompt actor to enter Order ID
4. Actor will click “Delete Entry” button
5. Actor will click “Confirm Deletion” button

**DELETE FROM Order**

**WHERE OrderId = 5;**

*Entity: Menu\_Order*

(33) Perform aggregate query for  Menu\_order

1. Manager will click “Query” button
2. Prompt manager to enter Menu\_id or Order\_id
3. List of possible menus will populate
4. Manager will select menu\_order if found

**SELECT \***

**FROM Menu\_Order**

**WHERE OrderId= 5;**

(34) Enter a new Menu\_Order entry

1. Manager will click “Add New Menu Order” button
2. Manager will add : Menu id, Order id, and Quantity for a new menu order
3. Manager will click “Save Entry” button.

**INSERT INTO Menu\_Order**

**VALUES** **(MenuID, OrderID, Quantity);**

(35) Update a Menu order entry

1. Manager will click “Edit Menu\_order” button
2. Prompt manager to update: Menu id, Order id, and Quantity for an existing menu order
3. Display updated menu\_order information.
4. Manager will click “Save Entry” button

**UPDATE FROM Menu\_Order**

**SET MenuID = ‘’, OrderID = ‘’, Quantity = ‘’**

(36) Delete a Menu order entry

1. Manager will click “Delete Menu” button
2. Prompt manager to enter Menu id / Order id and chose what he wishes to delete.
3. Manager will click “Delete Entry” button
4. Manager will click “Confirm Deletion” button

**DELETE FROM Menu\_Order**

**WHERE MenuID = ‘’;**

*Entity: Services*

(37) Search a Services from Restaurant

1. Worker will click “Services” button
2. Worker will click “Search” button
3. Prompt worker to enter Service ID or customer name
4. List of possible Service(s) will populate

**SELECT \***

**FROM Service**

**WHERE ServiceId= 5;**

(38) Enter a new Service entry

1. Worker will click “Service” button
2. Worker will click “Add New Service” button
3. Prompt worker to input service information
4. Worker will click “Save Entry” button

**INSERT INTO Service**

**VALUES (ServiceId, TableId, CustomerId, ReservationId, DateTaken,TimeTaken, IsCompleted);**

(39) Update a Service entry

1. Worker will click “Service” button
2. Worker will click “Edit Service” button
3. Prompt worker to enter Service ID
4. Prompt worker to select service information to make changes to
5. Worker will click “Save Entry” button

**UPDATE FROM Service**

**SET ServiceId, TableId, CustomerId, ReservationId, DateTaken,TimeTaken, IsCompleted**

**WHERE ServiceId = 5;**

(40) Delete a Service entry

1. Worker will click “Service” button
2. Worker will click “Delete Service” button
3. Prompt worker to enter Service ID
4. Worker will click “Delete Entry” button
5. Worker will click “Confirm Deletion” button

**DELETE FROM Service**

**WHERE** **ServiceId = 5;**

**Relationship Use Cases**

*Relationship: Menu –Contains-> Reward*

(41) Show the reward name in respect to the menu name

1. Worker will click “Menu” button
2. Worker will toggle “Contains Reward” button
3. Prompt worker to enter MenuId or show all
4. List of menu and associated rewards will populate

**SELECT m.menuName, r.rewardName**

**FROM Menu m**

**JOIN Reward r ON m.MenuId = r.MenuId;**

*Relationship: Menu –Sends-> Menu\_Order*

(42) Display all details of the menu that is placed onto an order

1. Worker will click “Order” button
2. Worker will click “Show Ordered Menu Item” button
3. Prompt worker to enter OrderId
4. List of MenuId associated with OrderId will populate

**SELECT m.\***

**FROM Menu m**

**JOIN Menu\_Order mo ON m.MenuId = mo.MenuId;**

**WHERE mo.OrderId = 3;**

*Relationship: Reward –Provides-> Redemption*

(43) Display the reward and menu name associated with the reward that is redeemed in redemption

1. Worker will click “Redemption” button
2. Worker will click “Show Reward Details” button
3. Prompt worker to enter RedemptionId
4. List of reward details will populated for the associated redemption

**SELECT r.rewardName, m.menuName, d.\***

**FROM Redemption d**

**JOIN Reward r ON d.RewardId = r.RewardId**

**JOIN Menu m ON m.MenuId = r.MenuId**

**WHERE d.RedemptionId = 4;**

*Relationship: Customer –Has-> Redemption*

(44) Display all rewards that a Customer has redeemed

1. Worker will click “Customer” button
2. Worker will click “Show Redeemed Rewards” button
3. Prompt worker to enter CustomerId
4. List of redemption details will be populated for the associated customer

**SELECT c.CustomerId, c.CustomerFirstName, c.CustomerLastName, r.\***

**FROM Customer c**

**JOIN Redemption r ON c.CustomerId = r.CustomerId**

**WHERE c.CustomerId = 5;**

*Relationship: Customer –Submits-> Order*

(45) Display all details for an order that a Customer has placed

1. Worker will click “Customer” button
2. Worker will click “Show Order History” button
3. Prompt worker to enter CustomerId and OrderId (or all OrderId)
4. List of order details will be populated for the associated customer

**SELECT c.CustomerId, c.CustomerFirstName, c.CustomerLastName, o.\***

**FROM Customer c**

**JOIN Order o ON c.CustomerId = o.CustomerId**

**WHERE c.CustomerId = 6;**

*Relationship: Customer –Receives-> Service*

(46) Display details of the customer and the table and reservation that they made

1. Worker will click “Customer” button
2. Prompt worker to enter CustomerId
3. Worker will click “Show Services” button
4. List of service details will be populated for the associated customer

**SELECT c.CustomerId, c.CustomerFirstName, c.CustomerLastName, s.TableId, s.ReservationId**

**FROM Customer c**

**JOIN Service s ON c.CustomerId = s.CustomerId**

**WHERE c.CustomerId = 6;**

*Relationship: Order –Sends-> Menu\_Order*

(47) Show the quantity of a menu item purchased by a customer

1. Worker will click “Order” button
2. Prompt worker to enter CustomerId
3. Worker will click “Show order details” button
4. List of menu details will be populated for the associated order from customer

**SELECT o.orderId, o.OrderDate, o.OrderTime, mo.MenuId, mo.Quantity, c.CustomerFirstName, c.CustomerLastName**

**FROM Order o**

**JOIN Menu\_Order mo ON o.OrderId = mo.OrderId**

**JOIN Customer c ON o.CustomerId = c.CustomerId**

**WHERE c.CustomerId = 7;**

*Relationship: Reservation –Has-> Services*

(48) Show all details of Reservation and Services

1. Worker will click “Service” button
2. Prompt worker to enter CustomerId
3. Worker will click “Show Reservation Details” button
4. List of reservation details will be populated for the associated service for a customer

**SELECT s.\*, r.\***

**FROM Service s**

**JOIN Reservation r ON s.ReservationId = r.ReservationId**

**WHERE s.CustomerId = 8;**

*Relationship: TableServ –Offers-> Services*

(49) Show details of table for a service and who managed that table

1. Worker will click “Service” button
2. Prompt worker to enter CustomerId
3. Worker will click “Show Table details” button
4. List of table details will be populated for the associated service for a customer

**SELECT s.ServiceId, c.CustomerFirstName, c.CustomerLastName, t.\*, e.EmployeeFirstName, e.EmployeeLastName**

**FROM Service s**

**JOIN Customer c ON s.CustomerId = c.CustomerId**

**JOIN TableServ t ON s.TableId = t.TableId**

**JOIN Employee e ON t.EmployeeId = e.EmployeeId**

**WHERE s.CustomerId = 9;**

*Relationship: Employee –Handles-> TableServ*

(50) Show employee names and all the table they handle

1. Worker will click “Employee” button
2. Prompt worker to enter CustomerId
3. Worker will click “Show Managed Tables” button
4. List of table details will be populated for the associated employee

**SELECT e.EmployeeId , e.EmployeeFirstName, e.EmployeeLastName, t.\***

**FROM Employee e**

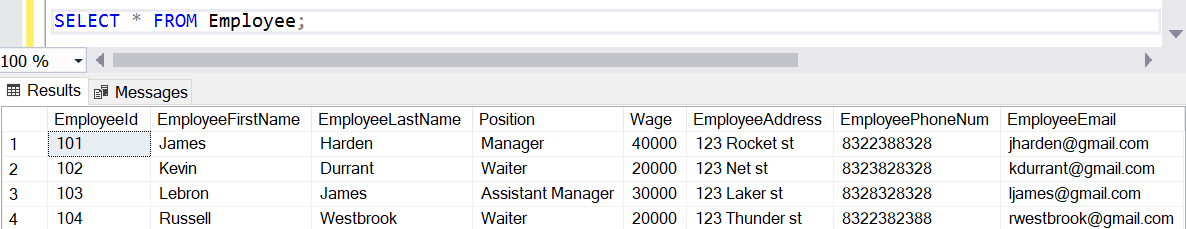
**JOIN TableServ t ON e.EmployeeId = t.EmployeeId**

**WHERE e.EmployeeId = 10;**

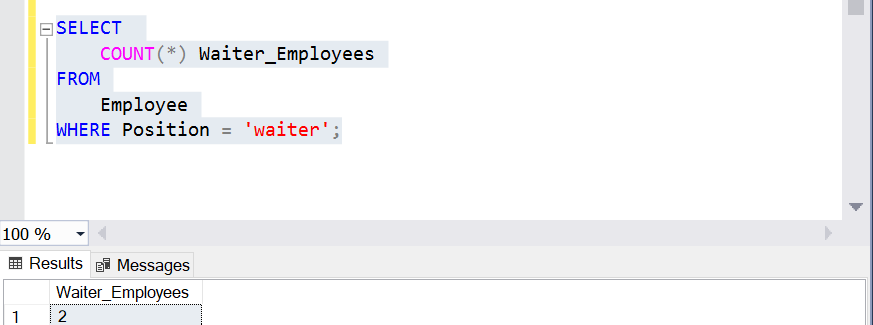
**Test Plan and Records**

**Employee Table**

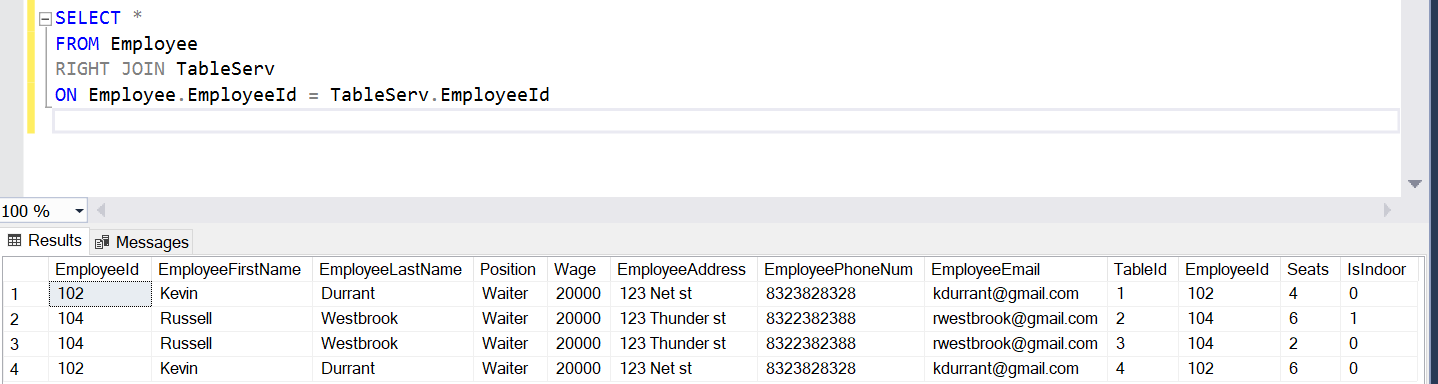
All Data:

****

Aggregate Function:

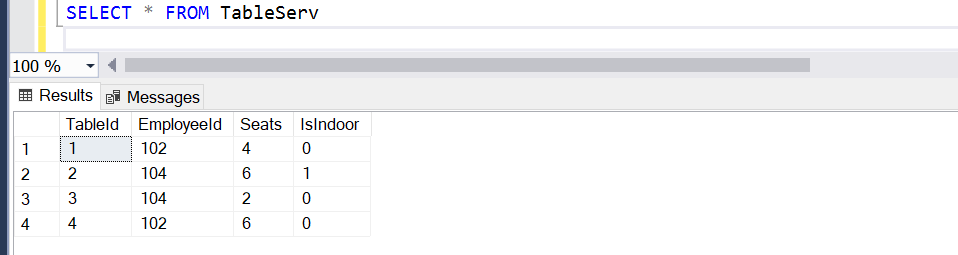


Joint Query with TableServ Entity:

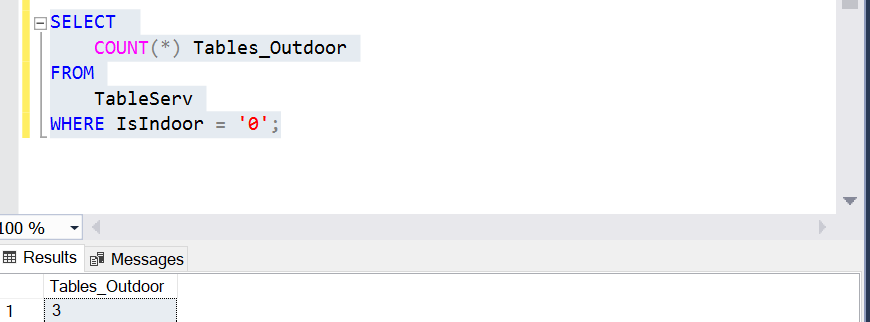


**TableServ Table**

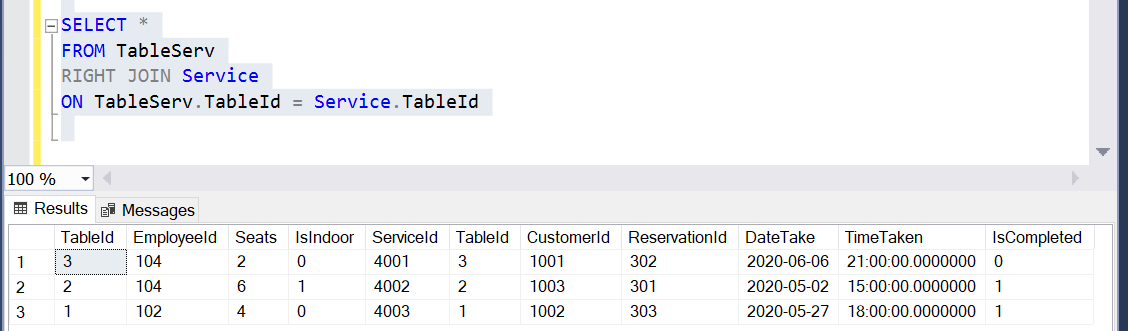
All Data:

****

Aggregate Function:

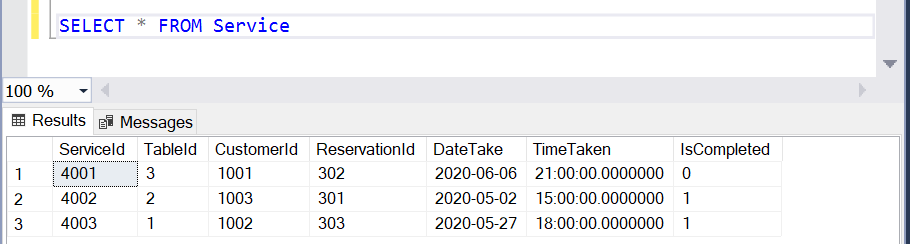


Joint Query with Service Entity:

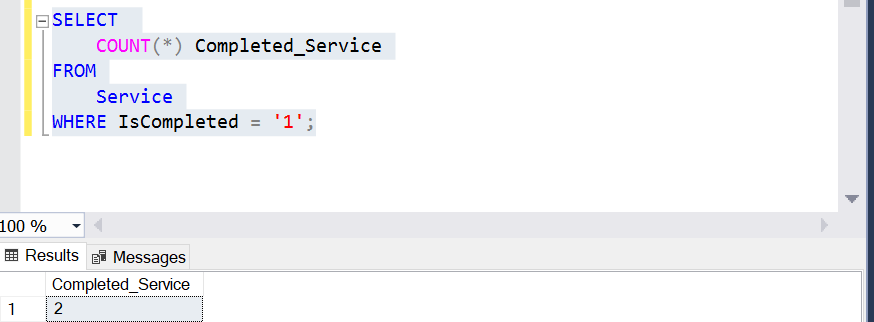


**Service Table**

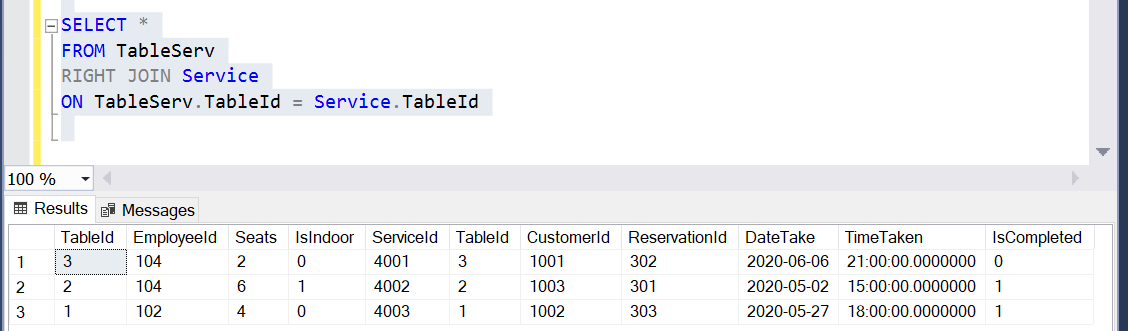
All Data:

****

Aggregate Function:

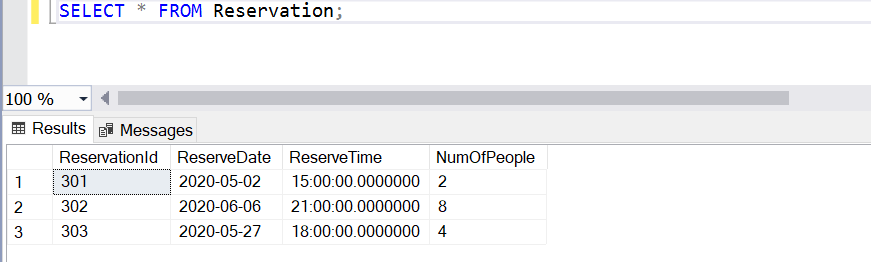
****

Joint Query with TableServ Entity:

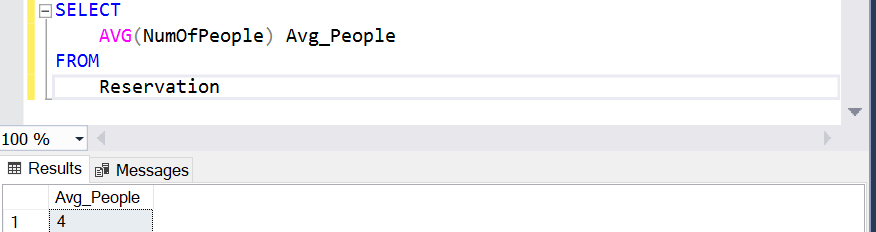


**Reservation Table**

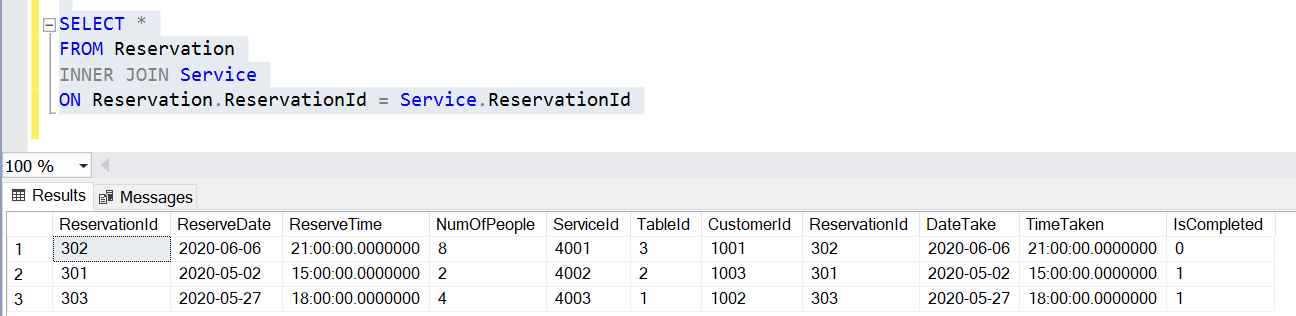
All Data:

****

Aggregate Function:

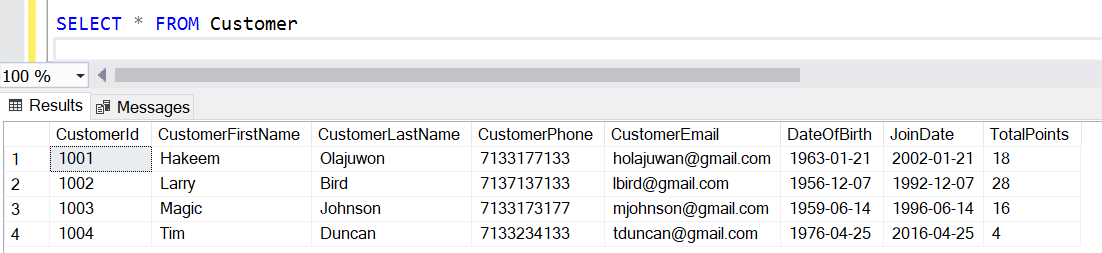


Joint Query with Service Entity:

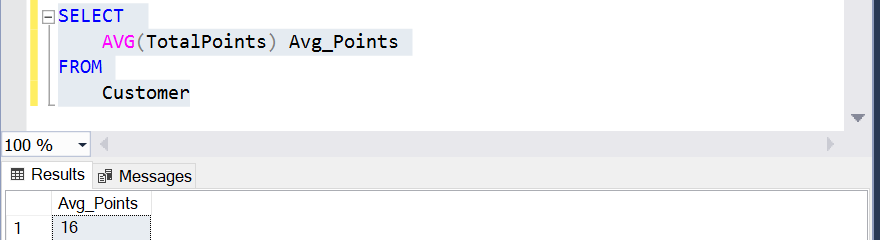


**Customer Table**

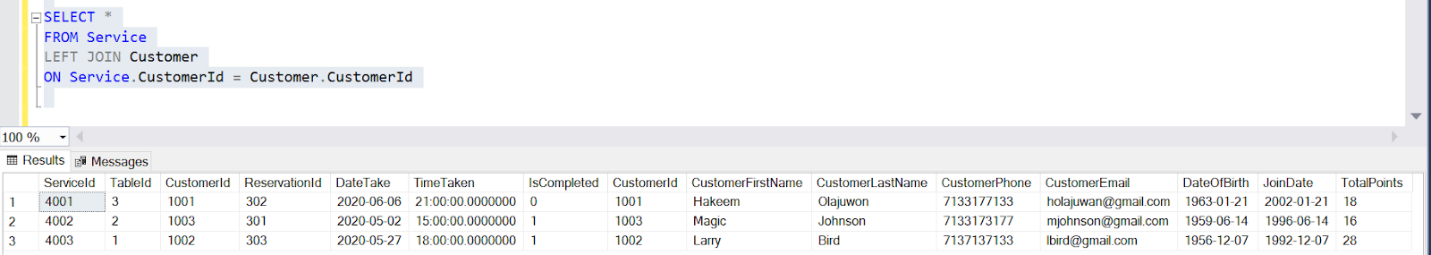
All Data:

****

Aggregate Function:

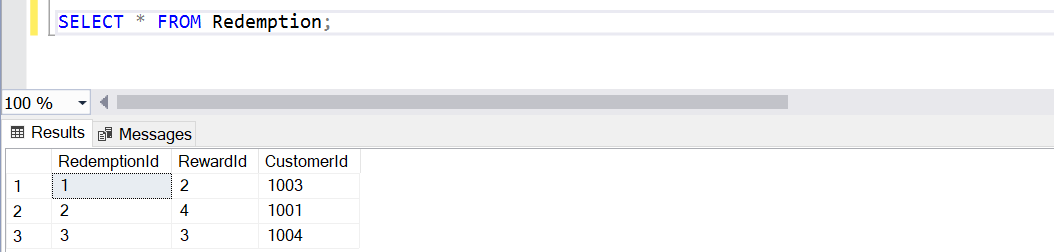


Joint Query with Service Entity:

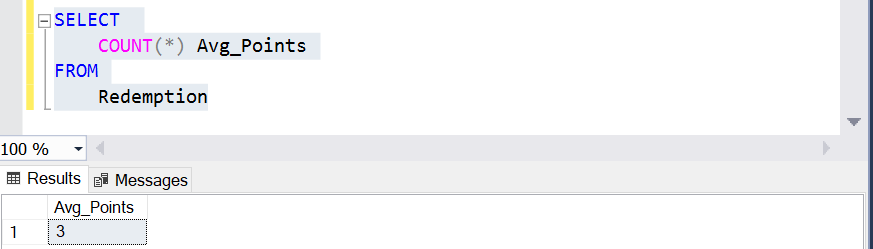


**Redemption Table**

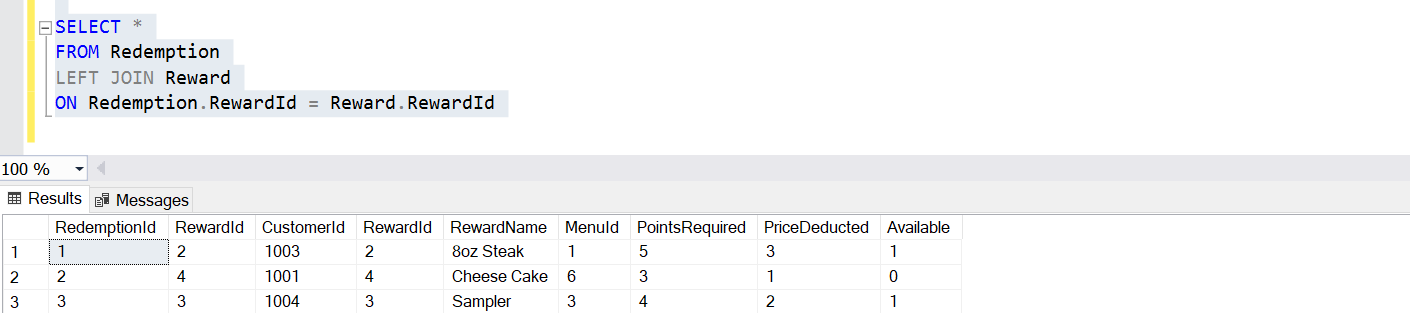
All Data:

****

Aggregate Function:

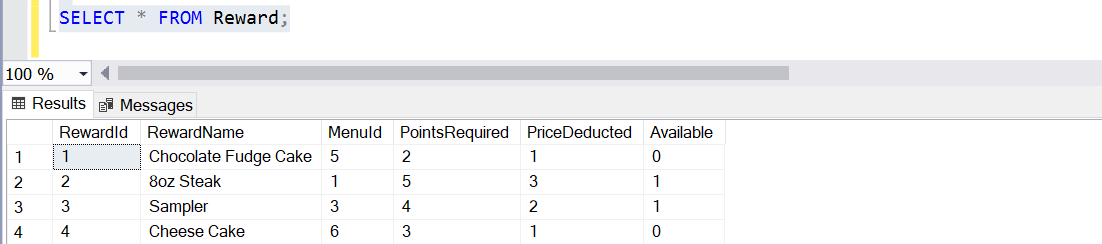
****

Joint Query with Reward Entity:

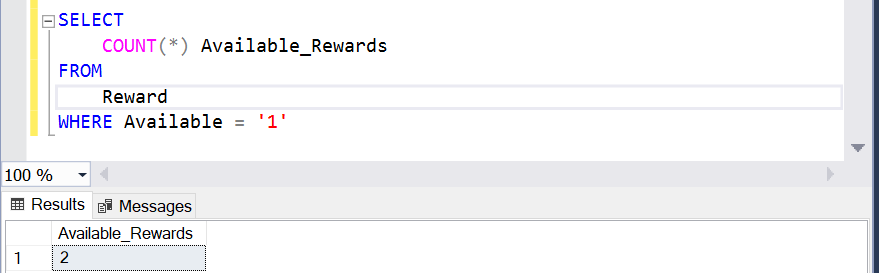


**Reward Table**

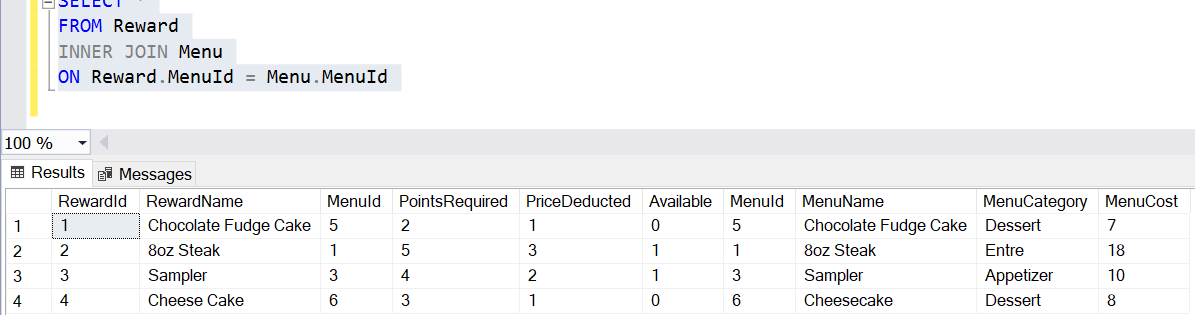
All Data:

****

Aggregate Function:

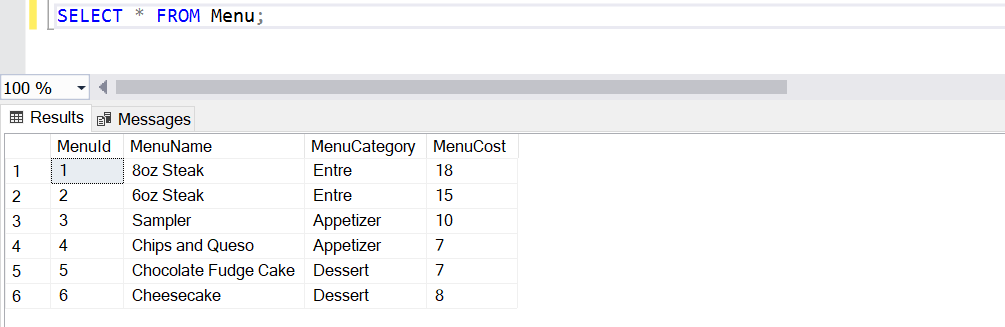
****

Joint Query with Menu Entity:

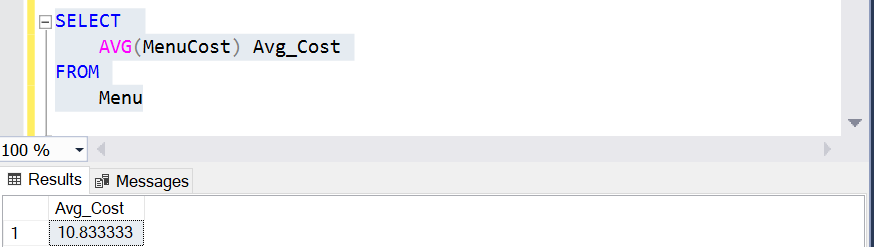


**Menu Table**

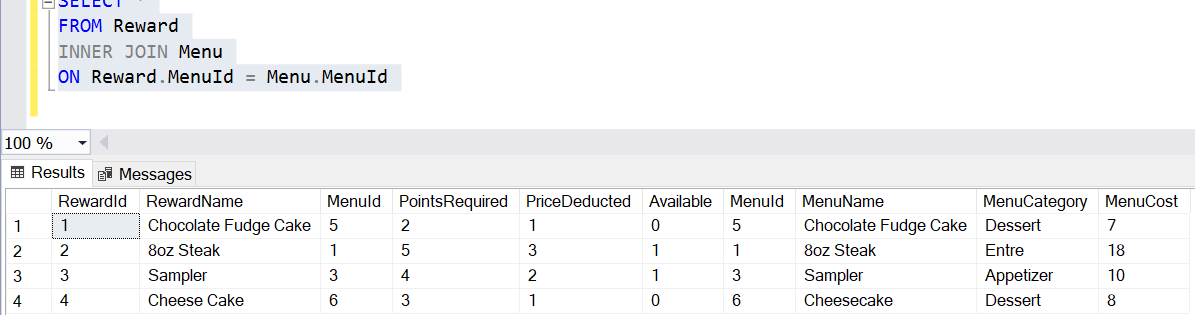
All Data:

****

Aggregate Function:

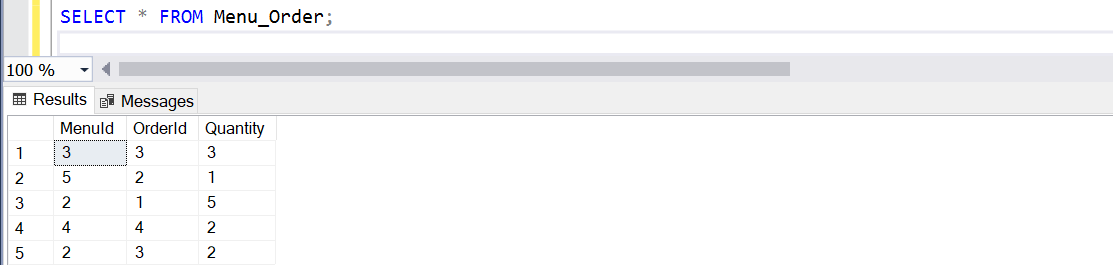


Joint Query with Reward Entity:

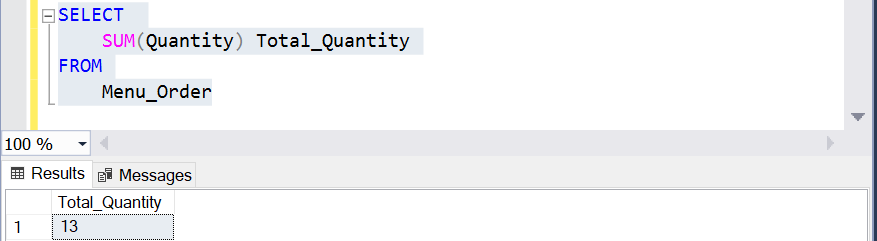


**Menu\_Order Table**

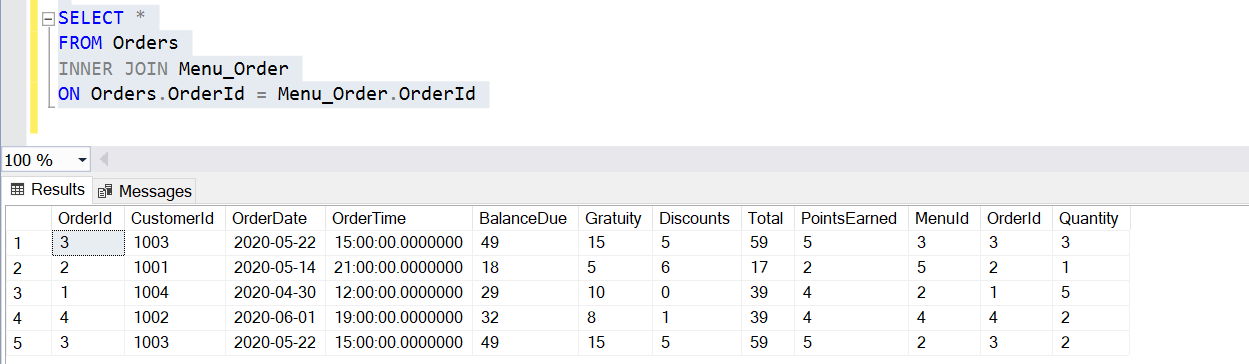
All Data:

****

Aggregate Function:

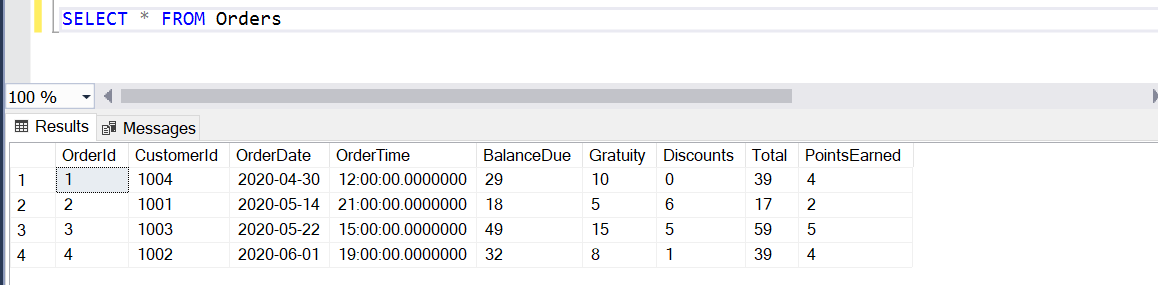


Joint Query with Orders Entity:

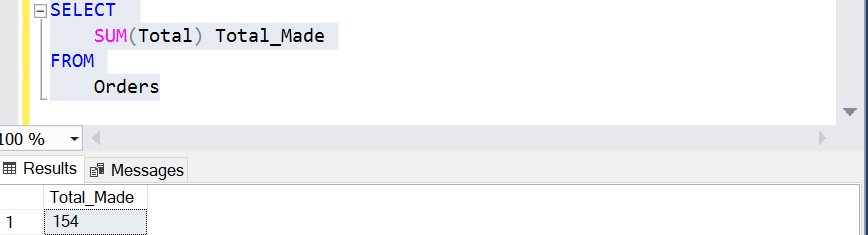


**Orders Table**

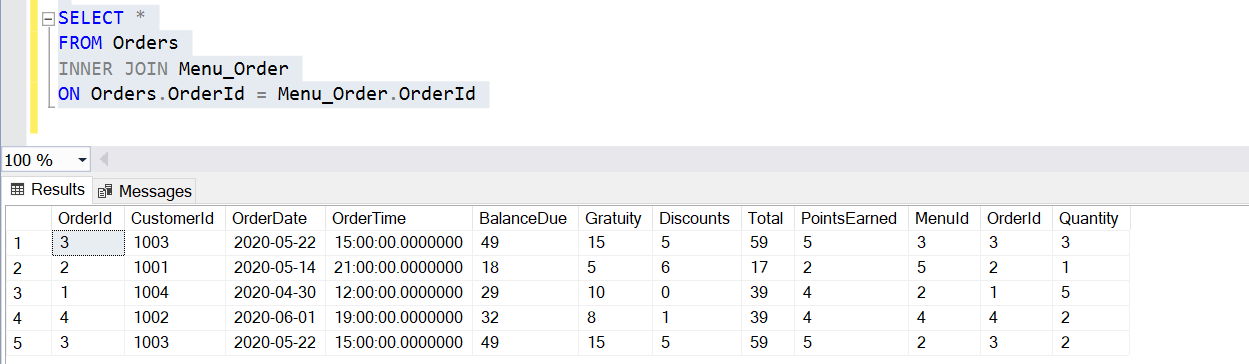
All Data:

****

Aggregate Function:



Joint Query with Menu\_Order Entity:



**Conclusion**

In conclusion, creating a database requires careful planning so that we can provide the right data to the right people with the right access. Data must connect in some ways to make logical sense so that update anomalies would be less of a hassle for the users and data integrity is kept at its best. It was in our best effort to avoid redundancy, yet the database may have been better if we have combined Order and Service tables, but we thought that would be too much information under one entity. Our focus was to provide information to both customers and workers at a restaurant which includes loyalty program for the customer and information about the restaurant orders and menus for the workers. It could have been extended and include supply inventory tables, but it may overcomplicate the database whose original intended purpose is designed for customers. All in all, the database functions to ease the users in looking for information for a particular restaurant that uses this loyalty program and it helps keep track of records that users could easily pull up any time.

**References**

Connolly, T. M., Begg, C. E., & Wesley, A. (2015). *Database systems: A practical approach to design, implementation, and management*. Boston: Pearson Education.

SQL Tutorial (n.d.). Retrieved March 24, 2020, from https://www.w3schools.com/sql/default.asp

What is Normalization? 1NF, 2NF, 3NF & BCNF with Examples. (n.d.). Retrieved from https://www.guru99.com/database-normalization.html