Martyna Olszewska, Szymon Paszkiewicz, Paweł Gorgolewski

Podstawy Baz Danych

Projekt:
System wspomagania działalności firmy świadczącej usługi Gastronomiczne

2021/2022

1. Wstęp

Celem projektu było zaplanowanie i implementacja systemu bazodanowego dla firmy świadczącej usługi gastronomiczne dla klientów indywidualnych oraz firm.

Ogólne informacje

W ofercie firmy jest żywność oraz napoje bezalkoholowe. Usługi świadczone są na miejscu oraz na wynos. Zamówienie na wynos może być zlecone na miejscu lub z wyprzedzeniem.. Firma dysponuje ograniczoną liczbą stolików. Istnieje możliwość wcześniejszej rezerwacji stolika dla co najmniej dwóch osób. Klientami są osoby indywidualne oraz firmy. Istnieje możliwość wystawienia faktury dla danego zamówienia lub faktury zbiorczej raz na miesiąc.

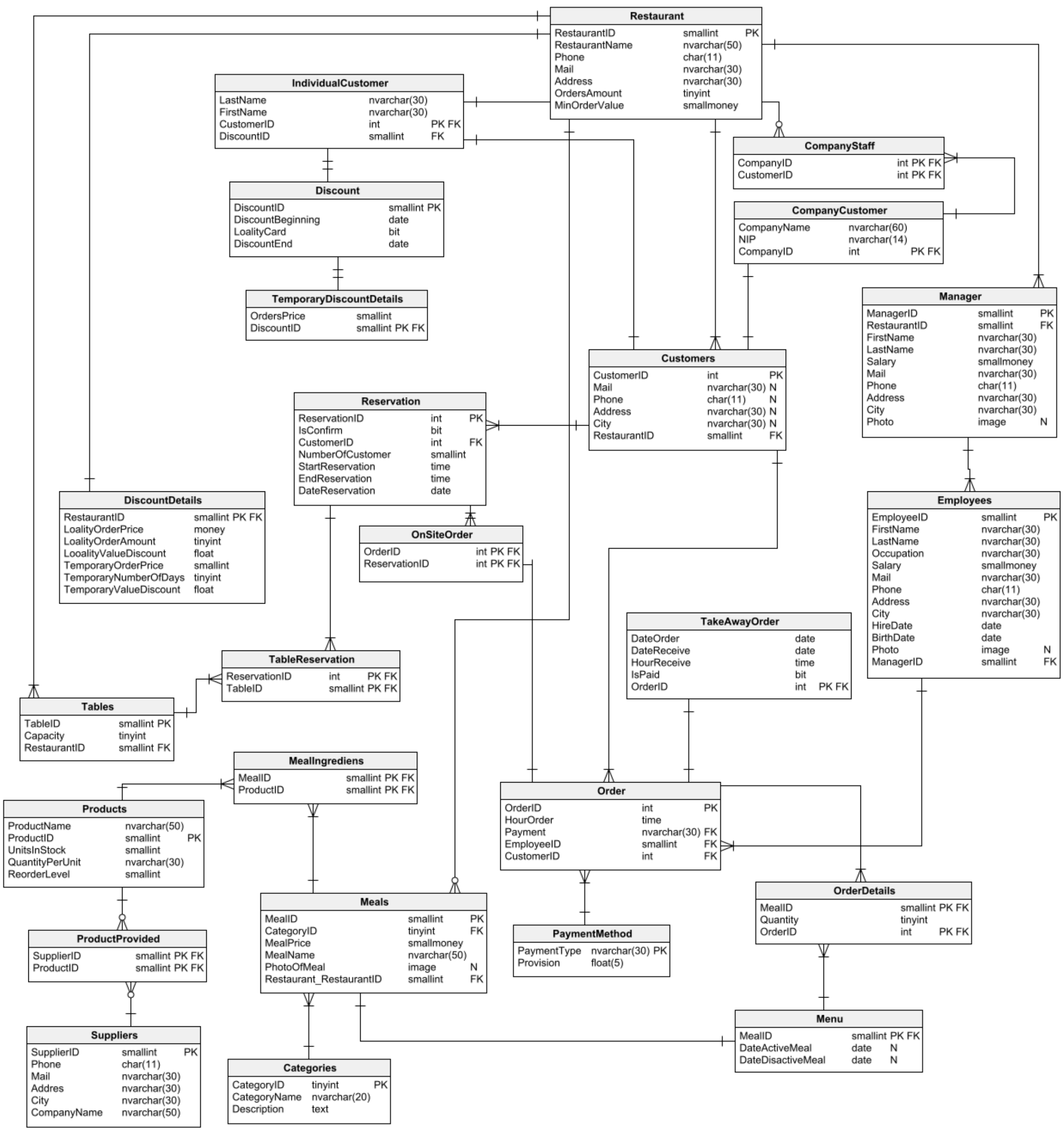
Menu ustalane jest co najmniej dziennym wyprzedzeniem. W firmie panuje zasada, że co najmniej połowa pozycji menu zmieniana jest co najmniej raz na dwa tygodnie. W dniach czwartek-piątek-sobota istnieje możliwość wcześniejszego zamówienia dań zawierających owoce morza. Z uwagi na indywidualny import takie zamówienie powinno być złożone maksymalnie do poniedziałku poprzedzającego zamówienie.

Internetowy formularz umożliwia klientowi indywidualnemu rezerwację stolika, przy jednoczesnym złożeniu zamówienia, z opcją płatności przed lub po zamówieniu, przy minimalnej wartości zamówienia , w przypadku klientów, którzy dokonali wcześniej co najmniej k zamówień . Internetowy formularz umożliwia także rezerwację stolików dla firm, w dwóch opcjach: rezerwacji stolików na firmę i/lub rezerwację stolików dla konkretnych pracowników firmy.

System umożliwia realizację programów rabatowych dla klientów indywidualnych przy spełnieniu określonych warunków dotyczących całkowitej ilości zamówień i wydanej kwoty. Klient może dostać jednorazowy rabat do wykorzystania w określonym czasie lub rabat bezterminowy, czyli tak zwaną kartę stałego klienta.

System umożliwia generowanie raportów miesięcznych i tygodniowych, dotyczących rezerwacji stolików, rabatów, menu, a także statystyk zamówienia oraz generowanie raportów dotyczących zamówień i rabatów dla klienta indywidualnego oraz firm.

2. Schemat Bazy Danych



3. Opisy Tabel

1. Restaurant

Tabela zawiera informacje o restauracjach, które mogą świadczyć usługi gastronomiczne. Posiada ona klucz główny *RestaurantID,* nazwę restauracji *RestaurantName, OrdersAmount* liczba zamówień jakie klient musi spełnić aby móc zrobić zamówienie przy rezerwacji i *MinOrderValue* minimalny koszt zamówień, które będą liczone do dostania możliwości zrobienia zamówienia przy rezerwacji przez klienta, numer telefonu *PhoneNumber*, adres email *Mail,* adres oraz miasto *Address* i *City.*

Warunki integralności:

1.RestaurantID jest unikalne
2.Phone w formacie + xx xxx-xxx-xxx, gdzie x to [0-15]
3.Mail zawiera znaki '@' i '.' oraz jest unikalny

```
CREATE TABLE [dbo].[Restaurant](
    [RestaurantID] [smallint] IDENTITY(1,1) NOT NULL,
    [RestaurantName] [nvarchar](50) NOT NULL,
    [Phone] [char](15) NOT NULL,
    [Mail] [nvarchar](30) NOT NULL,
    [Address] [nvarchar](30) NOT NULL,
    [OrdersAmount] [tinyint] NOT NULL,
    [MinOrderValue] SMALLMONEY NOT NULL,
 CONSTRAINT [RestaurantID] PRIMARY KEY CLUSTERED
(
    [RestaurantID] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Restaurant]  WITH CHECK ADD CHECK  (([Phone] like
'+[0-9][0-9] [0-9][0-9][0-9]-[0-9][0-9][0-9]-[0-9][0-9][0-9]'))
GO

ALTER TABLE [dbo].[Restaurant]  WITH CHECK ADD CHECK  (([Mail] like
'%@%.%'))
GO
```

2. IndividualCustomer

Tabela zawiera informacje o indywidualnych klientach. Kluczem głównym jest *CustomerID,* klucz obcy do tabeli Discount *DiscountID.* Posiada ona informacje o imieniu klienta *FirstName,* oraz jego nazwisku *LastName.*

Warunki integralności:

1.CustomerID i DiscountID są parą unikalną

```
CREATE TABLE [dbo].[IndividualCustomer](
    [LastName] [nvarchar](30) NOT NULL,
    [FirstName] [nvarchar](30) NOT NULL,
    [CustomerID] [int] NOT NULL,
    [DiscountID] [smallint] NOT NULL,
 CONSTRAINT [IndividualCustomer_pk] PRIMARY KEY CLUSTERED
(
    [CustomerID] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

3. Discount

Tabela zawiera informacje o rabatach dla klientów indywidualnych. Posiada ona klucz główny *DiscountID*, informacje o tym kiedy czasowa zniżka się rozpoczęła *DiscountBeginning,* czy klient posiada kartę lojalnościową *LoalityCard* oraz kiedy zniżka czasowa się kończy *DiscountEnd.*

Warunki integralności:

1.DiscountID jest unikalne

2.DiscountEnd nie może być wcześniejsze niż DiscountBeginning

3.LoalityCard jest 0 lub 1

```
CREATE TABLE [dbo].[Discount](
    [DiscountID] [smallint] IDENTITY(1,1) NOT NULL,
    [DiscountBeginning] [date] NOT NULL,
    [LoalityCard] [bit] NOT NULL,
    [DiscountEnd] [date] NOT NULL,
 CONSTRAINT [Discount_pk] PRIMARY KEY CLUSTERED
(
    [DiscountID] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

```
ALTER TABLE [dbo].[Discount] ADD  DEFAULT ((0)) FOR [LoalityCard]
GO

ALTER TABLE [dbo].[Discount]  WITH CHECK ADD  CONSTRAINT [check_Date]
CHECK  (([DiscountBeginning]<[DiscountEnd]))
GO

ALTER TABLE [dbo].[Discount] CHECK CONSTRAINT [check_Date]
GO

ALTER TABLE [dbo].[Discount]  WITH CHECK ADD CHECK  (([LoalityCard]=(0)
OR [LoalityCard]=(1)))
GO
```

4. TemporaryDiscountDetails

Tabela zawiera informacje o stanie dostępu do zniżki tymczasowej dla klienta indywidualnego. Posiada klucz główny *DiscountID* oraz informacje o sumie wydanej podczas wizyt w restauracji *OrdersPrice.*

Warunki integralności:

1.OrdersPrice nie może być mniejsze od 0

2.DiscoundID jest unikalne

```
CREATE TABLE [dbo].[TemporaryDiscountDetails](
    [OrdersPrice] [smallint] NOT NULL,
    [DiscountID] [smallint] NOT NULL,
 CONSTRAINT [TemporaryDiscountDetails_pk] PRIMARY KEY CLUSTERED
(
    [DiscountID] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[TemporaryDiscountDetails] ADD  DEFAULT ((0)) FOR
[OrdersPrice]
GO

ALTER TABLE [dbo].[TemporaryDiscountDetails]  WITH CHECK ADD CHECK
(([OrdersPrice]>=(0)))
GO
```

5. DiscountDetails

Tabela posiada informacje o tym jakie warunki trzeba spełniać, aby móc otrzymać rabat oraz szczegóły dotyczące rabatów. Posiada klucz główny, którym jest *RestaurantID*. Zawiera także informacje o kwocie którą trzeba wydać, aby otrzymać rabat tymczasowy *TemporaryOrdersPrice,* ile dni taki rabat jest ważny *TemporaryNumberOfDays* oraz wartość takiego rabatu *TemporaryValueDiscount,* wartość rabatu dla karty lojalnościowej *LoalityValueDiscount,* ilość zamówień, które trzeba zrealizować i za jaką kwotę, aby otrzymać taką kartę *LoalityOrderAmount, LoalityOrderPrice.*

Warunki integralności:

1.TemporaryValueDiscount jest z przedziału [0,1]
2.LoalityValueDiscount jest z przedziału [0,1]
3.TemporaryNumberOfDays jest liczbą większą od 0
4.TemporaryOrderPrice jest liczbą większą od 0
5.LoalityOrderPrice jest liczbą większą od 0
6.LoalityOrderAmount jest liczbą większą od 0

```
CREATE TABLE [dbo].[DiscountDetails](
    [RestaurantID] [smallint] NOT NULL,
    [LoalityOrderPrice] [money] NOT NULL,
    [LoalityOrderAmount] [tinyint] NOT NULL,
    [LooalityValueDiscount] [float] NOT NULL,
    [TemporaryOrderPrice] [smallint] NOT NULL,
    [TemporaryNumberOfDays] [tinyint] NOT NULL,
    [TemporaryValueDiscount] [float] NOT NULL,
 CONSTRAINT [DiscountDetails_pk] PRIMARY KEY CLUSTERED
(
    [RestaurantID] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[DiscountDetails]  WITH CHECK ADD CHECK
(([LoalityOrderPrice]>(0)))
GO

ALTER TABLE [dbo].[DiscountDetails]  WITH CHECK ADD CHECK
(([LoalityOrderAmount]>(0)))
GO
```

```
ALTER TABLE [dbo].[DiscountDetails]  WITH CHECK ADD CHECK
(([LooalityValueDiscount]>=(0) AND [LooalityValueDiscount]<=(1)))
GO

ALTER TABLE [dbo].[DiscountDetails]  WITH CHECK ADD CHECK
(([TemporaryOrderPrice]>(0)))
GO

ALTER TABLE [dbo].[DiscountDetails]  WITH CHECK ADD CHECK
(([TemporaryNumberOfDays]>(0)))
GO

ALTER TABLE [dbo].[DiscountDetails]  WITH CHECK ADD CHECK
(([TemporaryValueDiscount]>=(0) AND [TemporaryValueDiscount]<=(1)))
GO
```

6. Tables

Tabela zawiera informacje o stolikach dostępnych w restauracji. Zawiera ona klucz główny *TableID,* klucz obcy do tabeli *RestaurantID* oraz informacje o ilości miejsc danego stolika *Capacity.*

Warunki integralności:

1.TableID jest unikalne

2.Capacity jest liczbą większą od 0

```
CREATE TABLE [dbo].[Tables](
    [TableID] [smallint] IDENTITY(1,1) NOT NULL,
    [Capacity] [tinyint] NOT NULL,
    [RestaurantID] [smallint] NOT NULL,
 CONSTRAINT [Tables_pk] PRIMARY KEY CLUSTERED
(
    [TableID] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Tables]  WITH CHECK ADD CHECK  (([Capacity]>(0)))
GO
```

7. TableReservation

Tabela posiada dwa klucze główne *ReservationID* i *TableID.*

Warunki integralności:

1.TableID i ReservationID są unikalną parą

```
CREATE TABLE [dbo].[TableReservation](
    [ReservationID] [int] NOT NULL,
    [TableID] [smallint] NOT NULL,
 CONSTRAINT [TableReservation_pk] PRIMARY KEY CLUSTERED
(
    [ReservationID] ASC,
    [TableID] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

8. Reservation

Tabela zawiera informacje o szczegółach rezerwacji dokonywanych przez klientów. Posiada klucz główny *ReservationID* i klucz obcy do tabeli Customers *CustomerID.* Informacje o ilości gości rezerwacji *NumberOfCustomer,* dacie rezerwacji *DateReservation,* godzinie rozpoczęcia i zakończenia *StartReservation, EndReservation* oraz o tym czy rezerwacja jest już zatwierdzona przez pracownika *IsConfirm.*

Warunki integralności:

1.ResevationID jest unikalne
2.StartReservation jest mniejsze od EndReservation
3.IsConfirm jest 0 lub 1

```
CREATE TABLE [dbo].[Reservation](
    [ReservationID] [int] IDENTITY(1,1) NOT NULL,
    [IsConfirm] [bit] NOT NULL,
    [CustomerID] [int] NOT NULL,
    [NumberOfCustomer] [smallint] NOT NULL,
    [StartReservation] [time](7) NOT NULL,
    [EndReservation] [time](7) NOT NULL,
    [DateReservation] [date] NOT NULL,
 CONSTRAINT [Reservation_pk] PRIMARY KEY CLUSTERED
(
    [ReservationID] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
```

```
GO

ALTER TABLE [dbo].[Reservation] ADD  DEFAULT ((0)) FOR [IsConfirm]
GO

ALTER TABLE [dbo].[Reservation]  WITH CHECK ADD  CONSTRAINT [CheckTime]
CHECK  (([StartReservation]<[EndReservation]))
GO

ALTER TABLE [dbo].[Reservation] CHECK CONSTRAINT [CheckTime]
GO

ALTER TABLE [dbo].[Reservation]  WITH CHECK ADD CHECK  (([IsConfirm]=(1)
OR [IsConfirm]=(0)))
GO
```

9. OnSiteOrder

Tabela zawiera informacje o zamówieniach złożonych wewnątrz restauracji. Posiada dwa klucze główne *OrderID, ReservationID*.

Warunki integralności:

1.Para (OrderID, ReservationID) jest unikalna

```
CREATE TABLE [dbo].[OnSiteOrder](
    [OrderID] [int] NOT NULL,
    [ReservationID] [int] NOT NULL,
    [OrderDate] DATE,
 CONSTRAINT [OnSiteOrder_pk] PRIMARY KEY CLUSTERED
(
    [OrderID] ASC,
    [ReservationID] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

10. MealIngrediens

Tabela zawiera informacje o produktach potrzebnych do przygotowania konkretnego dania. Posiada dwa klucze główne *MealID, ProductID.*

Warunki integralności:

1.Para (MealID, ProductID) jest unikalna

```
CREATE TABLE [dbo].[MealIngrediens](
    [MealID] [smallint] NOT NULL,
    [ProductID] [smallint] NOT NULL,
 CONSTRAINT [MealIngrediens_pk] PRIMARY KEY CLUSTERED
(
    [MealID] ASC,
    [ProductID] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

11. Products

Tabela zawiera informacje o produktach używanych w restauracji. Posiada klucz główny *ProductID,* nazwę produktu *ProductName,* ilość w magazynie *UnitsInStock,* minimalnej ilości jaka musi być dostępna *ReorderLevel.*

Warunki integralności:

1. ProductID jest unikalne
2. UnitsInStock jest liczbą większą lub równą od 0
3. ReorderLevel jest liczbą większą lub równą od 0

```
CREATE TABLE [dbo].[Products](
    [ProductName] [nvarchar](50) NOT NULL,
    [ProductID] [smallint] IDENTITY(1,1) NOT NULL,
    [UnitsInStock] [smallint] NOT NULL,
    [QuantityPerUnit] [nvarchar](30) NOT NULL,
    [ReorderLevel] [smallint] NOT NULL,
 CONSTRAINT [Products_pk] PRIMARY KEY CLUSTERED
(
    [ProductID] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Products] ADD  DEFAULT ((0)) FOR [UnitsInStock]
GO

ALTER TABLE [dbo].[Products] ADD  DEFAULT ((0)) FOR [ReorderLevel]
GO
```

```
ALTER TABLE [dbo].[Products]  WITH CHECK ADD CHECK
(([ReorderLevel]>=(0)))
GO

ALTER TABLE [dbo].[Products]  WITH CHECK ADD CHECK
(([UnitsInStock]>=(0)))
GO
```

12. ProductProvided

Tabela zawiera informacje o produktach i ich dostawcach. Posiada dwa klucze główne *SupplierID, ProductID.*

Warunki integralności:

1.Para (SupplierID, ProductID) jest unikalna

```
CREATE TABLE [dbo].[ProductProvided](
    [SupplierID] [smallint] NOT NULL,
    [ProductID] [smallint] NOT NULL,
 CONSTRAINT [ProductProvided_pk] PRIMARY KEY CLUSTERED
(
    [SupplierID] ASC,
    [ProductID] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

13. Suppliers

Tabela zawiera informacje o dostawcach. Posiada klucz główny *SupplierID,* numer telefonu *Phone, email Mail,* nazwę firmy *CompanyName* oraz dokładny adres i miasto *Address, City*.

Warunki integralności:

1.SupplierID jest unikalne
2.Phone w formacie + xx xxx-xxx-xxx, gdzie x to [0-15]
3.Mail zawiera '@' i ' . '

```
CREATE TABLE [dbo].[Suppliers](
    [SupplierID] [smallint] IDENTITY(1,1) NOT NULL,
    [Phone] [char](15) NOT NULL,
    [Mail] [nvarchar](30) NOT NULL,
    [Address] [nvarchar](30) NOT NULL,
    [City] [nvarchar](30) NOT NULL,
    [CompanyName] [nvarchar](50) NOT NULL,
 CONSTRAINT [Suppliers_pk] PRIMARY KEY CLUSTERED
(
    [SupplierID] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Suppliers]  WITH CHECK ADD CHECK  (([Mail] like
'%@%.%'))
GO

ALTER TABLE [dbo].[Suppliers]  WITH CHECK ADD CHECK  (([Phone] like
'+[0-9][0-9] [0-9][0-9][0-9]-[0-9][0-9][0-9]-[0-9][0-9][0-9]'))
GO
```

14. Categories

Tabela zawiera informacje o kategoriach posiłków serwowanych w restauracji. Posiada ona klucz główny *CategoryID,* nazwę kategorii *CategoryName* oraz jej opis *Description.*

Warunki integralności:

1.CategoryID jest unikalne

```
CREATE TABLE [dbo].[Categories](
    [CategoryID] [tinyint] IDENTITY(1,1) NOT NULL,
    [CategoryName] [nvarchar](20) NOT NULL,
    [Description] [text] NOT NULL,
 CONSTRAINT [Categories_pk] PRIMARY KEY CLUSTERED
(
    [CategoryID] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
```

15. Meals

Tabela zawiera informacje o daniach dostępnych w restauracji. Posiada klucz główny *MealID***,** klucze obce do tabel Restaurant i Categories *RestaurantID, CategoryID,* cene dania *MealPrice*, jego nazwę *MealName* oraz zdjęcie *PhotoOfMeal.*

Warunki integralności:

1.MealID jest unikalne

2.MealPrice jest liczbą większą lub równą od 0

```sql
CREATE TABLE [dbo].[Meals](
    [MealID] [smallint] IDENTITY(1,1) NOT NULL,
    [CategoryID] [tinyint] NOT NULL,
    [MealPrice] [smallmoney] NOT NULL,
    [MealName] [nvarchar](50) NOT NULL,
    [PhotoOfMeal] [image] NULL,
    [RestaurantID] [smallint] NOT NULL,
 CONSTRAINT [Meals_pk] PRIMARY KEY CLUSTERED
(
    [MealID] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO

ALTER TABLE [dbo].[Meals]  WITH CHECK ADD CHECK  (([MealPrice]>=(0)))
GO
```

16. Menu

Tabela zawiera informacje o czasie aktywności danego dania w menu. Posiada klucz główny *MealID* oraz daty aktywacji i dezaktywacji dania w menu *DateActivateMeal, DateDisactivateMeal.*

Warunki integralności:

1.MealID jest unikalne

2.DateActiveMeal jest mniejsze od DateDisactiveMeal

```sql
CREATE TABLE [dbo].[Menu](
    [MealID] [smallint] NOT NULL,
    [DateActiveMeal] [date] NULL,
    [DateDisactiveMeal] [date] NULL,
 CONSTRAINT [Menu_pk] PRIMARY KEY CLUSTERED
(
    [MealID] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

17. OrderDetails

Tabela zawiera informacje o szczegółach zamówień. Posiada dwa klucze główne *MealID, OrderID,* a także informacje o ilości która została zamówiona *Quantity.*

Warunki integralności:

1.Para (MealID, OrderID) jest unikalna

2.Quantity jest liczbą większą 0

```
CREATE TABLE [dbo].[OrderDetails](
    [MealID] [smallint] NOT NULL,
    [Quantity] [tinyint] NOT NULL,
    [OrderID] [int] NOT NULL,
 CONSTRAINT [OrderDetails_pk] PRIMARY KEY CLUSTERED
(
    [OrderID] ASC,
    [MealID] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[OrderDetails]  WITH CHECK ADD CHECK
(([Quantity]>(0)))
GO
```

18. PaymentMethod

Tabela zawiera informacje płatnościach. Posiada klucz główny *PaymentType* oraz informacje o prowizji *Provision.*

Warunki integralności:

1.PaymentType jest unikalne

2.Provison jest liczbą większą lub równą od 0

```
CREATE TABLE [dbo].[PaymentMethod](
    [PaymentType] [nvarchar](30) NOT NULL,
    [Provision] [real] NOT NULL,
 CONSTRAINT [PaymentMethod_pk] PRIMARY KEY CLUSTERED
(
    [PaymentType] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[PaymentMethod] ADD  DEFAULT ((0)) FOR [Provision]
GO
```

```
ALTER TABLE [dbo].[PaymentMethod]  WITH CHECK ADD CHECK
(([Provision]>=(0)))
GO
```

19. Manager

Tabela zawiera informacje o menadżerach restauracji. Posiada ona klucz główny *ManagerID* i klucz obcy do tabeli Restaurant *RestaurantID, imię i nazwisko menadżera FirstName LastName,* wysokość jego pensji *Salary.* Informacje kontaktowe takie jak adres email *Mail,* numer telefonu *Phone,* jego adres oraz miasto *Address, City,* a także zdjecie *Photo.*

Warunki integralności:

1. ManagerID jest unikalne
2. Phone w formacie + xx xxx-xxx-xxx, gdzie x to [0-15]
3. Mail zawiera '@' i ' . '
4. Salary jest liczbą większą od 0

```
CREATE TABLE [dbo].[Manager](
    [ManagerID] [smallint] IDENTITY(1,1) NOT NULL,
    [RestaurantID] [smallint] NOT NULL,
    [FirstName] [nvarchar](30) NOT NULL,
    [LastName] [nvarchar](30) NOT NULL,
    [Salary] [smallmoney] NOT NULL,
    [Mail] [nvarchar](30) NOT NULL,
    [Phone] [char](15) NOT NULL,
    [Address] [nvarchar](30) NOT NULL,
    [City] [nvarchar](30) NOT NULL,
    [Photo] [image] NULL,
 CONSTRAINT [Manager_pk] PRIMARY KEY CLUSTERED
(
    [ManagerID] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO

ALTER TABLE [dbo].[Manager]  WITH CHECK ADD CHECK  (([Mail] like
'%@%.%'))
GO

ALTER TABLE [dbo].[Manager]  WITH CHECK ADD CHECK  (([Phone] like
'+[0-9][0-9] [0-9][0-9][0-9]-[0-9][0-9][0-9]-[0-9][0-9][0-9]'))
GO
```

```
ALTER TABLE [dbo].[Manager]  WITH CHECK ADD CHECK  (([Salary]>(0)))
GO
```

20. Empolyees

Tabela zawiera informacje o pracownikach restauracji. Posiada ona klucz główny *EmployeeID* i klucz obcy do tabeli Manager *ManagerID* informujący o tym jakiemu menadżerowi podlega, *imię i nazwisko pracownika FirstName LastName,* wysokość jego pensji *Salary.* Informacje kontaktowe takie jak adres email *Mail,* numer telefonu *Phone,* jego adres oraz miasto *Address, City,* a także zdjecie *Photo.* Zawiera także datę urodzenia pracownika *BirthDate* i datę zatrudnienia *HireDate* oraz stanowisko na jakim pracuje *Occupation.*

Warunki integralności:

1. EmployeeID jest unikalne
2. Phone w formacie + xx xxx-xxx-xxx, gdzie x to [0-15]
3. Mail zawiera '@' i '.'
4. Salary jest liczbą większą od 0

```
CREATE TABLE [dbo].[Employees](
    [EmployeeID] [smallint] IDENTITY(1,1) NOT NULL,
    [FirstName] [nvarchar](30) NOT NULL,
    [LastName] [nvarchar](30) NOT NULL,
    [Occupation] [nvarchar](30) NOT NULL,
    [Salary] [smallmoney] NOT NULL,
    [Mail] [nvarchar](30) NOT NULL,
    [Phone] [char](15) NOT NULL,
    [Address] [nvarchar](30) NOT NULL,
    [City] [nvarchar](30) NOT NULL,
    [HireDate] [date] NOT NULL,
    [BirthDate] [date] NOT NULL,
    [Photo] [image] NULL,
    [ManagerID] [smallint] NOT NULL,
 CONSTRAINT [Employees_pk] PRIMARY KEY CLUSTERED
(
    [EmployeeID] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO

ALTER TABLE [dbo].[Employees]  WITH CHECK ADD CHECK  (([Mail] like
'%@%.%'))
GO
```

```
ALTER TABLE [dbo].[Employees]  WITH CHECK ADD CHECK  (([Phone] like
'+[0-9][0-9] [0-9][0-9][0-9]-[0-9][0-9][0-9]-[0-9][0-9][0-9]'))
GO

ALTER TABLE [dbo].[Employees]  WITH CHECK ADD CHECK  (([Salary]>(0)))
GO
```

21. Order

Tabela zawiera informacje o zamówieniach. Posiada klucz główny *OrderID* oraz trzy klucze obce do tabel PaymentMethod, Employees, Customers - *Payment, EmployeeID, CustomerID* oraz informacje o godzinie złożenia zamówienia *HourOrder, Finished* informuje czy zamówienie zostało już zakończone.

Warunki integralności:

1.OrderID jest unikalne

```
CREATE TABLE [dbo].[Order](
    [OrderID] [int] IDENTITY(1,1) NOT NULL,
    [HourOrder] [time](7) NOT NULL,
    [Payment] [nvarchar](30) NOT NULL,
    [EmployeeID] [smallint] NOT NULL,
    [CustomerID] [int] NOT NULL,
    [Finished] [bit] NOT NULL,

 CONSTRAINT [Order_pk] PRIMARY KEY CLUSTERED
(
    [OrderID] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Order] ADD  DEFAULT (CONVERT([time],getdate())) FOR
[HourOrder]
GO
```

22. CompanyStaff

Zawiera informacje o pracownikach firmy, którzy są też klientami indywidualnymi. Posiada ona dwa klucze główne *CompanyID* i *CustomerID.*

Warunki integralności:

1.Para CompanyID i CustomerID jest unikalna

```
CREATE TABLE [dbo].[CompanyStaff](
    [CompanyID] [int] NOT NULL,
    [CustomerID] [int] NOT NULL,
 CONSTRAINT [CompanyStaff_pk] PRIMARY KEY CLUSTERED
(
    [CompanyID] ASC,
    [CustomerID] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[CompanyStaff]  WITH CHECK ADD  CONSTRAINT
[CompanyStaff_CompanyCustomer] FOREIGN KEY([CompanyID])
REFERENCES [dbo].[CompanyCustomer] ([CompanyID])
GO

ALTER TABLE [dbo].[CompanyStaff] CHECK CONSTRAINT
[CompanyStaff_CompanyCustomer]
GO

ALTER TABLE [dbo].[CompanyStaff]  WITH CHECK ADD  CONSTRAINT
[CompanyStaff_IndividualCustomer] FOREIGN KEY([CustomerID])
REFERENCES [dbo].[IndividualCustomer] ([CustomerID])
GO

ALTER TABLE [dbo].[CompanyStaff] CHECK CONSTRAINT
[CompanyStaff_IndividualCustomer]
GO
```

23. TakeAwayOrder

Tabela zawiera informacje o zamówieniach złożonych na wynos.Posiada klucz główny *OrderID,* datę złożenia zamówienia *DateOrder,* datę odbioru zamówienia i jego godzinę *DateReceive, HourReceive* oraz informacje o tym czy zamówienie zostało już opłacone *IsPaid.*

Warunki integralności:

1.DateRecive ma być późniejsza lub taka sama jak DateOrder

```
CREATE TABLE [dbo].[TakeAwayOrder](
    [DateOrder] [date] NOT NULL,
    [DateReceive] [date] NOT NULL,
    [HourReceive] [time](7) NOT NULL,
    [IsPaid] [bit] NOT NULL,
    [OrderID] [int] NOT NULL,
 CONSTRAINT [TakeAwayOrder_pk] PRIMARY KEY CLUSTERED
(
    [OrderID] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[TakeAwayOrder] ADD  DEFAULT
(CONVERT([date],getdate())) FOR [DateOrder]
GO

ALTER TABLE [dbo].[TakeAwayOrder]  WITH CHECK ADD  CONSTRAINT
[DateCheck] CHECK  (([DateReceive]>=[DateOrder]))
GO

ALTER TABLE [dbo].[TakeAwayOrder] CHECK CONSTRAINT [DateCheck]
GO
```

24. Customers

Tabela zawiera ogólne informacje o klientach takie jak adres mailowy *Mail,* numer telefonu *Phone,* adres i miasto *Address, City.* Posiada ona klucz główny *CustomerID* oraz klucz obcy do tabeli Restaurant *RestaurantID.*

Warunki integralności:

1. CustomerID jest unikalne
2. Phone w formacie + xx xxx-xxx-xxx, gdzie x to [0-15]
3. Mail zawiera ‘@’ i ‘ . ’

```
CREATE TABLE [dbo].[Customers](
    [CustomerID] [int] IDENTITY(1,1) NOT NULL,
    [Mail] [nvarchar](30) NULL,
    [Phone] [char](15) NULL,
    [Address] [nvarchar](30) NULL,
    [City] [nvarchar](30) NULL,
    [RestaurantID] [smallint] NOT NULL,
 CONSTRAINT [Customers_pk] PRIMARY KEY CLUSTERED
(
    [CustomerID] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Customers]  WITH CHECK ADD CHECK  (([Mail] like
'%@%.%'))
GO

ALTER TABLE [dbo].[Customers]  WITH CHECK ADD CHECK  (([Phone] like
'+[0-9][0-9] [0-9][0-9][0-9]-[0-9][0-9][0-9]-[0-9][0-9][0-9]'))
GO
```

25. CompanyCustomer

Tabela zawiera informacje o firmach. Posiada klucz główny *CompanyID,* nazwę firmy *CompanyName* oraz jej numer NIP *NIP.*

Warunki integralności:

1.NIP -dwa pierwsze znaki z zakresu [A-Z]

```
CREATE TABLE [dbo].[CompanyCustomer](
    [CompanyName] [nvarchar](60) NOT NULL,
    [NIP] [nvarchar](14) NOT NULL,
    [CompanyID] [int] NOT NULL,
 CONSTRAINT [CompanyCustomer_pk] PRIMARY KEY CLUSTERED
(
    [CompanyID] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[CompanyCustomer]  WITH CHECK ADD  CONSTRAINT
[CompanyCustomer_Customers] FOREIGN KEY([CompanyID])
REFERENCES [dbo].[Customers] ([CustomerID])
GO
```

```
ALTER TABLE [dbo].[CompanyCustomer] CHECK CONSTRAINT
[CompanyCustomer_Customers]
GO

ALTER TABLE [dbo].[CompanyCustomer]  WITH CHECK ADD CHECK  (([NIP] like
'[A-Z][A-Z]%'))
GO
```

4. Widoki

1. MenuView

Wyświetlanie aktualnych dań w menu z datą aktywacji i dezaktywacji

```
CREATE VIEW [dbo].[MenuView]
AS
SELECT ml.MealName, mn.DateActiveMeal, mn.DateDisactiveMeal FROM Menu AS
mn
INNER JOIN Meals AS ml ON mn.MealID = ml.MealID
WHERE mn.DateActiveMeal <= CAST(GETDATE() as date) AND
mn.DatedisactiveMeal >= CAST(GETDATE() as date);
GO
```

2. LoalityCustomerView

Wyświetlanie stałych klientów

```
CREATE VIEW [dbo].[LoyaltyCustomerView]
AS
SELECT ic.CustomerID, ic.FirstName + ' ' + ic.LastName as  Name FROM
IndividualCustomer as ic
INNER JOIN Discount as d ON d.DiscountID = ic.DiscountID
WHERE LoalityCard = 1;
GO
```

3. StoreroomStatusView

Wyświetlenie stanu magazynu

```
CREATE VIEW [dbo].[StoreroomStatusView]
AS
SELECT ProductID, ProductName, ReorderLevel, UnitsInStock FROM Products;
GO
```

4. TemporaryDiscountCustomerView

Wyświetlenie klientów, którzy mogą skorzystać z rabatu tymczasowego

```
CREATE VIEW [dbo].[TemporaryDiscountCustomerView]
AS
SELECT ic.CustomerID, ic.FirstName + ' '  + ic.LastName as Name FROM
IndividualCustomer as ic
INNER JOIN Discount as d ON d.DiscountID = ic.DiscountID
WHERE d.DiscountBeginning <= CAST(GETDATE() as date) AND d.DiscountEnd
>= CAST(GETDATE() as date);
GO
```

5. Widoki Parametryzowane

1. GenerateCompanyReport

Generowanie raportu dla wybranej firmy

```
AS ALTER FUNCTION [dbo].[GenerateCompanyReport](@companyName
NVARCHAR(60))
RETURNS @companyReport TABLE
(
    companyName NVARCHAR(60),
    amoutOfOrders INT,
    sumOfOrders FLOAT,
    totalSavings FLOAT
)
AS
BEGIN
    IF @companyName NOT IN (SELECT CompanyName FROM CompanyCustomer)
    BEGIN
        RETURN
    END

    DECLARE @companyOrders TABLE
    (
        orderId INT
    )
    DECLARE @companyID INT
    DECLARE @orderID INT
    SET @companyID = (SELECT CompanyID FROM CompanyCustomer WHERE
CompanyName = @companyName)
    DECLARE infoCustomers CURSOR FOR
    SELECT OrderID FROM "Order" WHERE CustomerID = @companyID
    OPEN infoCustomers
    FETCH NEXT FROM infoCustomers INTO @orderID
    WHILE @@FETCH_STATUS = 0
    BEGIN
        INSERT @companyOrders
        (
            orderId
        )
        VALUES
        (
            @orderID
        )
        FETCH  NEXT FROM infoCustomers INTO @orderID
    END
    CLOSE infoCustomers
    DEALLOCATE infoCustomers

    DECLARE @withoutDiscounts FLOAT
    SET @withoutDiscounts = 0
    DECLARE @total FLOAT
    SET @total = 0
    DECLARE @totalText VARCHAR(100)
    DECLARE infoOrders CURSOR FOR
    SELECT orderId FROM @companyOrders
    OPEN infoOrders
    FETCH NEXT FROM infoOrders INTO @orderID
    WHILE @@FETCH_STATUS = 0
    BEGIN
        DECLARE sumOrderInfo CURSOR FOR
        SELECT param_value FROM [dbo].[SumOrder](@orderID) WHERE
param_text LIKE 'In total:%'
        OPEN sumOrderInfo
        FETCH NEXT FROM sumOrderInfo INTO @totalText
        WHILE @@FETCH_STATUS = 0
        BEGIN
            SET @total = @total + CAST(SUBSTRING(@totalText, 1,
LEN(@totalText)-2) AS FLOAT)
```

```
            FETCH NEXT FROM sumOrderInfo INTO @totalText
        END
        CLOSE sumOrderInfo
        DEALLOCATE sumOrderInfo

        DECLARE @mealId SMALLINT
        DECLARE infoDetails CURSOR FOR
        SELECT MealID FROM OrderDetails WHERE OrderID = @orderID
        OPEN infoDetails
        FETCH NEXT FROM infoDetails INTO @mealId
        WHILE @@FETCH_STATUS = 0
        BEGIN
            SET @withoutDiscounts = @withoutDiscounts + (SELECT Quantity
FROM OrderDetails WHERE MealID = @mealID AND OrderID = @orderID) *
(SELECT MealPrice FROM Meals WHERE MealID = @mealId)

            FETCH NEXT FROM infoDetails INTO @mealId
        END
        CLOSE infoDetails
        DEALLOCATE infoDetails

        FETCH  NEXT FROM infoOrders INTO @orderID
    END
    CLOSE infoOrders
    DEALLOCATE infoOrders

    INSERT @companyReport
    (
        companyName,
        amoutOfOrders,
        sumOfOrders,
        totalSavings
    )
    VALUES
    (
        @companyName,
        (SELECT COUNT(*) FROM @companyOrders),@total,
        (@withoutDiscounts - @total)
    )
    RETURN
END
```

2. GenerateIndividualCustomerReport

Generowanie raportu dla wybranego klienta indywidualnego

```
ALTER FUNCTION [dbo].[GenerateIndividualCustomerReport](@firstName 
NVARCHAR(30), @lastName NVARCHAR(30))
RETURNS @companyReport TABLE
(
    customerName NVARCHAR(60),
    amoutOfOrders INT,
    sumOfOrders FLOAT,
    totalSavings FLOAT
)
AS
BEGIN
    IF (@firstName+@lastName) NOT IN (SELECT FirstName+LastName FROM 
IndividualCustomer)
    BEGIN
        RETURN
    END

    DECLARE @individualOrders TABLE
    (
        orderId INT
    )

    DECLARE @customerID INT
    DECLARE @orderID INT
    SET @customerID = (SELECT CustomerID FROM IndividualCustomer WHERE 
FirstName = @firstName AND LastName = @lastName)
    DECLARE infoCustomers CURSOR FOR
    SELECT OrderID FROM "Order" WHERE CustomerID = @customerID
    OPEN infoCustomers
    FETCH NEXT FROM infoCustomers INTO @orderID
    WHILE @@FETCH_STATUS = 0
    BEGIN
        INSERT @individualOrders
        (
            orderId
        )
        VALUES
        (
            @orderID
        )
        FETCH  NEXT FROM infoCustomers INTO @orderID
    END
    CLOSE infoCustomers
    DEALLOCATE infoCustomers

    DECLARE @withoutDiscounts FLOAT
    SET @withoutDiscounts = 0
    DECLARE @total FLOAT
    SET @total = 0
    DECLARE @totalText VARCHAR(100)
    DECLARE infoOrders CURSOR FOR
    SELECT orderId FROM @individualOrders
    OPEN infoOrders
    FETCH NEXT FROM infoOrders INTO @orderID
    WHILE @@FETCH_STATUS = 0
    BEGIN
        DECLARE sumOrderInfo CURSOR FOR
        SELECT param_value FROM [dbo].[SumOrder](@orderID) WHERE
param_text LIKE 'In total:%'
        OPEN sumOrderInfo
        FETCH NEXT FROM sumOrderInfo INTO @totalText
        WHILE @@FETCH_STATUS = 0
        BEGIN
            SET @total = @total + CAST(SUBSTRING(@totalText, 1,
LEN(@totalText)-2) AS FLOAT)
            FETCH NEXT FROM sumOrderInfo INTO @totalText
        END
        CLOSE sumOrderInfo
        DEALLOCATE sumOrderInfo

        DECLARE @mealId SMALLINT
        DECLARE infoDetails CURSOR FOR
        SELECT MealID FROM OrderDetails WHERE OrderID = @orderID
        OPEN infoDetails
        FETCH NEXT FROM infoDetails INTO @mealId
        WHILE @@FETCH_STATUS = 0
        BEGIN
            SET @withoutDiscounts = @withoutDiscounts + (SELECT Quantity
FROM OrderDetails WHERE MealID = @mealID AND OrderID = @orderID) *
(SELECT MealPrice FROM Meals WHERE MealID = @mealId)

            FETCH NEXT FROM infoDetails INTO @mealId
        END
        CLOSE infoDetails
        DEALLOCATE infoDetails

        FETCH  NEXT FROM infoOrders INTO @orderID
    END
    CLOSE infoOrders
    DEALLOCATE infoOrders

    INSERT @companyReport
    (
        customerName,
        amoutOfOrders,
        sumOfOrders,
        totalSavings
    )
    VALUES
    (
        (@firstName + ' ' + @lastName),
        (SELECT COUNT(*) FROM @individualOrders),@total,
        (@withoutDiscounts - @total)
    )
    RETURN
END
```

3. SumOrder

Zsumowanie konkretnego zamówienia

```
ALTER FUNCTION [dbo].[SumOrder](@orderID INT)
RETURNS @summary TABLE
(
    param_text VARCHAR(100),
    param_value VARCHAR(100)
)
AS
BEGIN
    DECLARE @restaurantID SMALLINT
    DECLARE @customerID INT
    DECLARE @name VARCHAR(61)
    DECLARE @type BIT
    SET @customerID = (SELECT CustomerID FROM [Order] WHERE OrderID =
@orderID)
```

```
    SET @restaurantID = (SELECT RestaurantID FROM Customers WHERE
CustomerID = @customerID)
    IF(dbo.IsIndividualCustomer(@customerID) = 1)
    BEGIN
    SET @name = (SELECT  FirstName + ' ' + LastName FROM
IndividualCustomer WHERE @customerID = CustomerID)
    END
    ELSE
    BEGIN
    SET @name = (SELECT  CompanyName FROM CompanyCustomer WHERE
@customerID = CompanyID)
    END
    INSERT @summary
        (
        param_text,
        param_value
        )
        VALUES
        (
        'Name: ',
        @name
        )
    SET @type =
    (
    SELECT CASE WHEN EXISTS
        (
            SELECT OrderID FROM TakeAwayOrder WHERE OrderID = @orderID
        )
        THEN 1
        ELSE 0
    END
    )

    DECLARE @billDate DATE
    IF (@type = 1)
    BEGIN
        IF((SELECT IsPaid FROM TakeAwayOrder WHERE @orderID = OrderID) =
1)
        BEGIN
            SET @billDate = (SELECT DateOrder FROM TakeAwayOrder WHERE
@orderID = OrderID)
        END
        ELSE
        BEGIN
            SET @billDate = (SELECT DateReceive FROM TakeAwayOrder WHERE
@orderID = OrderID)
        END
    END
    ELSE
    BEGIN
        SET @billDate = GETDATE()
    END
        INSERT @summary
        (
        param_text,
        param_value
        )
        VALUES
        (
        'Day Received: ',
        @billDate
        )

    DECLARE @quantity TINYINT
    DECLARE @price MONEY
    DECLARE @mealName NVARCHAR(50)

    DECLARE info CURSOR FOR
    SELECT MealName, MealPrice, Quantity FROM Meals as ml
    INNER JOIN OrderDetails as od ON ml.MealID = od.MealID
    WHERE @orderID = OrderID
    GROUP BY MealName, MealPrice, Quantity
    OPEN info

    FETCH  NEXT FROM info INTO @mealName, @price, @quantity
    WHILE @@FETCH_STATUS = 0
    BEGIN
        INSERT @summary
        (
        param_text,
        param_value
        )
        VALUES
        (
        CONCAT('Meal: ', @mealName, ' Quantity:', @quantity),
        CONCAT('  ', @quantity, ' x ', @price, ' $')
        )
        FETCH  NEXT FROM info INTO @mealName, @price, @quantity
    END
    CLOSE info
    DEALLOCATE info
    DECLARE @total MONEY
    DECLARE @discount float
    SET @total = [dbo].[SumOrderValue](@orderID)
    INSERT @summary
        (
        param_text,
        param_value
        )
        VALUES
        (
        'In total: ',
        CONCAT(@total, ' $')
        )
    RETURN
END
```

4. CustomerDiscountView

Wyświetlanie rabatów dla danego klienta

```
ALTER FUNCTION [dbo].[CustomerDiscountView](@customerID INT)
RETURNS @discount TABLE
(
    param_text VARCHAR(100),
    param_value VARCHAR(100)
)
AS
BEGIN
    DECLARE @discountID SMALLINT
    DECLARE @discountStart DATE
    DECLARE @discountEnd DATE
    DECLARE @loyalityCard BIT
    DECLARE @name VARCHAR(61)
    DECLARE @restaurantID SMALLINT
    SET @name = (SELECT  FirstName + ' ' + LastName FROM
IndividualCustomer WHERE @customerID = CustomerID)
    SET @discountID = (SELECT DiscountID FROM dbo.IndividualCustomer
WHERE @customerID = CustomerID)
```

```
    SET @discountStart = (SELECT DiscountBeginning FROM dbo.Discount
WHERE @discountID = DiscountID)
    SET @discountEnd = (SELECT DiscountEnd FROM dbo.Discount WHERE
@discountID = DiscountID)
    SET @loyalityCard = (SELECT LoalityCard FROM dbo.Discount WHERE
@discountID = DiscountID)
    SET @restaurantID = (SELECT RestaurantID FROM Customers WHERE
@customerID = CustomerID)
    INSERT @discount
    (
    param_text,
    param_value
    )
    VALUES
    (
    'Name: ',
    @name
    )

    IF (@loyalityCard = 1)
        BEGIN
        DECLARE @loyalityDetails VARCHAR(5)
        SET @loyalityDetails = FORMAT((SELECT LooalityValueDiscount FROM
DiscountDetails WHERE @restaurantID = RestaurantID), 'P0')
        INSERT @discount
        (
        param_text,
        param_value
        )
        VALUES
        (
        'Loyality discount: ',
        @loyalityDetails
        )
        END
    IF (@discountStart IS NOT NULL AND @discountStart < GETDATE() AND
@discountEnd > GETDATE())
        BEGIN
        DECLARE @temporaryDetails VARCHAR(5)
        SET @temporaryDetails = FORMAT((SELECT TemporaryValueDiscount
FROM DiscountDetails WHERE @restaurantID = RestaurantID), 'P0')
        INSERT @discount
        (
        param_text,
        param_value
        )
        VALUES
        (
        'Temporary discount: ',
        @temporaryDetails
        )
        END
    IF(NOT (@discountStart IS NOT NULL AND @discountStart < GETDATE()
AND @discountEnd > GETDATE()) AND @loyalityCard = 0)
        BEGIN
        INSERT @discount
        (
        param_text,
        param_value
        )
        VALUES
        (
        'Discount' ,
        'No discount available'
        )
        END
    RETURN
    END
```

5. ShowFreeTables

Wyświetlanie, które stoliki są wolne danego dnia w danej godzinie

```
ALTER FUNCTION [dbo].[ShowFreeTables](@date DATE, @hour TIME,
@restaurantID SMALLINT)
RETURNS @tables TABLE
(
    "table" VARCHAR(100)
)
AS
BEGIN
    DECLARE @table SMALLINT
    DECLARE CUR CURSOR FOR
    SELECT t.TableID FROM Tables t
    WHERE TableID NOT IN
    (
    SELECT tr.TableID  FROM TableReservation tr
    INNER JOIN Reservation r ON r.ReservationID = tr.ReservationID
    WHERE RestaurantID = @restaurantID AND DateReservation = @date
    AND NOT ((StartReservation <= @hour AND EndReservation <= @hour)
OR (StartReservation >= dateadd(HOUR, 2, @hour) AND EndReservation >=
dateadd(HOUR, 2, @hour)))
    )
    OPEN CUR
        FETCH  NEXT FROM CUR INTO @table
    WHILE @@FETCH_STATUS = 0
    BEGIN
        INSERT @tables
            (
            "table"
            )
            VALUES
            (
            @table
            )
            FETCH  NEXT FROM CUR INTO @table
    END
    CLOSE CUR
    DEALLOCATE CUR
    RETURN
END
```

6. ShowReservedTables

Wyświetlanie, które stoliki są zarezerwowane danego dnia i w jakich godzinach.

```
ALTER FUNCTION [dbo].[ShowReservedTables](@date DATE, @restaurantID
SMALLINT)
RETURNS @tables TABLE
(
    "table" VARCHAR(100),
    "start" VARCHAR(100),
    "end"   VARCHAR(100)
)
AS
BEGIN
    DECLARE @tableID SMALLINT
    DECLARE @begining TIME
    DECLARE @ending TIME
    DECLARE info CURSOR FOR
    SELECT t.TableID, StartReservation, EndReservation FROM Reservation
r
    INNER JOIN TableReservation t ON t.ReservationID = r.ReservationID
    INNER JOIN Tables tb ON tb.TableID = t.TableID
    WHERE @restaurantID = RestaurantID AND DateReservation = @date
    OPEN info
    FETCH NEXT FROM info INTO @tableID, @begining, @ending
    WHILE @@FETCH_STATUS = 0
    BEGIN
        INSERT @tables
        (
            "table",
            "start",
            "end"
        )
        VALUES
        (
            @tableID,
            @begining,
            @ending
        )
        FETCH NEXT FROM info INTO @tableID, @begining, @ending
    END
    CLOSE info
    DEALLOCATE info
    RETURN
END
```

7. ShowMenuForGivenDay

Wyświetlanie menu w danym dniu

```
ALTER FUNCTION [dbo].[ShowMenuForGivenDay](@restaurantID SMALLINT, @day
DATE)
RETURNS @menu TABLE
(
    meal NVARCHAR(50)
)
AS
BEGIN
    DECLARE @mealName NVARCHAR(50)
    DECLARE info CURSOR FOR
    SELECT MealName FROM Menu m
    INNER JOIN Meals ml ON ml.MealID = m.MealID
    WHERE @restaurantID = RestaurantID AND
    DateActiveMeal IS NOT NULL AND
    DateActiveMeal <= @day AND
    (DateDisactiveMeal IS NULL OR DateDisactiveMeal >= @day)
    OPEN info
    FETCH  NEXT FROM info INTO @mealName
    WHILE @@FETCH_STATUS = 0
    BEGIN
        INSERT @menu
            (
            meal
            )
            VALUES
            (
            @mealName
            )
            FETCH  NEXT FROM info INTO @mealName
    END
    CLOSE info
    DEALLOCATE info
    RETURN
END
```

8. ShowDisactiveMeals

Wyświetlanie dań które nie są aktywne aktualne i można je wpisać do menu. Pokazuje również jak długo dane danie nie było w menu.

```
ALTER FUNCTION [dbo].[ShowDisactiveMeals](@restaurantID SMALLINT)
RETURNS @menu TABLE
(
    meal NVARCHAR(50),
    timeWaiting NVARCHAR(7)
)
AS
BEGIN
    DECLARE @mealName NVARCHAR(50)
    DECLARE @disactiveTime DATE
    DECLARE info CURSOR FOR
    SELECT MealName, DateDisactiveMeal FROM Menu m
    INNER JOIN Meals ml ON ml.MealID = m.MealID
    WHERE @restaurantID = RestaurantID AND
     NOT (DateActiveMeal IS NOT NULL AND
    DateActiveMeal <= GETDATE() AND
    (DateDisactiveMeal IS NULL OR DateDisactiveMeal >= GETDATE()))
    OPEN info
    FETCH  NEXT FROM info INTO @mealName, @disactiveTime
    WHILE @@FETCH_STATUS = 0
    BEGIN
        INSERT @menu
            (
            meal,
            timeWaiting
            )
            VALUES
            (
            @mealName,
            DATEDIFF(day, @disactiveTime, GETDATE())
            )
            FETCH  NEXT FROM info INTO @mealName, @disactiveTime
    END
    CLOSE info
    DEALLOCATE info
    RETURN
END
```

9. GenerateInvoice

Generowanie faktury dla danego zamówienia

```
ALTER FUNCTION [dbo].[GenerateInvoice](@orderID INT)
RETURNS @invoice TABLE
(
    messageGiven NVARCHAR(80),
    answer NVARCHAR(80)
)
AS
BEGIN
    DECLARE @customerID INT
    DECLARE @invoiceTime DATE
    DECLARE @restaurantID SMALLINT

    SET @customerID = (SELECT CustomerID FROM "Order" WHERE @orderID =
OrderID)
    SET @restaurantID = (SELECT RestaurantID FROM Customers WHERE
@customerID = CustomerID)

    INSERT @invoice
```

```
            (
                messageGiven,
                answer
            )
            VALUES
            (
                'Name: ',
                (SELECT RestaurantName FROM Restaurant WHERE
RestaurantID = @restaurantID)
            )

    INSERT @invoice
            (
                messageGiven,
                answer
            )
            VALUES
            (
            'Place: ',
            (SELECT Address FROM Restaurant WHERE RestaurantID =
@restaurantID)
            )
    IF ([dbo].[IsTakeSiteOrder](@orderID) = 1)
    BEGIN
        INSERT @invoice
            (
                messageGiven,
                answer
            )
        VALUES
            (
                'Order date: ',
                CONVERT(varchar, getdate(), 1)
            )
        INSERT @invoice
            (
                messageGiven,
                answer
            )
        VALUES
            (
                'Service day: ',
                CONVERT(varchar, getdate(), 1)
            )
    END
    ELSE
    BEGIN
        INSERT @invoice
            (
                messageGiven,
                answer
            )
        VALUES
            (
                'Order date: ',
                (SELECT DateOrder FROM TakeAwayOrder WHERE @orderID =
OrderID)
            )
        INSERT @invoice
            (
                messageGiven,
                answer
            )
        VALUES
            (
                'Service day: ',
                (SELECT DateReceive FROM TakeAwayOrder WHERE @orderID =
OrderID)
            )
    END


    IF ([dbo].[IsIndividualCustomer](@customerID) = 1)
    BEGIN
        INSERT @invoice
            (
                messageGiven,
                answer
            )
            VALUES
            (
            'Name: ',
            (SELECT FirstName + ' ' LastName FROM IndividualCustomer
WHERE CustomerID = @customerID)
            )
        IF ((SELECT Address FROM Customers WHERE CustomerID =
@customerID) IS NOT NULL)
        BEGIN
            INSERT @invoice
            (
                messageGiven,
                answer
            )
            VALUES
            (
            'Address: ',
            (SELECT Address FROM Customers WHERE CustomerID =
@customerID)
            )
        END
    END
    ELSE
    BEGIN
        INSERT @invoice
            (
                messageGiven,
                answer
            )
        VALUES
            (
                'Name: ',
                (SELECT CompanyName FROM CompanyCustomer WHERE CompanyID
= @customerID)
            )
        IF ((SELECT Address FROM Customers WHERE CustomerID =
@customerID) IS NOT NULL)
        BEGIN
            INSERT @invoice
            (
                messageGiven,
                answer
            )
            VALUES
            (
                'Address: ',
                (SELECT Address FROM Customers WHERE CustomerID =
@customerID)
            )
        END
        INSERT @invoice
            (
                messageGiven,
                answer
            )
        VALUES
            (
                'NIP: ',
                (SELECT NIP FROM CompanyCustomer WHERE CompanyID =
@customerID)
            )
    END

    DECLARE @quantity TINYINT
    DECLARE @price MONEY
    DECLARE @mealName NVARCHAR(50)

    DECLARE info CURSOR FOR
    SELECT MealName, MealPrice, Quantity FROM Meals as ml
    INNER JOIN OrderDetails as od ON ml.MealID = od.MealID
    WHERE @orderID = OrderID
    GROUP BY MealName, MealPrice, Quantity
    OPEN info

    FETCH  NEXT FROM info INTO @mealName, @price, @quantity
    WHILE @@FETCH_STATUS = 0
    BEGIN
        INSERT @invoice
        (
        messageGiven,
        answer
        )
        VALUES
        (
        CONCAT('Meal: ', @mealName, ' Quantity:', @quantity),
        CONCAT('  ', @quantity, ' x ', @price, ' $')
        )
        FETCH  NEXT FROM info INTO @mealName, @price, @quantity
    END
    CLOSE info
    DEALLOCATE info
    DECLARE @total MONEY
    SET @total = [dbo].[SumOrderValue](@orderID)

    INSERT @invoice
        (
            messageGiven,
            answer
        )
    VALUES
        (
            'Discout: ',
            FORMAT((1 - [dbo].[GetDiscount](@customerID)), 'P0')
        )

    INSERT @invoice
        (
            messageGiven,
            answer
        )
    VALUES
        (
    'Total: ',
    CONCAT(@total, ' $')
        )
    RETURN
END
```

10. GenerateMonthlyInvoice

Generowanie faktury zbiorczej miesięcznej dla konkretnego klienta.

```
ALTER FUNCTION [dbo].[GenerateMonthlyInvoice](@customerID INT)
RETURNS @invoice TABLE
(
    messageGiven NVARCHAR(150),
    answer NVARCHAR(150)
)
AS
BEGIN
    DECLARE @invoiceTime DATE
    DECLARE @restaurantID SMALLINT
    DECLARE @month TINYINT

    SET @restaurantID = (SELECT RestaurantID FROM Customers WHERE
@customerID = CustomerID)
```

```
    SET @month = MONTH(GETDATE()) - 1
    IF (@month = 0)
    BEGIN
        SET @month = 12
    END
    INSERT @invoice
            (
                messageGiven,
                answer
            )
            VALUES
            (
                'Name: ',
                (SELECT RestaurantName FROM Restaurant WHERE
RestaurantID = @restaurantID)
            )

    INSERT @invoice
            (
                messageGiven,
                answer
            )
            VALUES
            (
            'Place: ',
            (SELECT Address FROM Restaurant WHERE RestaurantID =
@restaurantID)
            )

    INSERT @invoice
        (
            messageGiven,
            answer
        )
    VALUES
        (
            'Invoice date: ',
            CONVERT(varchar, getdate(), 1)
        )
    INSERT @invoice
        (
            messageGiven,
            answer
        )
    VALUES
        (
            'Invoice month: ',
             @month
        )


    IF ([dbo].[IsIndividualCustomer](@customerID) = 1)
    BEGIN
        INSERT @invoice
            (
                messageGiven,
                answer
            )
            VALUES
            (
            'Name: ',
            (SELECT FirstName + ' ' LastName FROM IndividualCustomer
WHERE CustomerID = @customerID)
            )
        IF ((SELECT Address FROM Customers WHERE CustomerID =
@customerID) IS NOT NULL)
        BEGIN
            INSERT @invoice
            (
                messageGiven,
                answer
            )
            VALUES
            (
            'Address: ',
            (SELECT Address FROM Customers WHERE CustomerID =
@customerID)
            )
        END
    END
    ELSE
    BEGIN
        INSERT @invoice
            (
                messageGiven,
                answer
            )
        VALUES
            (
                'Name: ',
                (SELECT CompanyName FROM CompanyCustomer WHERE CompanyID
= @customerID)
            )
        IF ((SELECT Address FROM Customers WHERE CustomerID =
@customerID) IS NOT NULL)
        BEGIN
            INSERT @invoice
            (
                messageGiven,
                answer
            )
            VALUES
            (
                'Address: ',
                (SELECT Address FROM Customers WHERE CustomerID =
@customerID)
            )
        END
        INSERT @invoice
            (
                messageGiven,
                answer
            )
        VALUES
            (
                'NIP: ',
                (SELECT NIP FROM CompanyCustomer WHERE CompanyID =
@customerID)
            )
    END

    DECLARE @orderDate DATE
    DECLARE @price MONEY
    DECLARE @orderID INT
    DECLARE @total MONEY
    DECLARE @yearDate INT
    DECLARE @monthDate INT
    SET @total = 0
    SET @yearDate = YEAR(GETDATE())
    SET @monthDate = MONTH(GETDATE()) - 1
    IF (@monthDate = 0)
    BEGIN
        SET @yearDate = YEAR(GETDATE()) - 1
        SET @monthDate = 12
    END

    DECLARE info CURSOR FOR
    SELECT OrderID FROM "Order" WHERE CustomerID = @customerID
    OPEN info
    FETCH  NEXT FROM info INTO @orderID
    WHILE @@FETCH_STATUS = 0
    BEGIN
        IF ([dbo].IsTakeSiteOrder(@orderID) = 1)
        BEGIN
            SET @price = [dbo].SumOrderValue(@orderID)
            SET @orderDate = (SELECT DateReservation FROM OnSiteOrder o
            INNER JOIN Reservation r ON r.ReservationID =
o.ReservationID
            WHERE @orderID = OrderID)
            IF (MONTH(@orderDate) = @monthDate AND YEAR(@orderDate) =
@yearDate)
            BEGIN
                INSERT @invoice
                (
                messageGiven,
                answer
                )
                VALUES
                (
                    CONCAT('OrderID: ', @orderID, ' Cost:', @price, '
$'),
                    CONCAT('Date: ', @orderDate)
                )
                SET @total = @total + @price
            END
        END
        ELSE
        BEGIN
            SET @price = [dbo].SumOrderValue(@orderID)
            SET @orderDate = (SELECT DateOrder FROM TakeAwayOrder
            WHERE @orderID = OrderID)
            IF (MONTH(@orderDate) = @monthDate AND YEAR(@orderDate) =
@yearDate)
            BEGIN
                INSERT @invoice
                    (
                        messageGiven,
                        answer
                    )
                VALUES
                    (
                        CONCAT('OrderID: ', @orderID, ' Cost:', @price,
' $'),
                        CONCAT('Date: ', @orderDate)
                    )
                SET @total = @total + @price
            END
        END
        FETCH  NEXT FROM info INTO @orderID
    END
    CLOSE info
    DEALLOCATE info
    SET @total = @total*dbo.GetDiscount(@customerID)
    INSERT @invoice
        (
            messageGiven,
            answer
        )
    VALUES
        (
            'Total: ',
            CONCAT(@total, ' $')
        )
    RETURN
END
```

11. ShowIngredientsMeal

Wyświetlanie składników danego dania

```
ALTER FUNCTION [dbo].[ShowIngredientsMeal](@mealID SMALLINT)
RETURNS @names TABLE
(
    productName NVARCHAR(50)
)
```

```
AS
BEGIN
    DECLARE @product NVARCHAR(50)
    DECLARE info CURSOR FOR
    SELECT ProductName FROM MealIngrediens mi
    INNER JOIN Products p ON p.ProductID = mi.ProductID
    WHERE MealID = @mealID
    OPEN info
    FETCH  NEXT FROM info INTO @product
    WHILE @@FETCH_STATUS = 0
    BEGIN
        INSERT @names
            (
                productName
            )
        VALUES
            (
                @product
            )
        FETCH  NEXT FROM info INTO @product
    END
    CLOSE info
    DEALLOCATE info
    RETURN
END
```

12. ShowOrderDates

Wyświetlanie dni kiedy klient składał zamówienia.

```
ALTER FUNCTION [dbo].[ShowOrderDates](@customerID INT)
RETURNS @dates TABLE
(
    orderDate NVARCHAR(50)
)
AS
BEGIN
    DECLARE @orderID INT
    DECLARE @date DATE
    DECLARE info CURSOR FOR
    SELECT OrderID FROM "Order" WHERE CustomerID = @customerID
    OPEN info
    FETCH  NEXT FROM info INTO @orderID
    WHILE @@FETCH_STATUS = 0
    BEGIN
        IF([dbo].IsTakeSiteOrder(@orderID) = 1)
        BEGIN
            INSERT @dates
                (
                    orderDate
                )
            VALUES
                (
                CONVERT(varchar,
                (SELECT DateReservation FROM OnSiteOrder o
                    INNER JOIN Reservation r ON o.ReservationID =
r.ReservationID
                    WHERE @orderID = OrderID),
                    1)
                )
        END
        ELSE
        BEGIN
            INSERT @dates
                (
                    orderDate
                )
            VALUES
                (
                CONVERT(varchar,
                (SELECT DateOrder FROM TakeAwayOrder t
                    WHERE @orderID = OrderID),
                    1)
                )
        END
        FETCH  NEXT FROM info INTO @orderID
    END
    CLOSE info
    DEALLOCATE info
    RETURN
END
```

13. ShowMostFrequentlyMeal

Wyświetlanie 5 najczęściej zamawianych dań w danym przedziale dni

```
ALTER FUNCTION [dbo].[ShowMostFrequentlyMeal](@restaurantId SMALLINT,
@startDate DATE, @endDate DATE)
```

```sql
RETURNS @topMeals TABLE
(
    mealName NVARCHAR(50)
)
AS
BEGIN
    DECLARE @restaurantCustomers TABLE
    (
        customerID INT
    )

    DECLARE @customer INT
    DECLARE infoCustomers CURSOR FOR
    SELECT CustomerID FROM Customers WHERE RestaurantID = @restaurantId
    OPEN infoCustomers
    FETCH NEXT FROM infoCustomers INTO @customer
    WHILE @@FETCH_STATUS = 0
    BEGIN
        INSERT @restaurantCustomers
        (
            customerID
        )
        VALUES
        (
            @customer
        )
        FETCH  NEXT FROM infoCustomers INTO @customer
    END
    CLOSE infoCustomers
    DEALLOCATE infoCustomers

    DECLARE @ordersBetweenDates TABLE
    (
        idOrder INT
    )

    DECLARE @orderekID INT
    DECLARE infoFromTakeAways CURSOR FOR
    SELECT OrderID FROM TakeAwayOrder WHERE DateOrder BETWEEN @startDate
AND @endDate
    OPEN infoFromTakeAways
    FETCH  NEXT FROM infoFromTakeAways INTO @orderekID
    WHILE @@FETCH_STATUS = 0
    BEGIN
        INSERT @ordersBetweenDates
        (
            idOrder
        )
        VALUES
        (
            @orderekID
        )
        FETCH  NEXT FROM infoFromTakeAways INTO @orderekID
    END
    CLOSE infoFromTakeAways
    DEALLOCATE infoFromTakeAways

    DECLARE infoFromReservations CURSOR FOR
    SELECT OrderID FROM OnSiteOrder INNER JOIN Reservation ON
Reservation.ReservationID = OnSiteOrder.ReservationID WHERE
Reservation.DateReservation BETWEEN @startDate AND @endDate
    OPEN infoFromReservations
    FETCH  NEXT FROM infoFromReservations INTO @orderekID
    WHILE @@FETCH_STATUS = 0
    BEGIN
        INSERT @ordersBetweenDates
        (
            idOrder
        )
        VALUES
        (
            @orderekID
        )
        FETCH  NEXT FROM infoFromReservations INTO @orderekID
    END
    CLOSE infoFromReservations
    DEALLOCATE infoFromReservations

    DECLARE @ordersBetweenDatesFromRestaurant TABLE
    (
        idOrder INT
    )

    DECLARE info CURSOR FOR
    SELECT idOrder FROM @ordersBetweenDates
    OPEN info
    FETCH NEXT FROM info INTO @orderekID
    WHILE @@FETCH_STATUS = 0
    BEGIN
        IF @orderekID IN (SELECT OrderID FROM "Order" WHERE CustomerID
IN (SELECT customerID FROM @restaurantCustomers))
        BEGIN
            INSERT @ordersBetweenDatesFromRestaurant
            (
                idOrder
            )
            VALUES
            (
                @orderekID
            )
        END
        FETCH NEXT FROM info INTO @orderekID
    END
    CLOSE info
    DEALLOCATE info

    DECLARE @sortedMealsFreq TABLE
    (
        mealID SMALLINT
    )

    DECLARE @mealID SMALLINT
    DECLARE infoMeals CURSOR FOR
    SELECT TOP 5 MealID FROM OrderDetails WHERE OrderID IN (SELECT
idOrder from @ordersBetweenDatesFromRestaurant) GROUP BY MealID ORDER BY
SUM(Quantity) DESC
    OPEN infoMeals
    FETCH  NEXT FROM infoMeals INTO @mealID
    WHILE @@FETCH_STATUS = 0
    BEGIN
        INSERT @sortedMealsFreq
        (
            mealID
        )
        VALUES
        (
            @mealID
        )
        FETCH  NEXT FROM infoMeals INTO @mealID
    END
    CLOSE infoMeals
    DEALLOCATE infoMeals

    DECLARE @mealName NVARCHAR(50)
    DECLARE mealInfo CURSOR FOR
    SELECT MealName FROM Meals WHERE MealID IN (SELECT mealID FROM
@sortedMealsFreq)
    OPEN mealInfo
    FETCH  NEXT FROM mealInfo INTO @mealName
    WHILE @@FETCH_STATUS = 0
    BEGIN
        INSERT @topMeals
        (
            mealName
        )
        VALUES
        (
            @mealName
        )
        FETCH  NEXT FROM mealInfo INTO @mealName
    END
    CLOSE mealInfo
    DEALLOCATE mealInfo
    RETURN
END
```

14. ShowProductSuppliers

Wyświetlenie dostawców danego produktu

```
ALTER FUNCTION [dbo].[ShowProductSuppliers](@productName NVARCHAR(50))
RETURNS @productSuppliers TABLE
(
    supplierName NVARCHAR(50)
)
AS
BEGIN
    DECLARE @productid SMALLINT
    SET @productId = (SELECT ProductID FROM Products WHERE ProductName =
@productName)

    DECLARE @suppliersIds TABLE
    (
        supplierID SMALLINT
    )

    DECLARE @supplierID SMALLINT
    DECLARE productProviderInfo CURSOR FOR
    SELECT SupplierID FROM ProductProvided WHERE ProductID = @productId
    OPEN productProviderInfo
    FETCH  NEXT FROM productProviderInfo INTO @supplierID
    WHILE @@FETCH_STATUS = 0
    BEGIN
        INSERT @productSuppliers
        (
            supplierName
        )
        VALUES
        (
            (SELECT CompanyName FROM Suppliers WHERE SupplierID =
@supplierID)
        )
        FETCH  NEXT FROM productProviderInfo INTO @supplierID
    END
    CLOSE productProviderInfo
    DEALLOCATE productProviderInfo
    RETURN
END
```

15. ShowNotConfirmedReservation

Wyświetlanie rezerwacji które nie są jeszcze zaakceptowane

```
ALTER FUNCTION [dbo].[ShowNotConfirmedReservation](@restaurantID
SMALLINT)
RETURNS @notConfirmedIds TABLE
(
    ReservationID INT
)
AS
BEGIN
    DECLARE @reservations TABLE
    (
        reservationID INT
    )

    DECLARE @table SMALLINT
    DECLARE info CURSOR FOR
    SELECT TableID FROM Tables WHERE RestaurantID = @restaurantID
    OPEN info
    FETCH  NEXT FROM info INTO @table
    WHILE @@FETCH_STATUS = 0
    BEGIN
        INSERT @reservations
        (
            reservationID
        )
        VALUES
        (
            (SELECT ReservationID FROM TableReservation WHERE TableID =
@table)
        )
        FETCH  NEXT FROM info INTO @table
    END
    CLOSE info
    DEALLOCATE info

    DECLARE @reservationID INT
    DECLARE info CURSOR FOR
    SELECT ReservationID FROM Reservation WHERE ReservationID IN (SELECT
reservationID FROM @reservations) AND IsConfirm = 0
    OPEN info
    FETCH  NEXT FROM info INTO @reservationID
    WHILE @@FETCH_STATUS = 0
    BEGIN
        INSERT @notConfirmedIds
        (
            ReservationID
        )
        VALUES
        (
            @reservationID
        )
        FETCH  NEXT FROM info INTO @reservationID
    END
    CLOSE info
    DEALLOCATE info
    RETURN
END
```

16. ShowAllMeals

Wyświetlenie całej bazy dań

```
ALTER FUNCTION [dbo].[ShowAllMeals](@restaurantID SMALLINT)
RETURNS @meals TABLE
(
    MealName NVARCHAR(50)
)
AS
BEGIN
    DECLARE @mealName NVARCHAR(50)
    DECLARE info CURSOR FOR
    SELECT MealName FROM Meals WHERE RestaurantID = @restaurantID
    OPEN info
    FETCH  NEXT FROM info INTO @mealName
    WHILE @@FETCH_STATUS = 0
    BEGIN
        INSERT @meals
        (
            MealName
        )
        VALUES
        (
            @mealName
        )
        FETCH  NEXT FROM info INTO @mealName
    END
    CLOSE info
    DEALLOCATE info
    RETURN
END
```

17. GenerateMonthlyRestaurantReport

Generowanie miesięcznego raportu dla restauracji

```
ALTER FUNCTION [dbo].[GenerateMonthlyRestaurantReport](@month SMALLINT,
@year SMALLINT, @restaurantID SMALLINT)
```

```
RETURNS @restaurantReport TABLE
(
    totalNumberOfTables INT,
    totalSumOfDiscounts FLOAT,
    mostPopularMeals NVARCHAR(260),
    totalNumberOfCompanyOrders INT,
    totalSumOfCompanyOrders FLOAT,
    totalNumberOfIndividualCustomersOrders INT,
    totalSumOfIndividualCustomersOrders FLOAT,
    mostPopularTimeForIndividualCustomers NVARCHAR(20),
    mostPopularTimeForCompanyCustomers NVARCHAR(20)
)
BEGIN
    DECLARE @OrdersBetweenDates TABLE
    (
        orderId INT
    )

    INSERT @OrdersBetweenDates
    SELECT idOrder FROM [dbo].[GetOrdersBetweenDates](@restaurantID,
DATEFROMPARTS(@year, @month, 1), EOMONTH(DATEFROMPARTS(@year, @month,
1)))

    DECLARE @companyBefore12 INT = 0, @companyBetween12And15 INT = 0,
@companyAfter15 INT = 0
    DECLARE @individualBefore12 INT = 0, @individualBetween12And15 INT =
0, @individualAfter15 INT = 0
    DECLARE @totalNumbersOfTables INT = 0
    DECLARE @reservation INT
    DECLARE @orderHour INT
    DECLARE tableinfo CURSOR FOR
    SELECT DISTINCT ReservationID FROM OnSiteOrder WHERE OrderID IN
(SELECT orderId from @OrdersBetweenDates)
    OPEN tableInfo
    FETCH NEXT FROM tableInfo INTO @reservation
    WHILE @@FETCH_STATUS = 0
    BEGIN
        SET @totalNumbersOfTables = @totalNumbersOfTables + (SELECT
COUNT(*) FROM TableReservation WHERE ReservationID = @reservation)

        SET @orderHour = (SELECT DATEPART(HOUR, StartReservation) FROM
Reservation WHERE ReservationID = @reservation)
```

```
        If [dbo].[IsIndividualCustomer]((SELECT CustomerID FROM
Reservation WHERE ReservationID = @reservation)) = 1
        BEGIN
            IF @orderHour > 15
            BEGIN
                SET @individualAfter15 = @individualAfter15 + 1
            END
            ELSE IF @orderHour > 12
            BEGIN
                SET @individualBetween12And15 =
@individualBetween12And15 + 1
            END
            ELSE
            BEGIN
                SET @individualBefore12 = @individualBefore12 + 1
            END
        END
        ELSE
        BEGIN
            IF @orderHour > 15
            BEGIN
                SET @companyAfter15 = @companyAfter15 + 1
            END
            ELSE IF @orderHour > 12
            BEGIN
                SET @companyBetween12And15 = @companyBetween12And15 + 1
            END
            ELSE
            BEGIN
                SET @companyBefore12 = @companyBefore12 + 1
            END
        END
        FETCH NEXT FROM tableInfo INTO @reservation
    END
    CLOSE tableInfo
    DEALLOCATE tableInfo

    DECLARE @orderID INT
    DECLARE takeawayinfo CURSOR FOR
    SELECT OrderID FROM TakeAwayOrder WHERE OrderID IN (SELECT orderId
FROM @OrdersBetweenDates)
    OPEN takeawayinfo
    FETCH NEXT FROM takeawayinfo INTO @orderID
    WHILE @@FETCH_STATUS = 0
    BEGIN
        SET @orderHour = (SELECT DATEPART(HOUR, HourReceive) FROM
TakeAwayOrder WHERE OrderID = @orderID)

        If [dbo].[IsIndividualCustomer]((SELECT CustomerID FROM "Order"
WHERE OrderID = @orderID)) = 1
        BEGIN
            IF @orderHour > 15
            BEGIN
                SET @individualAfter15 = @individualAfter15 + 1
            END
            ELSE IF @orderHour > 12
            BEGIN
                SET @individualBetween12And15 =
@individualBetween12And15 + 1
            END
            ELSE
            BEGIN
                SET @individualBefore12 = @individualBefore12 + 1
            END
        END
        ELSE
        BEGIN
            IF @orderHour > 15
            BEGIN
                SET @companyAfter15 = @companyAfter15 + 1
            END
            ELSE IF @orderHour > 12
            BEGIN
                SET @companyBetween12And15 = @companyBetween12And15 + 1
            END
            ELSE
            BEGIN
                SET @companyBefore12 = @companyBefore12 + 1
            END
        END
        FETCH NEXT FROM takeawayinfo INTO @reservation
    END
    CLOSE takeawayinfo
    DEALLOCATE takeawayinfo

    DECLARE @mostPopularMeals NVARCHAR(260) = ''
    DECLARE @oneMeal NVARCHAR(50)
    DECLARE mealInfo CURSOR FOR
    SELECT mealName FROM [dbo].[ShowMostFrequentlyMeal](@restaurantID,
DATEFROMPARTS(@year, @month, 1), EOMONTH(DATEFROMPARTS(@year, @month,
1)))
    OPEN mealInfo
    FETCH NEXT FROM mealInfo INTO @oneMeal
    WHILE @@FETCH_STATUS = 0
    BEGIN
        SET @mostPopularMeals += ', ' + @oneMeal
        FETCH NEXT FROM mealInfo INTO @oneMeal
    END
    CLOSE mealInfo
    DEALLOCATE mealInfo

    IF @mostPopularMeals != ''
    BEGIN
        SET @mostPopularMeals = SUBSTRING(@mostPopularMeals,3,
LEN(@mostPopularMeals))
    END

    DECLARE @popularIndividualHours NVARCHAR(20), @popularCompanyHours
NVARCHAR(20)

    IF @companyAfter15 >= @companyBetween12And15 AND @companyAfter15 >=
@companyBefore12
    BEGIN
        SET @popularCompanyHours = 'After 15'
    END
    ELSE IF @companyBetween12And15 >= @companyBefore12
    BEGIN
        SET @popularCompanyHours = 'Between 12 and 15'
    END
    ELSE
    BEGIN
        SET @popularCompanyHours = 'Before 12'
    END

    IF @individualAfter15 >= @individualBetween12And15 AND
@individualAfter15 >= @individualBefore12
    BEGIN
        SET @popularIndividualHours = 'After 15'
    END
    ELSE IF @individualBetween12And15 >= @individualBefore12
    BEGIN
        SET @popularIndividualHours = 'Between 12 and 15'
    END
    ELSE
    BEGIN
        SET @popularIndividualHours = 'Before 12'
    END

    DECLARE @totalDiscounts FLOAT = 0
    DECLARE @numberOfCompanyOrders INT = 0, @sumOfCompanyOrders FLOAT =
0
    DECLARE @numberOfIndividualOrders INT = 0, @sumOfIndividualOrders
FLOAT = 0

    DECLARE @report TABLE
    (
        customerName NVARCHAR(60),
        amoutOfOrders INT,
        sumOfOrders FLOAT,
        totalSavings FLOAT
    )

    INSERT @report
    SELECT * FROM
[dbo].[GenerateMonthlyIndividualCustomersReport](@month,@year,@restauran
tID)

    DECLARE @customerName NVARCHAR(60)
    DECLARE individualInfo CURSOR FOR
    SELECT customerName FROM @report
    OPEN individualInfo
    FETCH NEXT FROM individualInfo INTO @customerName
    WHILE @@FETCH_STATUS = 0
    BEGIN
        SET @numberOfIndividualOrders = @numberOfIndividualOrders +
(SELECT amoutOfOrders FROM @report WHERE customerName = @customerName)
        SET @sumOfIndividualOrders = @sumOfIndividualOrders + (SELECT
sumOfOrders FROM @report WHERE customerName = @customerName)
        SET @totalDiscounts = @totalDiscounts + (SELECT totalSavings
FROM @report WHERE customerName = @customerName)
        FETCH NEXT FROM individualInfo INTO @customerName
    END
```

```sql
    CLOSE individualInfo
    DEALLOCATE individualInfo

    DELETE FROM @report
    INSERT @report
    SELECT * FROM
[dbo].[GenerateMonthlyCompanyReport](@month,@year,@restaurantID)

    DECLARE customerInfo CURSOR FOR
    SELECT customerName FROM @report
    OPEN customerInfo
    FETCH NEXT FROM customerInfo INTO @customerName
    WHILE @@FETCH_STATUS = 0
    BEGIN
        SET @numberOfCompanyOrders = @numberOfCompanyOrders + (SELECT
amoutOfOrders FROM @report WHERE customerName = @customerName)
        SET @sumOfCompanyOrders = @sumOfCompanyOrders + (SELECT
sumOfOrders FROM @report WHERE customerName = @customerName)
        SET @totalDiscounts = @totalDiscounts + (SELECT totalSavings
FROM @report WHERE customerName = @customerName)
        FETCH NEXT FROM customerInfo INTO @customerName
    END
    CLOSE customerInfo
    DEALLOCATE customerInfo

    INSERT @restaurantReport
    (
        totalNumberOfTables,
        totalSumOfDiscounts,
        mostPopularMeals,
        totalNumberOfCompanyOrders,
        totalSumOfCompanyOrders,
        totalNumberOfIndividualCustomersOrders,
        totalSumOfIndividualCustomersOrders,
        mostPopularTimeForIndividualCustomers,
        mostPopularTimeForCompanyCustomers
    )
    VALUES
    (
        @totalNumbersOfTables,
        @totalDiscounts,
        @mostPopularMeals,
        @numberOfCompanyOrders,
        @sumOfCompanyOrders,
        @numberOfIndividualOrders,
        @sumOfIndividualOrders,
        @popularIndividualHours,
        @popularCompanyHours
    )
    RETURN
END
```

18. GenerateWeeklyRestaurantReport

Generowanie tygodniowego raportu dla restauracji

```
ALTER FUNCTION [dbo].[GenerateWeeklyRestaurantReport](@day SMALLINT,
@month SMALLINT, @year SMALLINT, @restaurantID SMALLINT)
RETURNS @restaurantReport TABLE
(
    totalNumberOfTables INT,
    totalSumOfDiscounts FLOAT,
    mostPopularMeals NVARCHAR(260),
    totalNumberOfCompanyOrders INT,
    totalSumOfCompanyOrders FLOAT,
    totalNumberOfIndividualCustomersOrders INT,
    totalSumOfIndividualCustomersOrders FLOAT,
    mostPopularTimeForIndividualCustomers NVARCHAR(20),
    mostPopularTimeForCompanyCustomers NVARCHAR(20)
)
BEGIN
    DECLARE @OrdersBetweenDates TABLE
    (
        orderId INT
    )

    INSERT @OrdersBetweenDates
    SELECT idOrder FROM [dbo].[GetOrdersBetweenDates](@restaurantID,
DATEADD(day, -6, DATEFROMPARTS(@year, @month, @day)),
DATEFROMPARTS(@year, @month, @day))

    DECLARE @companyBefore12 INT = 0, @companyBetween12And15 INT = 0,
@companyAfter15 INT = 0
    DECLARE @individualBefore12 INT = 0, @individualBetween12And15 INT =
0, @individualAfter15 INT = 0
```

```
    DECLARE @totalNumbersOfTables INT = 0
    DECLARE @reservation INT
    DECLARE @orderHour INT
    DECLARE tableinfo CURSOR FOR
    SELECT DISTINCT ReservationID FROM OnSiteOrder WHERE OrderID IN
(SELECT orderId from @OrdersBetweenDates)
    OPEN tableInfo
    FETCH NEXT FROM tableInfo INTO @reservation
    WHILE @@FETCH_STATUS = 0
    BEGIN
        SET @totalNumbersOfTables = @totalNumbersOfTables + (SELECT
COUNT(*) FROM TableReservation WHERE ReservationID = @reservation)

        SET @orderHour = (SELECT DATEPART(HOUR, StartReservation) FROM
Reservation WHERE ReservationID = @reservation)
        If [dbo].[IsIndividualCustomer]((SELECT CustomerID FROM
Reservation WHERE ReservationID = @reservation)) = 1
        BEGIN
            IF @orderHour > 15
            BEGIN
                SET @individualAfter15 = @individualAfter15 + 1
            END
            ELSE IF @orderHour > 12
            BEGIN
                SET @individualBetween12And15 =
@individualBetween12And15 + 1
            END
            ELSE
            BEGIN
                SET @individualBefore12 = @individualBefore12 + 1
            END
        END
        ELSE
        BEGIN
            IF @orderHour > 15
            BEGIN
                SET @companyAfter15 = @companyAfter15 + 1
            END
            ELSE IF @orderHour > 12
            BEGIN
                SET @companyBetween12And15 = @companyBetween12And15 + 1
            END
            ELSE
            BEGIN
                SET @companyBefore12 = @companyBefore12 + 1
            END
        END
        FETCH NEXT FROM tableInfo INTO @reservation
    END
    CLOSE tableInfo
    DEALLOCATE tableInfo

    DECLARE @orderID INT
    DECLARE takeawayinfo CURSOR FOR
    SELECT OrderID FROM TakeAwayOrder WHERE OrderID IN (SELECT orderId
FROM @OrdersBetweenDates)
    OPEN takeawayinfo
    FETCH NEXT FROM takeawayinfo INTO @orderID
    WHILE @@FETCH_STATUS = 0
    BEGIN
        SET @orderHour = (SELECT DATEPART(HOUR, HourReceive) FROM
TakeAwayOrder WHERE OrderID = @orderID)

        If [dbo].[IsIndividualCustomer]((SELECT CustomerID FROM "Order"
WHERE OrderID = @orderID)) = 1
        BEGIN
            IF @orderHour > 15
            BEGIN
                SET @individualAfter15 = @individualAfter15 + 1
            END
            ELSE IF @orderHour > 12
            BEGIN
                SET @individualBetween12And15 =
@individualBetween12And15 + 1
            END
            ELSE
            BEGIN
                SET @individualBefore12 = @individualBefore12 + 1
            END
        END
        ELSE
        BEGIN
            IF @orderHour > 15
            BEGIN
                SET @companyAfter15 = @companyAfter15 + 1
            END
            ELSE IF @orderHour > 12
            BEGIN
                SET @companyBetween12And15 = @companyBetween12And15 + 1
            END
            ELSE
            BEGIN
                SET @companyBefore12 = @companyBefore12 + 1
            END
        END
        FETCH NEXT FROM takeawayinfo INTO @reservation
    END
    CLOSE takeawayinfo
    DEALLOCATE takeawayinfo

    DECLARE @mostPopularMeals NVARCHAR(260) = ''
    DECLARE @oneMeal NVARCHAR(50)
    DECLARE mealInfo CURSOR FOR
    SELECT mealName FROM [dbo].[ShowMostFrequentlyMeal](@restaurantID,
DATEFROMPARTS(@year, @month, 1), EOMONTH(DATEFROMPARTS(@year, @month,
1)))
    OPEN mealInfo
    FETCH NEXT FROM mealInfo INTO @oneMeal
    WHILE @@FETCH_STATUS = 0
    BEGIN
        SET @mostPopularMeals += ', ' + @oneMeal
        FETCH NEXT FROM mealInfo INTO @oneMeal
    END
    CLOSE mealInfo
    DEALLOCATE mealInfo

    IF @mostPopularMeals != ''
    BEGIN
        SET @mostPopularMeals = SUBSTRING(@mostPopularMeals,3,
LEN(@mostPopularMeals))
    END

    DECLARE @popularIndividualHours NVARCHAR(20), @popularCompanyHours
NVARCHAR(20)

    IF @companyAfter15 >= @companyBetween12And15 AND @companyAfter15 >=
@companyBefore12
    BEGIN
        SET @popularCompanyHours = 'After 15'
    END
    ELSE IF @companyBetween12And15 >= @companyBefore12
    BEGIN
        SET @popularCompanyHours = 'Between 12 and 15'
    END
    ELSE
    BEGIN
        SET @popularCompanyHours = 'Before 12'
    END

    IF @individualAfter15 >= @individualBetween12And15 AND
@individualAfter15 >= @individualBefore12
    BEGIN
        SET @popularIndividualHours = 'After 15'
    END
    ELSE IF @individualBetween12And15 >= @individualBefore12
    BEGIN
        SET @popularIndividualHours = 'Between 12 and 15'
    END
    ELSE
    BEGIN
        SET @popularIndividualHours = 'Before 12'
    END

    DECLARE @totalDiscounts FLOAT = 0
    DECLARE @numberOfCompanyOrders INT = 0, @sumOfCompanyOrders FLOAT =
0
    DECLARE @numberOfIndividualOrders INT = 0, @sumOfIndividualOrders
FLOAT = 0

    DECLARE @report TABLE
    (
        customerName NVARCHAR(60),
        amoutOfOrders INT,
        sumOfOrders FLOAT,
        totalSavings FLOAT
    )

    INSERT @report
    SELECT * FROM
[dbo].[GenerateMonthlyIndividualCustomersReport](@month,@year,@restauran
tID)
```

```
    DECLARE @customerName NVARCHAR(60)
    DECLARE individualInfo CURSOR FOR
    SELECT customerName FROM @report
    OPEN individualInfo
    FETCH NEXT FROM individualInfo INTO @customerName
    WHILE @@FETCH_STATUS = 0
    BEGIN
        SET @numberOfIndividualOrders = @numberOfIndividualOrders +
(SELECT amoutOfOrders FROM @report WHERE customerName = @customerName)
        SET @sumOfIndividualOrders = @sumOfIndividualOrders + (SELECT
sumOfOrders FROM @report WHERE customerName = @customerName)
        SET @totalDiscounts = @totalDiscounts + (SELECT totalSavings
FROM @report WHERE customerName = @customerName)
        FETCH NEXT FROM individualInfo INTO @customerName
    END
    CLOSE individualInfo
    DEALLOCATE individualInfo

    DELETE FROM @report
    INSERT @report
    SELECT * FROM
[dbo].[GenerateMonthlyCompanyReport](@month,@year,@restaurantID)

    DECLARE customerInfo CURSOR FOR
    SELECT customerName FROM @report
    OPEN customerInfo
    FETCH NEXT FROM customerInfo INTO @customerName
    WHILE @@FETCH_STATUS = 0
    BEGIN
        SET @numberOfCompanyOrders = @numberOfCompanyOrders + (SELECT
amoutOfOrders FROM @report WHERE customerName = @customerName)
        SET @sumOfCompanyOrders = @sumOfCompanyOrders + (SELECT
sumOfOrders FROM @report WHERE customerName = @customerName)
        SET @totalDiscounts = @totalDiscounts + (SELECT totalSavings
FROM @report WHERE customerName = @customerName)
        FETCH NEXT FROM customerInfo INTO @customerName
    END
    CLOSE customerInfo
    DEALLOCATE customerInfo

    INSERT @restaurantReport
    (
        totalNumberOfTables,
        totalSumOfDiscounts,
        mostPopularMeals,
        totalNumberOfCompanyOrders,
        totalSumOfCompanyOrders,
        totalNumberOfIndividualCustomersOrders,
        totalSumOfIndividualCustomersOrders,
        mostPopularTimeForIndividualCustomers,
        mostPopularTimeForCompanyCustomers
    )
    VALUES
    (
        @totalNumbersOfTables,
        @totalDiscounts,
        @mostPopularMeals,
        @numberOfCompanyOrders,
        @sumOfCompanyOrders,
        @numberOfIndividualOrders,
        @sumOfIndividualOrders,
        @popularIndividualHours,
        @popularCompanyHours
    )
    RETURN
END
```

19. GenerateWeeklyIndividualCustomersReport

Generowanie raportu tygodniowego odnośnie wszystkich klientów indywidualnych

```
ALTER FUNCTION [dbo].[GenerateWeeklyIndividualCustomersReport](@day
SMALLINT, @month SMALLINT, @year SMALLINT, @restaurantID SMALLINT)
RETURNS @monthReport TABLE
(
    customerName NVARCHAR(60),
    amoutOfOrders INT,
    sumOfOrders FLOAT,
    totalSavings FLOAT
)
AS
BEGIN
    DECLARE @individualCustomers TABLE
    (
        customerId INT
    )

    DECLARE @customerID INT
    DECLARE restaurantCustomersInfo CURSOR FOR
    SELECT CustomerID FROM Customers WHERE RestaurantID = @restaurantID
    OPEN restaurantCustomersInfo
    FETCH NEXT FROM restaurantCustomersInfo INTO @customerID
    WHILE @@FETCH_STATUS = 0
    BEGIN
        IF [dbo].[IsIndividualCustomer](@customerID)  = 1
            BEGIN
            INSERT @individualCustomers
            (
                customerId
            )
            VALUES
            (
                @customerID
            )
            END
        FETCH  NEXT FROM restaurantCustomersInfo INTO @customerID
    END
    CLOSE restaurantCustomersInfo
    DEALLOCATE restaurantCustomersInfo

    DECLARE @individualOrders TABLE
    (
        orderId INT
    )

    DECLARE @orderID INT
    DECLARE infoCustomers CURSOR FOR
    SELECT idOrder FROM [dbo].[GetOrdersBetweenDates](@restaurantID,
DATEADD(day, -6, DATEFROMPARTS(@year, @month, @day)),
DATEFROMPARTS(@year, @month, @day))
    OPEN infoCustomers
    FETCH NEXT FROM infoCustomers INTO @orderID
    WHILE @@FETCH_STATUS = 0
    BEGIN
        IF (SELECT CustomerID FROM "Order" WHERE OrderID = @orderID) IN
(SELECT customerId FROM @individualCustomers)
        BEGIN
```

```
            INSERT @individualOrders
            (
                orderId
            )
            VALUES
            (
                @orderID
            )
        END
        FETCH  NEXT FROM infoCustomers INTO @orderID
    END
    CLOSE infoCustomers
    DEALLOCATE infoCustomers

    DECLARE @withoutDiscounts FLOAT
    DECLARE @total FLOAT
    DECLARE @totalText VARCHAR(100)
    DECLARE @amoutOfOrders INT

    DECLARE restaurantCustomersInfo CURSOR FOR
    SELECT customerID FROM @individualCustomers
    OPEN restaurantCustomersInfo
    FETCH NEXT FROM restaurantCustomersInfo INTO @customerID
    WHILE @@FETCH_STATUS = 0
    BEGIN
        SET @withoutDiscounts = 0
        SET @total = 0
        SET @amoutOfOrders = 0
        DECLARE infoOrders CURSOR FOR
        SELECT orderId FROM @individualOrders
        OPEN infoOrders
        FETCH NEXT FROM infoOrders INTO @orderID
        WHILE @@FETCH_STATUS = 0
        BEGIN
            IF @customerID = (SELECT CustomerID FROM "Order" WHERE
OrderID = @orderID)
            BEGIN
                SET @amoutOfOrders = @amoutOfOrders + 1
                DECLARE sumOrderInfo CURSOR FOR
                SELECT param_value FROM [dbo].[SumOrder](@orderID) WHERE
param_text LIKE 'In total:%'
                OPEN sumOrderInfo
                FETCH NEXT FROM sumOrderInfo INTO @totalText
                WHILE @@FETCH_STATUS = 0
                BEGIN
                    SET @total = @total + CAST(SUBSTRING(@totalText, 1, 
LEN(@totalText)-2) AS FLOAT)
                    FETCH NEXT FROM sumOrderInfo INTO @totalText
                END
                CLOSE sumOrderInfo
                DEALLOCATE sumOrderInfo

                DECLARE @mealId SMALLINT
                DECLARE infoDetails CURSOR FOR
                SELECT MealID FROM OrderDetails WHERE OrderID = @orderID
                OPEN infoDetails
                FETCH NEXT FROM infoDetails INTO @mealId
                WHILE @@FETCH_STATUS = 0
                BEGIN
                    SET @withoutDiscounts = @withoutDiscounts + (SELECT 
Quantity FROM OrderDetails WHERE MealID = @mealID AND OrderID = 
@orderID) * (SELECT MealPrice FROM Meals WHERE MealID = @mealId)

                    FETCH NEXT FROM infoDetails INTO @mealId
                END
                CLOSE infoDetails
                DEALLOCATE infoDetails
            END
            FETCH  NEXT FROM infoOrders INTO @orderID
        END
        CLOSE infoOrders
        DEALLOCATE infoOrders

        INSERT @monthReport
        (
            customerName,
            amoutOfOrders,
            sumOfOrders,
            totalSavings
        )
        VALUES
        (
            (SELECT FirstName + ' ' + LastName FROM IndividualCustomer 
WHERE CustomerID = @customerID),
            @amoutOfOrders,
            @total,
            (@withoutDiscounts - @total)
        )
        FETCH NEXT FROM restaurantCustomersInfo INTO @customerID
    END
    RETURN
END
```

20. GenerateWeeklyCompanyReport

Generowanie raportu tygodniowego odnośnie wszystkich firm.

```
ALTER FUNCTION [dbo].[GenerateWeeklyCompanyReport](@day SMALLINT, @month
SMALLINT, @year SMALLINT, @restaurantID SMALLINT)
RETURNS @monthReport TABLE
(
    companyName NVARCHAR(60),
    amoutOfOrders INT,
    sumOfOrders FLOAT,
    totalSavings FLOAT
)
AS
BEGIN
    DECLARE @companyCustomer TABLE
    (
        customerId INT
    )

    DECLARE @companyID INT
    DECLARE restaurantCustomersInfo CURSOR FOR
    SELECT CustomerID FROM Customers WHERE RestaurantID = @restaurantID
    OPEN restaurantCustomersInfo
    FETCH NEXT FROM restaurantCustomersInfo INTO @companyID
    WHILE @@FETCH_STATUS = 0
    BEGIN
        IF [dbo].[IsIndividualCustomer](@companyID) = 0
            BEGIN
            INSERT @companyCustomer
            (
                customerId
            )
            VALUES
            (
                @companyID
            )
            END
        FETCH  NEXT FROM restaurantCustomersInfo INTO @companyID
    END
    CLOSE restaurantCustomersInfo
    DEALLOCATE restaurantCustomersInfo

    DECLARE @companyOrders TABLE
    (
        orderId INT
    )

    DECLARE @orderID INT
    DECLARE infoCustomers CURSOR FOR
    SELECT idOrder FROM [dbo].[GetOrdersBetweenDates](@restaurantID,
DATEADD(day, -6, DATEFROMPARTS(@year, @month, @day)),
DATEFROMPARTS(@year, @month, @day))
    OPEN infoCustomers
    FETCH NEXT FROM infoCustomers INTO @orderID
    WHILE @@FETCH_STATUS = 0
    BEGIN
        IF (SELECT CustomerID FROM "Order" WHERE OrderID = @orderID) IN
(SELECT customerId FROM @companyCustomer)
        BEGIN
            INSERT @companyOrders
            (
                orderId
            )
            VALUES
            (
                @orderID
            )
        END
        FETCH  NEXT FROM infoCustomers INTO @orderID
    END
    CLOSE infoCustomers
    DEALLOCATE infoCustomers

    DECLARE @withoutDiscounts FLOAT
    DECLARE @total FLOAT
    DECLARE @totalText VARCHAR(100)
    DECLARE @amoutOfOrders INT
    DECLARE restaurantCustomersInfo CURSOR FOR
    SELECT customerID FROM @companyCustomer
    OPEN restaurantCustomersInfo
    FETCH NEXT FROM restaurantCustomersInfo INTO @companyID
    WHILE @@FETCH_STATUS = 0
    BEGIN
        SET @withoutDiscounts = 0
        SET @total = 0
        SET @amoutOfOrders = 0
        DECLARE infoOrders CURSOR FOR
        SELECT orderId FROM @companyOrders
        OPEN infoOrders
        FETCH NEXT FROM infoOrders INTO @orderID
        WHILE @@FETCH_STATUS = 0
        BEGIN
            IF @companyID = (SELECT CustomerID FROM "Order" WHERE
OrderID = @orderID)
            BEGIN
                SET @amoutOfOrders = @amoutOfOrders + 1
                DECLARE sumOrderInfo CURSOR FOR
                SELECT param_value FROM [dbo].[SumOrder](@orderID) WHERE
param_text LIKE 'In total:%'
                OPEN sumOrderInfo
                FETCH NEXT FROM sumOrderInfo INTO @totalText
                WHILE @@FETCH_STATUS = 0
                BEGIN
                    SET @total = @total + CAST(SUBSTRING(@totalText, 1,
LEN(@totalText)-2) AS FLOAT)
                    FETCH NEXT FROM sumOrderInfo INTO @totalText
                END
                CLOSE sumOrderInfo
                DEALLOCATE sumOrderInfo

                DECLARE @mealId SMALLINT
                DECLARE infoDetails CURSOR FOR
                SELECT MealID FROM OrderDetails WHERE OrderID = @orderID
                OPEN infoDetails
                FETCH NEXT FROM infoDetails INTO @mealId
                WHILE @@FETCH_STATUS = 0
                BEGIN
```

```
                    SET @withoutDiscounts = @withoutDiscounts + (SELECT
Quantity FROM OrderDetails WHERE MealID = @mealID AND OrderID =
@orderID) * (SELECT MealPrice FROM Meals WHERE MealID = @mealId)

                    FETCH NEXT FROM infoDetails INTO @mealId
                END
                CLOSE infoDetails
                DEALLOCATE infoDetails
            END
            FETCH  NEXT FROM infoOrders INTO @orderID
        END
        CLOSE infoOrders
        DEALLOCATE infoOrders

        INSERT @monthReport
        (
            companyName,
            amoutOfOrders,
            sumOfOrders,
            totalSavings
        )
        VALUES
        (
            (SELECT CompanyName FROM CompanyCustomer WHERE CompanyID =
@companyID),
            @amoutOfOrders,
            @total,
            (@withoutDiscounts - @total)
        )
        FETCH NEXT FROM restaurantCustomersInfo INTO @companyID
    END
    RETURN
END
```

21. GenerateMonthlyIndividualCustomersReport

Generowanie raportu miesięcznego odnośnie wszystkich klientów indywidualnych

```
ALTER FUNCTION [dbo].[GenerateMonthlyIndividualCustomersReport](@month
SMALLINT, @year SMALLINT, @restaurantID SMALLINT)
RETURNS @monthReport TABLE
(
    customerName NVARCHAR(60),
    amoutOfOrders INT,
    sumOfOrders FLOAT,
    totalSavings FLOAT
)
AS
BEGIN
    DECLARE @individualCustomers TABLE
    (
        customerId INT
    )

    DECLARE @customerID INT
    DECLARE restaurantCustomersInfo CURSOR FOR
    SELECT CustomerID FROM Customers WHERE RestaurantID = @restaurantID
    OPEN restaurantCustomersInfo
    FETCH NEXT FROM restaurantCustomersInfo INTO @customerID
    WHILE @@FETCH_STATUS = 0
    BEGIN
        IF [dbo].[IsIndividualCustomer](@customerID) = 1
            BEGIN
            INSERT @individualCustomers
            (
                customerId
            )
            VALUES
            (
                @customerID
            )
            END
        FETCH  NEXT FROM restaurantCustomersInfo INTO @customerID
    END
    CLOSE restaurantCustomersInfo
    DEALLOCATE restaurantCustomersInfo

    DECLARE @individualOrders TABLE
    (
        orderId INT
    )

    DECLARE @orderID INT
    DECLARE infoCustomers CURSOR FOR
    SELECT idOrder FROM [dbo].[GetOrdersBetweenDates](@restaurantID,
DATEFROMPARTS(@year, @month, 1), EOMONTH(DATEFROMPARTS(@year, @month,
1)))
    OPEN infoCustomers
    FETCH NEXT FROM infoCustomers INTO @orderID
    WHILE @@FETCH_STATUS = 0
    BEGIN
        IF (SELECT CustomerID FROM "Order" WHERE OrderID = @orderID) IN
(SELECT customerId FROM @individualCustomers)
        BEGIN
            INSERT @individualOrders
            (
                orderId
            )
            VALUES
            (
                @orderID
            )
        END
        FETCH  NEXT FROM infoCustomers INTO @orderID
    END
    CLOSE infoCustomers
    DEALLOCATE infoCustomers

    DECLARE @withoutDiscounts FLOAT
    DECLARE @total FLOAT
    DECLARE @totalText VARCHAR(100)
    DECLARE @amoutOfOrders INT

    DECLARE restaurantCustomersInfo CURSOR FOR
    SELECT customerID FROM @individualCustomers
    OPEN restaurantCustomersInfo
    FETCH NEXT FROM restaurantCustomersInfo INTO @customerID
    WHILE @@FETCH_STATUS = 0
    BEGIN
        SET @withoutDiscounts = 0
        SET @total = 0
        SET @amoutOfOrders = 0
        DECLARE infoOrders CURSOR FOR
        SELECT orderId FROM @individualOrders
        OPEN infoOrders
        FETCH NEXT FROM infoOrders INTO @orderID
        WHILE @@FETCH_STATUS = 0
        BEGIN
            IF @customerID = (SELECT CustomerID FROM "Order" WHERE
OrderID = @orderID)
            BEGIN
                SET @amoutOfOrders = @amoutOfOrders + 1
                DECLARE sumOrderInfo CURSOR FOR
                SELECT param_value FROM [dbo].[SumOrder](@orderID) WHERE
param_text LIKE 'In total:%'
                OPEN sumOrderInfo
                FETCH NEXT FROM sumOrderInfo INTO @totalText
                WHILE @@FETCH_STATUS = 0
                BEGIN
                    SET @total = @total + CAST(SUBSTRING(@totalText, 1,
LEN(@totalText)-2) AS FLOAT)
                    FETCH NEXT FROM sumOrderInfo INTO @totalText
                END
                CLOSE sumOrderInfo
                DEALLOCATE sumOrderInfo

                DECLARE @mealId SMALLINT
                DECLARE infoDetails CURSOR FOR
                SELECT MealID FROM OrderDetails WHERE OrderID = @orderID
                OPEN infoDetails
                FETCH NEXT FROM infoDetails INTO @mealId
                WHILE @@FETCH_STATUS = 0
                BEGIN
                    SET @withoutDiscounts = @withoutDiscounts + (SELECT
Quantity FROM OrderDetails WHERE MealID = @mealID AND OrderID =
@orderID) * (SELECT MealPrice FROM Meals WHERE MealID = @mealId)

                    FETCH NEXT FROM infoDetails INTO @mealId
                END
                CLOSE infoDetails
                DEALLOCATE infoDetails
            END
            FETCH  NEXT FROM infoOrders INTO @orderID
        END
        CLOSE infoOrders
        DEALLOCATE infoOrders

        INSERT @monthReport
        (
            customerName,
            amoutOfOrders,
            sumOfOrders,
            totalSavings
        )
        VALUES
        (
            (SELECT FirstName + ' ' + LastName FROM IndividualCustomer
WHERE CustomerID = @customerID),
            @amoutOfOrders,
            @total,
            (@withoutDiscounts - @total)
        )
        FETCH NEXT FROM restaurantCustomersInfo INTO @customerID
    END
    RETURN
END
```

22. GenerateMonthlyCompanyReport

Generowanie raportu miesięcznego odnośnie wszystkich firm

```
ALTER FUNCTION [dbo].[GenerateMonthlyCompanyReport](@month SMALLINT,
@year SMALLINT, @restaurantID SMALLINT)
RETURNS @monthReport TABLE
(
    companyName NVARCHAR(60),
    amoutOfOrders INT,
    sumOfOrders FLOAT,
    totalSavings FLOAT
)
AS
BEGIN
    DECLARE @companyCustomers TABLE
    (
        customerId INT
    )

    DECLARE @customerID INT
    DECLARE restaurantCustomersInfo CURSOR FOR
    SELECT CustomerID FROM Customers WHERE RestaurantID = @restaurantID
    OPEN restaurantCustomersInfo
    FETCH NEXT FROM restaurantCustomersInfo INTO @customerID
    WHILE @@FETCH_STATUS = 0
    BEGIN
        IF [dbo].[IsIndividualCustomer](@customerID) = 0
            BEGIN
            INSERT @companyCustomers
            (
                customerId
            )
            VALUES
            (
                @customerID
            )
            END
        FETCH  NEXT FROM restaurantCustomersInfo INTO @customerID
    END
    CLOSE restaurantCustomersInfo
    DEALLOCATE restaurantCustomersInfo

    DECLARE @companyOrders TABLE
    (
        orderId INT
    )

    DECLARE @orderID INT
    DECLARE infoCustomers CURSOR FOR
    SELECT idOrder FROM [dbo].[GetOrdersBetweenDates](@restaurantID,
DATEFROMPARTS(@year, @month, 1), EOMONTH(DATEFROMPARTS(@year, @month,
1)))
    OPEN infoCustomers
    FETCH NEXT FROM infoCustomers INTO @orderID
    WHILE @@FETCH_STATUS = 0
    BEGIN
        IF (SELECT CustomerID FROM "Order" WHERE OrderID = @orderID) IN
(SELECT customerId FROM @companyCustomers)
        BEGIN
            INSERT @companyOrders
            (
                orderId
            )
            VALUES
            (
                @orderID
            )
        END
        FETCH  NEXT FROM infoCustomers INTO @orderID
    END
    CLOSE infoCustomers
    DEALLOCATE infoCustomers

    DECLARE @withoutDiscounts FLOAT
    DECLARE @total FLOAT
    DECLARE @totalText VARCHAR(100)
    DECLARE @amoutOfOrders INT

    DECLARE restaurantCustomersInfo CURSOR FOR
    SELECT customerID FROM @companyCustomers
    OPEN restaurantCustomersInfo
    FETCH NEXT FROM restaurantCustomersInfo INTO @customerID
    WHILE @@FETCH_STATUS = 0
    BEGIN
        SET @withoutDiscounts = 0
        SET @total = 0
        SET @amoutOfOrders = 0
        DECLARE infoOrders CURSOR FOR
        SELECT orderId FROM @companyOrders
        OPEN infoOrders
        FETCH NEXT FROM infoOrders INTO @orderID
        WHILE @@FETCH_STATUS = 0
        BEGIN
            IF @customerID = (SELECT CustomerID FROM "Order" WHERE
OrderID = @orderID)
            BEGIN
                SET @amoutOfOrders = @amoutOfOrders + 1
                DECLARE sumOrderInfo CURSOR FOR
                SELECT param_value FROM [dbo].[SumOrder](@orderID) WHERE
param_text LIKE 'In total:%'
                OPEN sumOrderInfo
                FETCH NEXT FROM sumOrderInfo INTO @totalText
                WHILE @@FETCH_STATUS = 0
                BEGIN
                    SET @total = @total + CAST(SUBSTRING(@totalText, 1,
LEN(@totalText)-2) AS FLOAT)
                    FETCH NEXT FROM sumOrderInfo INTO @totalText
                END
                CLOSE sumOrderInfo
                DEALLOCATE sumOrderInfo
                DECLARE @mealId SMALLINT
                DECLARE infoDetails CURSOR FOR
                SELECT MealID FROM OrderDetails WHERE OrderID = @orderID
                OPEN infoDetails
                FETCH NEXT FROM infoDetails INTO @mealId
                WHILE @@FETCH_STATUS = 0
                BEGIN
                    SET @withoutDiscounts = @withoutDiscounts + (SELECT
Quantity FROM OrderDetails WHERE MealID = @mealID AND OrderID =
@orderID) * (SELECT MealPrice FROM Meals WHERE MealID = @mealId)

                    FETCH NEXT FROM infoDetails INTO @mealId
                END
                CLOSE infoDetails
                DEALLOCATE infoDetails
            END
            FETCH  NEXT FROM infoOrders INTO @orderID
        END
        CLOSE infoOrders
        DEALLOCATE infoOrders

        INSERT @monthReport
        (
            companyName,
            amoutOfOrders,
            sumOfOrders,
            totalSavings
        )
        VALUES
        (
            (SELECT CompanyName FROM CompanyCustomer WHERE CompanyID =
@customerID),
            @amoutOfOrders,
            @total,
            (@withoutDiscounts - @total)
        )
        FETCH NEXT FROM restaurantCustomersInfo INTO @customerID
    END
    RETURN
END
```

23. MonthlyIncomeView

Wyświetlanie miesięcznego przychodu restauracji.

```
ALTER FUNCTION [dbo].[MonthlyIncomeView](@restaurantID SMALLINT)
RETURNS @result TABLE
(
    messageGiven NVARCHAR(150),
    answer NVARCHAR(150)
)
AS
BEGIN
    DECLARE @month TINYINT
    DECLARE @year SMALLINT

    SET @year = YEAR(GETDATE())
    SET @month = MONTH(GETDATE()) - 1
    IF (@month = 0)
    BEGIN
        SET @month = 12
        SET @year = YEAR(GETDATE()) - 1
    END
    INSERT @result
            (
                messageGiven,
                answer
            )
            Values
            (
                'Name: ',
                (SELECT RestaurantName FROM Restaurant WHERE
RestaurantID = @restaurantID)
            )

    INSERT @result
            (
                messageGiven,
                answer
            )
        VALUES
            (
                'Date: ',
                CONVERT(varchar, getdate(), 1)
            )
    INSERT @result
            (
                messageGiven,
                answer
            )
        VALUES
            (
                'Month: ',
                @month
            )

    INSERT @result
            (
                messageGiven,
                answer
            )
        VALUES
            (
                'Year: ',
                @year
            )

    DECLARE @orderID INT
    DECLARE @total MONEY

    SET @total = 0
    DECLARE info CURSOR FOR
    SELECT o.OrderID FROM "Order" as o
        INNER JOIN TakeAwayOrder as tao ON (MONTH(tao.DateOrder) =
@month AND YEAR(tao.DateOrder) = @year AND o.OrderID = tao.OrderID)


    OPEN info
    FETCH  NEXT FROM info INTO @orderID
    WHILE @@FETCH_STATUS = 0
    BEGIN
        SET @total = @Total + [dbo].[SumOrderValue](@orderID)
        FETCH  NEXT FROM info INTO @orderID
    END
    CLOSE info
    DEALLOCATE info
    DECLARE info CURSOR FOR
    SELECT o.OrderID FROM "Order" as o
        JOIN OnSiteOrder as oo ON (oo.OrderID = o.OrderID)
        JOIN Reservation as r ON (r.ReservationID = oo.ReservationID AND
MONTH(r.DateReservation) = @month AND YEAR(r.DateReservation) = @year)
    OPEN info
    FETCH  NEXT FROM info INTO @orderID
    WHILE @@FETCH_STATUS = 0
    BEGIN
        SET @total = @Total + [dbo].[SumOrderValue](@orderID)
        FETCH  NEXT FROM info INTO @orderID
    END
    CLOSE info
    DEALLOCATE info

    INSERT @result
        (
            messageGiven,
            answer
        )
    VALUES
        (
            'Total income: ',
            CONCAT(@total, ' $')
        )
    RETURN
END
```

24. GetOrdersBetweenDates

Zwraca tabelę wszystkich OrderID które zostały złożone między danymi datami

```
ALTER FUNCTION [dbo].[GetOrdersBetweenDates](@restaurantId SMALLINT,
@startDate DATE, @endDate DATE)
RETURNS @ordersBetweenDatesFromRestaurant TABLE
(
    idOrder INT
)
AS
BEGIN
    DECLARE @restaurantCustomers TABLE
    (
        customerID INT
    )
    DECLARE @customer INT
    DECLARE infoCustomers CURSOR FOR
    SELECT CustomerID FROM Customers WHERE RestaurantID = @restaurantId
    OPEN infoCustomers
    FETCH NEXT FROM infoCustomers INTO @customer
    WHILE @@FETCH_STATUS = 0
    BEGIN
        INSERT @restaurantCustomers
        (
            customerID
        )
        VALUES
        (
            @customer
        )
        FETCH  NEXT FROM infoCustomers INTO @customer
    END
    CLOSE infoCustomers
    DEALLOCATE infoCustomers

    DECLARE @ordersBetweenDates TABLE
    (
        idOrder INT
    )

    DECLARE @orderekID INT
    DECLARE infoFromTakeAways CURSOR FOR
    SELECT OrderID FROM TakeAwayOrder WHERE DateOrder BETWEEN @startDate
AND @endDate
    OPEN infoFromTakeAways
    FETCH  NEXT FROM infoFromTakeAways INTO @orderekID
    WHILE @@FETCH_STATUS = 0
    BEGIN
        INSERT @ordersBetweenDates
        (
            idOrder
        )
        VALUES
        (
            @orderekID
        )
        FETCH  NEXT FROM infoFromTakeAways INTO @orderekID
    END
    CLOSE infoFromTakeAways
    DEALLOCATE infoFromTakeAways

    DECLARE infoFromReservations CURSOR FOR
    SELECT OrderID FROM OnSiteOrder INNER JOIN Reservation ON
Reservation.ReservationID = OnSiteOrder.ReservationID WHERE
Reservation.DateReservation BETWEEN @startDate AND @endDate
    OPEN infoFromReservations
    FETCH  NEXT FROM infoFromReservations INTO @orderekID
    WHILE @@FETCH_STATUS = 0
    BEGIN
        INSERT @ordersBetweenDates
        (
            idOrder
        )
        VALUES
        (
            @orderekID
        )
        FETCH  NEXT FROM infoFromReservations INTO @orderekID
    END
    CLOSE infoFromReservations
    DEALLOCATE infoFromReservations

    DECLARE info CURSOR FOR
    SELECT idOrder FROM @ordersBetweenDates
    OPEN info
    FETCH NEXT FROM info INTO @orderekID
    WHILE @@FETCH_STATUS = 0
    BEGIN
        IF @orderekID IN (SELECT OrderID FROM "Order" WHERE CustomerID
IN (SELECT customerID FROM @restaurantCustomers))
        BEGIN
            INSERT @ordersBetweenDatesFromRestaurant
            (
                idOrder
            )
            VALUES
            (
                @orderekID
            )
        END
        FETCH NEXT FROM info INTO @orderekID
    END
    CLOSE info
    DEALLOCATE info
    RETURN
END
```

6. Funkcje zwracające wartości skalarne

1. GetDiscount

Zwrócenie wartość rabatu dla danego klienta

```
CREATE FUNCTION [dbo].[GetDiscount](@customerID INT)
RETURNS FLOAT
AS
BEGIN
    IF ([dbo].[IsIndividualCustomer](@customerID)= 0)
        BEGIN
            RETURN 1
        END

    DECLARE @discountID INT
    DECLARE @result FLOAT
    DECLARE @loyality BIT
    DECLARE @temporary BIT
    DECLARE @restaurantID SMALLINT

    SET @result = 1
    SET @discountID = (SELECT DiscountID FROM IndividualCustomer WHERE
@customerID = CustomerID)
    SET @restaurantID = (SELECT RestaurantID FROM Customers WHERE
@customerID = CustomerID)
    SET @loyality =
    (
    SELECT CASE WHEN
    (SELECT LoalityCard FROM Discount WHERE @discountID = DiscountID) =
1
        THEN 1
        ELSE 0
    END
    )
    SET @temporary =
    (
            SELECT CASE WHEN EXISTS
        (
            SELECT DiscountID FROM Discount WHERE
            @discountID = DiscountID AND
            DiscountBeginning IS NOT NULL AND
            DiscountEnd IS NOT NULL AND
            DiscountBeginning <= GETDATE() AND
            DiscountEND >= GETDATE()
        )
        THEN 1
        ELSE 0
    END
    )
    IF (@loyality = 1 AND @result > (1-(SELECT LooalityValueDiscount
FROM DiscountDetails WHERE RestaurantID = @restaurantID)))
    BEGIN
        SET @result = (1-(SELECT LooalityValueDiscount FROM
DiscountDetails WHERE RestaurantID = @restaurantID))
    END
    IF (@temporary = 1 AND @result > (1-(SELECT TemporaryValueDiscount
FROM DiscountDetails WHERE RestaurantID = @restaurantID)))
    BEGIN
        SET @result = (1-(SELECT TemporaryValueDiscount FROM
DiscountDetails WHERE RestaurantID = @restaurantID))
    END
    RETURN @result
END
```

2. IsIndividualCustomer

Sprawdzenie czy klient jest indywidualnym, czy jest firmą

```
ALTER FUNCTION [dbo].[IsIndividualCustomer](@customerID INT)
RETURNS BIT
AS
BEGIN
    DECLARE @result BIT
    SET @result =
    (
    SELECT CASE WHEN EXISTS
        (
            SELECT CustomerID FROM IndividualCustomer WHERE @customerID
= CustomerID
        )
        THEN 1
        ELSE 0
    END
    )
    RETURN @result
END
```

3. IsTakeSiteOrder

Sprawdzenie czy dane zamówienie jest na miejscu

```
ALTER FUNCTION [dbo].[IsTakeSiteOrder](@orderID INT)
RETURNS BIT
AS
BEGIN
    DECLARE @result BIT
    SET @result =
    (
    SELECT CASE WHEN EXISTS
        (
            SELECT OrderID FROM OnSiteOrder WHERE @orderID = OrderID
        )
        THEN 1
        ELSE 0
    END
    )
    RETURN @result
END
```

4. SumOrderValue

Wyświetlenie zsumowanej wartości danego zamówienia

```
ALTER FUNCTION [dbo].[SumOrderValue](@orderID INT)
RETURNS MONEY
AS
BEGIN
    DECLARE @result MONEY
    DECLARE @customerID INT
    DECLARE @discount float
    SET @customerID = (SELECT CustomerID FROM "Order" WHERE @orderID =
OrderID)
```

```sql
    SET @discount = [dbo].[GetDiscount](@customerID)
    SET @result =
    (
    SELECT SUM(Quantity * MealPrice * @discount) FROM OrderDetails o
    INNER JOIN Meals m ON m.MealID = o.MealID
    WHERE OrderID = @orderID
    )
    RETURN @result
END
```

7. Procedury

1. AddNewMeal

Dodanie nowego dania do bazy wszystkich dań.

```sql
CREATE Procedure [dbo].[AddNewMeal]
(
@categoryID TINYINT,
@mealPrice SMALLMONEY,
@mealName NVARCHAR(50),
@photo IMAGE = NULL,
@restaurantID SMALLINT
)
AS
BEGIN
   DECLARE @mealID SMALLINT
   SET IDENTITY_INSERT dbo.Meals ON
   SET @mealID = IDENT_CURRENT('dbo.Meals') + 1

   INSERT INTO dbo.Meals
   (
       MealID,
       CategoryID,
       MealPrice,
       MealName,
       PhotoOfMeal,
       RestaurantID
   )
   VALUES
   (
       @mealID,
       @categoryID,
       @mealPrice,
       @mealName,
       @photo,
       @restaurantID
   )
   SET IDENTITY_INSERT dbo.Meals OFF

   INSERT INTO dbo.Menu
   (
       MealID,
       DateActiveMeal,
       DateDisactiveMeal
   )
   VALUES
   (
       @mealID,
       NULL,
       NULL
   )
END
GO
```

2. AddNewIndividualCustomer

Dodanie nowego klienta indywidualnego do bazy danych.

```
ALTER PROCEDURE [dbo].[AddNewIndividualCustomer]
(
@firstName NVARCHAR(30),
@lastName NVARCHAR(30),
@restaurantID SMALLINT,
@mail NVARCHAR(30) = NULL,
@phone CHAR(15) = NULL,
@address NVARCHAR(30) = NULL,
@city NVARCHAR(30) = NULL
)
AS
BEGIN
    DECLARE @customerID INT
    SET IDENTITY_INSERT dbo.Customers ON
    SET @customerID = IDENT_CURRENT('dbo.Customers') + 1
    INSERT INTO dbo.Customers
```

```sql
(
CustomerID,
Mail,
Phone,
"Address",
City,
RestaurantID
)
VALUES
(
@customerID,
@mail,
@phone,
@address,
@city,
@restaurantID
)

SET IDENTITY_INSERT dbo.Customers OFF
SET IDENTITY_INSERT dbo.Discount ON
DECLARE @discountID SMALLINT
SET @discountID = IDENT_CURRENT('dbo.Discount') + 1

INSERT INTO dbo.Discount
(
    DiscountID,
    DiscountBeginning,
    LoalityCard,
    DiscountEnd
)
VALUES
(
    @discountID,
    NULL,
    0,
    NULL
)

INSERT INTO dbo.TemporaryDiscountDetails
(
    OrdersPrice,
    DiscountID
)
```

```
   VALUES
   (
       0,
       @discountID
   )

   SET IDENTITY_INSERT dbo.Discount OFF

   INSERT INTO dbo.IndividualCustomer
   (
   CustomerID,
   FirstName,
   LastName,
   DiscountID
   )
   VALUES
   (
   @customerID,
   @firstName,
   @lastName,
   @discountID
   )
END
GO
```

3. AddNewCompany

Dodanie nowej firmy do bazy danych.

```
ALTER PROCEDURE [dbo].[AddNewCompany]
(
@companyName NVARCHAR(60),
@NIP NVARCHAR(14),
@restaurantID SMALLINT,
@mail NVARCHAR(30) = NULL,
@phone CHAR(15) = NULL,
@address NVARCHAR(30) = NULL,
@city NVARCHAR(30) = NULL
)
AS
BEGIN
        DECLARE @customerID INT
    SET IDENTITY_INSERT dbo.Customers ON
    SET @customerID = IDENT_CURRENT('dbo.Customers') + 1
    INSERT INTO dbo.Customers
    (
    CustomerID,
    Mail,
    Phone,
    "Address",
    City,
    RestaurantID
    )
    VALUES
    (
    @customerID,
    @mail,
    @phone,
    @address,
    @city,
    @restaurantID
    )

    INSERT INTO dbo.CompanyCustomer
    (
    CompanyID,
    CompanyName,
    NIP
    )
    VALUES
    (
    @customerID,
    @companyName,
    @NIP
    )

    SET IDENTITY_INSERT dbo.Customers OFF
END
```

4. AddNewIngredientsToMeal

Dodanie nowych produktów do składników dania.

```
CREATE Procedure [dbo].[AddNewIngredientsToMeal]
(
    @ProductID SMALLINT,
    @MealID SMALLINT
)
```

```
AS
BEGIN
    INSERT INTO [dbo].[MealIngrediens]
    (
        MealID,
        ProductID
    )
    VALUES
    (
        @MealID,
        @ProductID
    )
END
```

5. ActivateMeal

Aktywowanie dania w aktualnym menu

```
ALTER PROCEDURE [dbo].[ActivateMeal]
(
@mealID SMALLINT,
@startDate DATE,
@endDate DATE
)
AS
BEGIN
    UPDATE Menu
    SET
    DateActiveMeal = @startDate,
    DateDisactiveMeal = @endDate
    WHERE @mealID = MealID
END
```

6. AddReservation

Dodanie nowej rezerwacji

```
CREATE Procedure [dbo].[AddReservation]
(
    @IsConfirm BIT,
    @CustomerID INT,
    @NumberOfCustomer SMALLINT,
    @StartReservation TIME,
    @DateReservation DATE
)
```

```
AS
BEGIN
    SET IDENTITY_INSERT dbo.Reservation ON
    DECLARE @ReservationID INT
    SET @ReservationID = IDENT_CURRENT('dbo.Reservation') + 1

    IF (@StartReservation IS NULL)
    BEGIN
        SET @StartReservation = CONVERT(TIME,GETDATE())
    END

    IF (@DateReservation IS NULL)
    BEGIN
        SET @DateReservation = GETDATE()
    END

    DECLARE @EndReservation TIME
    SET @EndReservation = DATEADD(HOUR,2,@StartReservation)

    INSERT INTO dbo."Reservation"
    (
        ReservationID,
        IsConfirm,
        CustomerID,
        NumberOfCustomer,
        StartReservation,
        EndReservation,
        DateReservation
    )
    VALUES
    (
        @ReservationID,
        @IsConfirm,
        @CustomerID,
        @NumberOfCustomer,
        @StartReservation,
        @EndReservation,
        @DateReservation
    )
END
```

7. AddTakeAwayOrder

Stworzenie nowego zamówienia na wynos

```
CREATE Procedure [dbo].[AddTakeAwayOrder]
(
    @EmployeeID SMALLINT,
    @CustomerID INT,
    @DateReceive DATE,
    @HourReceive TIME,
    @IsPaid BIT,
    @Payment NVARCHAR(30)
)
AS
BEGIN
    SET IDENTITY_INSERT dbo."Order" ON
    DECLARE @OrderID SMALLINT
    SET @OrderID = IDENT_CURRENT('dbo.Order') + 1

    DECLARE @HourOrder TIME
    SET @HourOrder = CONVERT(TIME,GETDATE())

    INSERT INTO dbo."Order"
    (
        OrderID,
        HourOrder,
        Payment,
        EmployeeID,
        CustomerID
    )
    VALUES
    (
        @OrderID,
        @HourOrder,
        @Payment,
        @EmployeeID,
        @CustomerID
    )

    DECLARE @DateOrder  DATE
    SET @DateOrder = GETDATE()

    INSERT INTO dbo.TakeAwayOrder
    (
        DateOrder,
        DateReceive,
        HourReceive,
        IsPaid,
        OrderID
    )
    VALUES
    (
        @DateOrder,
        @DateReceive,
        @HourReceive,
        @IsPaid,
        @OrderID
    )
END
```

8. AddOnSiteOrder

Stworzenie nowego zamówienia na miejscu

```
ALTER Procedure [dbo].[AddTakeOnSite]
(
@EmployeeID SMALLINT,
@CustomerID INT,
@Payment NVARCHAR(30),
@reservationID INT
)
AS
BEGIN
    SET IDENTITY_INSERT dbo."Order" ON
    DECLARE @OrderID SMALLINT
    SET @OrderID = IDENT_CURRENT('dbo.Order') + 1

    DECLARE @HourOrder TIME
    SET @HourOrder = CONVERT(TIME,GETDATE())

    INSERT INTO dbo."Order"
    (
        OrderID,
        HourOrder,
        Payment,
        EmployeeID,
        CustomerID
    )
    VALUES
    (
        @OrderID,
        @HourOrder,
        @Payment,
        @EmployeeID,
        @CustomerID
    )
    SET IDENTITY_INSERT dbo."Order" OFF
    DECLARE @orderDate  DATE
    SET @orderDate  = GETDATE()


    INSERT INTO dbo.OnSiteOrder
    (
        OrderID,
        ReservationID,
        OrderDate
    )
    VALUES
    (
        @OrderID,
        @reservationID,
        @orderDate
    )
END
```

9. AddMealToOrder

Dodanie dań do zamówienia

```
ALTER Procedure [dbo].[AddMealToOrder] (
    @MealdID SMALLINT,
    @Quantity TINYINT,
    @OrderID SMALLINT
    )
AS
BEGIN
    INSERT INTO dbo.OrderDetails
    (
        MealID,
        Quantity,
        OrderID
    )
    VALUES
    (
    @MealdID,
    @Quantity,
    @OrderID
    )

    DECLARE @productID SMALLINT
    DECLARE productInfo CURSOR FOR
    SELECT ProductID FROM MealIngrediens WHERE MealID = @MealdID
    OPEN productInfo
    FETCH NEXT FROM productInfo INTO @productID
    WHILE @@FETCH_STATUS = 0
    BEGIN
        UPDATE Products
        SET
        UnitsInStock = UnitsInStock - @Quantity
        WHERE ProductID = @productID

        FETCH NEXT FROM productInfo INTO @productID
    END
    CLOSE productInfo
    DEALLOCATE productInfo
END
```

10. AddTableToReservation

Przydzielenie stolika do rezerwacji

```
ALTER PROCEDURE [dbo].[AddTableToReservation]
(
    @ReservationID INT,
    @TableID SMALLINT,
    @RestaurantID SMALLINT
)
AS
BEGIN
    IF @TableID IS NULL
        BEGIN
            DECLARE @date DATE
            DECLARE @hour TIME
            DECLARE @NumberOfCustomers SMALLINT

            SET @hour = (SELECT StartReservation FROM Reservation WHERE
ReservationID = @ReservationID)
            SET @date = (SELECT DateReservation FROM Reservation WHERE
ReservationID = @ReservationID)
            SET @NumberOfCustomers = (SELECT NumberOfCustomer FROM
Reservation WHERE ReservationID = @ReservationID)

            SET @TableID = (SELECT TOP 1 "table" FROM
dbo.GetTableForREservation(@date, @hour, @RestaurantID,
@NumberOfCustomers))

            INSERT INTO [dbo].[TableReservation]
            (
                ReservationID,
                TableID
            )
            VALUES
            (
                @ReservationID,
                @TableID
            )
        END
    ELSE
        BEGIN
            INSERT INTO [dbo].[TableReservation]
            (
                ReservationID,
                TableID
            )
            VALUES
            (
                @ReservationID,
                @TableID
            )
        END
END
```

11. AddEmployee

Dodanie nowego pracownika.

```
CREATE Procedure [dbo].[AddEmployee]
(
    @FirstName NVARCHAR(30),
    @LastName NVARCHAR(30),
    @Occupation NVARCHAR(30),
    @Salary SMALLMONEY,
    @Mail NVARCHAR(30),
    @Phone CHAR(11),
    @Address NVARCHAR(30),
    @City NVARCHAR(30),
    @HireDate DATE,
    @BirthDate DATE,
    @Photo IMAGE,
    @ManagerID SMALLINT
)
AS
BEGIN
    SET IDENTITY_INSERT dbo.Employees ON
    DECLARE @EmployeeID SMALLINT
    SET @EmployeeID = IDENT_CURRENT('dbo.Employees') + 1
    INSERT INTO [dbo].[Employees]
    (
        EmployeeID,
        FirstName,
        LastName,
        Occupation,
        Salary,
        Mail,
        Phone,
        "Address",
        City,
        HireDate,
        BirthDate,
        Photo,
        ManagerID
    )
    VALUES
    (
        @EmployeeID,
        @FirstName,
        @LastName,
        @Occupation,
        @Salary,
        @Mail,
        @Phone,
        @Address,
        @City,
        @HireDate,
        @BirthDate,
        @Phone,
        @ManagerID
    )
    SET IDENTITY_INSERT dbo.Categories OFF
END
```

12. AddProduct

Dodanie nowego składnika do bazy

```
CREATE PROCEDURE [dbo].[AddProduct]
(
    @ProductaName varchar(50),
    @UnitsInStock SMALLINT,
    @QuantityPerUnit nvarchar(30),
    @ReorderLevel SMALLINT
)
AS
BEGIN
    SET IDENTITY_INSERT dbo.Products ON
    DECLARE @ProductID SMALLINT
    SET @ProductID = IDENT_CURRENT('dbo.Products') + 1
    INSERT INTO [dbo].[Products]
        (
            ProductName,
            ProductID,
            UnitsInStock,
            QuantityPerUnit,
            ReorderLevel
        )
    VALUES
        (
            @ProductaName,
            @ProductID,
            @UnitsInStock,
            @QuantityPerUnit,
            @ReorderLevel
        )
    SET IDENTITY_INSERT dbo.Products OFF
END
```

13. AddNewSupplier

Dodanie nowego dostawcy do bazy

```
ALTER Procedure [dbo].[AddNewSupplier]
(
@phone CHAR(15),
@mail NVARCHAR(30),
@address NVARCHAR(30),
@city NVARCHAR(30),
@companyName NVARCHAR
)
AS
BEGIN
   DECLARE @supplierID SMALLINT
   SET IDENTITY_INSERT dbo.Customers ON
   SET @supplierID = IDENT_CURRENT('dbo.Suppliers') + 1

   INSERT INTO dbo.Suppliers
   (
       SupplierID,
       Phone,
       Mail,
       "Address",
       City,
       CompanyName
   )
   VALUES
   (
       @supplierID,
       @phone,
       @mail,
       @address,
       @city,
       @companyName
   )
   SET IDENTITY_INSERT dbo.Customers OFF
END
GO
```

14. ConfirmReservation

Potwierdzenie rezerwacji

```
CREATE PROCEDURE [dbo].[ConfirmReservation]
(
    @ReservationID INT
)
AS
BEGIN
    UPDATE dbo.Reservation SET IsConfirm = 1 WHERE ReservationID =
@ReservationID;
END
```

15. AddNewRestaurant

Dodanie nowej restauracji

```
ALTER Procedure [dbo].[AddNewRestaurant]
(
@RestaurantName nvarchar(50),
@Phone Char(15),
@Maill nvarchar(30),
@Address nvarchar(30),
@LoalityOrderPrice money,
@LoalityOrderAmount tinyint,
@LooalityValueDiscount float,
@TemporaryOrderPrice smallint,
@TemporaryNumberOfDays tinyint,
@TemporaryValueDiscount float,
@OrderAmount tinyint,
@MinOrderValue smallmoney
)
AS
BEGIN
    SET IDENTITY_INSERT dbo.Restaurant ON
    DECLARE @RestaurantID SMALLINT
    SET @RestaurantID = IDENT_CURRENT('dbo.Restaurant') +1

    INSERT INTO dbo.Restaurant
    (
        RestaurantID,
        RestaurantName,
        Phone,
        Mail,
        "Address",
        OrdersAmount,
        MinOrderValue
    )
    VALUES
    (
        @RestaurantID,
        @RestaurantName,
        @Phone,
        @Maill,
        @Address,
        @OrderAmount,
        @MinOrderValue
    )

    INSERT INTO dbo.DiscountDetails
    (
        RestaurantID,
        LoalityOrderPrice,
        LoalityOrderAmount,
        LooalityValueDiscount,
        TemporaryOrderPrice,
        TemporaryNumberOfDays,
        TemporaryValueDiscount
    )
    VALUES
    (
        @RestaurantID,
        @LoalityOrderPrice,
        @LoalityOrderAmount,
        @LooalityValueDiscount,
        @TemporaryOrderPrice,
        @TemporaryValueDiscount,
        @TemporaryValueDiscount
    )
END
```

16. AddNewManager

Dodanie nowego menadżera

```
CREATE PROCEDURE [dbo].[AddNewManager]
(
    @RestaurantID SMALLINT,
    @FirstName NVARCHAR(30),
    @LastName NVARCHAR(30),
    @Salary SMALLMONEY,
    @Mail NVARCHAR(30),
    @Phone CHAR(15),
    @Address NVARCHAR(30),
    @City NVARCHAR(30),
    @Photo IMAGE
)
AS
BEGIN
    SET IDENTITY_INSERT dbo.Manager ON
    DECLARE @ManagerID SMALLINT
    SET @ManagerID = IDENT_CURRENT('dbo.Manager') +1

    INSERT INTO dbo.Manager
    (
        ManagerID,
        RestaurantID,
        FirstName,
        LastName,
        Salary,
        Mail,
        Phone,
        "Address",
        City,
        Photo
    )
    VALUES
    (
        @ManagerID,
        @RestaurantID,
        @FirstName,
        @LastName,
        @Salary,
        @Mail,
        @Phone,
        @Address,
        @City,
        @Photo
    )
    SET IDENTITY_INSERT dbo.Manager OFF
```

```
END
```

17. AddNewCategory

Dodanie nowej kategorii

```
CREATE Procedure [dbo].[AddNewCategory]
(
    @CategoryName NVARCHAR(20),
    @Description TEXT
)
AS
BEGIN
    SET IDENTITY_INSERT dbo.Categories ON
    DECLARE @CategoryID TINYINT
    SET @CategoryID = IDENT_CURRENT('dbo.Categories') +1
    INSERT INTO [dbo].[Categories]
    (
        CategoryID,
        CategoryName,
        "Description"
    )
    VALUES
    (
        @CategoryID,
        @CategoryName,
        @Description
    )
    SET IDENTITY_INSERT dbo.Categories OFF
END
```

18. AddProductFromSupplier

Dodanie produktu dostarczanego przez dostawcę

```
ALTER Procedure [dbo].[AddProductFromSupplier]
(
@supplierID SMALLINT,
@productID SMALLINT
)
AS
BEGIN
    INSERT INTO dbo.ProductProvided
    (
        SupplierID,
        ProductID
    )
    VALUES
    (
        @supplierID,
        @productID
    )
END
```

19. SetOrderAsFinished

Ustawienie zamówienia jako zakończone.

```
CREATE PROCEDURE [dbo].[SetOrderAsFinished]
(
    @orderID INT
)
AS
BEGIN
    UPDATE [dbo].[Order] SET Finished = 1 WHERE OrderID = @orderID;
END
```

8. Triggery

1. UpperCaseNameCheck

Sprawdzenie czy użytkownik podał dużą literę w imieniu.

```
      ALTER TRIGGER [dbo].[UpperCaseNameCheck] ON
[dbo].[IndividualCustomer]
      AFTER INSERT
      AS
      BEGIN
          DECLARE @name NVARCHAR(30)
          SET @name = (SELECT FirstName FROM inserted)
          IF (CAST(UPPER(SUBSTRING(@name, 1, 1)) AS BINARY) !=
CAST(SUBSTRING(@name, 1, 1) AS BINARY))
          BEGIN
              RAISERROR('FIRST NAME SHOULD START WITH UPPER CASE', 16,
1)
              ROLLBACK TRANSACTION
          END
      END
```

2. CheckAmountOrders

Sprawdzenie czy klient spełnia warunki, aby dokonać zamówienia przy rezerwacji.

```
CREATE TRIGGER [dbo].[CheckAmountOrders] ON [dbo].[Order]
AFTER INSERT
AS
BEGIN
    DECLARE @customerID INT
    DECLARE @orderID INT
    SET @customerID = (SELECT CustomerID FROM inserted)
    SET @orderID = (SELECT OrderID FROM inserted)
    IF (dbo.IsIndividualCustomer(@customerID) = 1 AND
dbo.IsTakeSiteOrder(@orderID) = 1)
    BEGIN
        DECLARE @reservationID INT
        DECLARE @startReservation TIME
        DECLARE @dateReservation DATE
        SET @reservationID = (SELECT ReservationID FROM
OnSiteOrder WHERE OrderID = @orderID)
        SET @dateReservation = (SELECT DateReservation FROM
Reservation WHERE ReservationID = @reservationID)
        SET @startReservation = (SELECT StartReservation FROM
Reservation WHERE ReservationID = @reservationID)
        IF (@dateReservation != CAST(GETDATE() as DATE) OR
@startReservation > CAST(GETDATE() AS TIME))
        BEGIN
            DECLARE @ordersAmount SMALLINT
            DECLARE @amount SMALLINT
            SET @ordersAmount = (SELECT OrdersAmount FROM
Restaurant WHERE RestaurantID = (SELECT RestaurantID FROM Customers
WHERE CustomerID = @customerID))
            SET @amount  = (SELECT COUNT(*) FROM "Order" WHERE
@customerID = CustomerID AND Finished = 1)

            IF (@amount < @ordersAmount)
            BEGIN
                RAISERROR('NOT ENOUGH ORDERS', 16, 1)
                ROLLBACK TRANSACTION
            END
        END
    END
END
```

```
    GO
```

3. CheckReservationNumberCustomers

Sprawdzenie czy są minimum dwie osoby przy rezerwacji.

```
CREATE TRIGGER [dbo].[CheckReservationNumberCustomers ] ON
[dbo].[Reservation]
AFTER INSERT
AS
BEGIN
    IF (SELECT (NumberOfCustomer) FROM Inserted) < 2
    BEGIN
        RAISERROR('RESERVATION FOR AT LEAST 2 PEOPLE!', 16, 1)
        ROLLBACK TRANSACTION
    END

ALTER TABLE [dbo].[Reservation] ENABLE
TRIGGER[CheckReservationNumberCustomers]
END
```

4. CheckReorderLevel

Sprawdzenie czy jest wystarczająca ilość produktów w magazynie.

```
ALTER TRIGGER [dbo].[CheckReorderLevel] ON [dbo].[OrderDetails]
AFTER INSERT
AS
BEGIN
    DECLARE @mealID SMALLINT = (SELECT MealID FROM Inserted)
    DECLARE @productID SMALLINT
    DECLARE @productName NVARCHAR(50)
    DECLARE productInfo CURSOR FOR
    SELECT ProductID FROM MealIngrediens WHERE MealID = @mealID
    OPEN productInfo
    FETCH NEXT FROM productInfo INTO @productID
    WHILE @@FETCH_STATUS = 0
    BEGIN
        IF (SELECT (ReorderLevel - UnitsInStock) FROM Products WHERE
ProductID = @productID) > 0
        BEGIN
            SET @productName = (SELECT ProductName FROM Products WHERE
ProductID = @productID)
            PRINT CONCAT('WATCH OUT! QuantityPerUnit < ReorderLevel FOR
PRODUCT: ', @productName)
```

```sql
        END
        FETCH NEXT FROM productInfo INTO @productID
    END
    CLOSE productInfo
    DEALLOCATE productInfo

ALTER TABLE [dbo].[OrderDetails] ENABLE TRIGGER [CheckReorderLevel]
END
```

5. CheckLoyalityCustomer

Sprawdzenie czy klient może dostać już kartę stałego klienta

```sql
ALTER TRIGGER [dbo].[CheckLoyalityCustomer] ON [dbo].[Order]
AFTER UPDATE
AS
BEGIN
    DECLARE @customerID INT = (SELECT CustomerID FROM Inserted)

    IF ([dbo].[IsIndividualCustomer](@customerID) = 1)
    BEGIN
        DECLARE @discountID SMALLINT = (SELECT DiscountID FROM
IndividualCustomer WHERE CustomerID=@customerID)
        IF ((SELECT LoalityCard FROM Discount WHERE DiscountID =
@discountID) = 0 AND UPDATE(Finished) AND (SELECT Finished FROM
Inserted) = 1)
        BEGIN
            DECLARE @numberOfProperOrders SMALLINT = 0
            DECLARE @restaurantID SMALLINT = (SELECT RestaurantID FROM
Customers WHERE CustomerID = @customerID)
            DECLARE @minNumbersOfOrders TINYINT = (SELECT
LoalityOrderAmount FROM DiscountDetails WHERE RestaurantID =
@restaurantID)
            DECLARE @minPriceOfOrder MONEY = (SELECT LoalityOrderPrice
FROM DiscountDetails WHERE RestaurantID = @restaurantID)

            DECLARE @orderID INT
            DECLARE ordersInfo CURSOR FOR
            SELECT OrderID FROM "Order" WHERE CustomerID = @customerID
            OPEN ordersInfo
            FETCH NEXT FROM ordersInfo INTO @orderID
            WHILE @@FETCH_STATUS = 0
            BEGIN
```

```
                IF ((SELECT Finished FROM "Order" WHERE OrderID =
@orderID) = 1 AND [dbo].[SumOrderValue](@orderID) >= @minPriceOfOrder)
                BEGIN
                    SET @numberOfProperOrders = @numberOfProperOrders +
1
                END

                FETCH NEXT FROM ordersInfo INTO @orderID
            END

            IF (@numberOfProperOrders >= @minNumbersOfOrders)
            BEGIN
                UPDATE [dbo].[Discount] SET LoalityCard = 1 WHERE
DiscountID = @discountID;
                PRINT 'WOOOW, YOU ARE OUR LOYALITY CUSTOMER NOW!'
            END
            ELSE
            BEGIN
                DECLARE @howManyToLoyality NVARCHAR(30) =
CAST((@minNumbersOfOrders-@numberOfProperOrders) AS NVARCHAR(30))
                PRINT CONCAT('UNFORTUNATELY YOU NEED
',@howManyToLoyality ,' MORE TO BE OUT LOYALITY CUSTOMER :(')
            END
        END
    END


ALTER TABLE [dbo].[Order] ENABLE TRIGGER [CheckLoyalityCustomer]
END
```

6. CheckTemporaryDiscount

Sprawdzenie czy klient może dostać tymczasową zniżkę.

```
    ALTER TRIGGER [dbo].[CheckTemporaryDiscount] ON [dbo].[Order]
    AFTER UPDATE
    AS
    BEGIN
        DECLARE @customerID INT
        DECLARE @orderID INT
        DECLARE @restaurantID SMALLINT
        DECLARE @discountID SMALLINT
        DECLARE @temporaryPrice SMALLINT
        DECLARE @restaurantPrice SMALLINT
```

```
          DECLARE @money MONEY
          SET @customerID = (SELECT CustomerID FROM inserted)
          SET @orderID = (SELECT OrderID FROM inserted)
          SET @restaurantID = (SELECT RestaurantID FROM  Customers WHERE
@customerID = CustomerID)
          SET @discountID = (SELECT DiscountID FROM IndividualCustomer
WHERE @customerID = CustomerID)
          SET @temporaryPrice = (SELECT OrdersPrice FROM
TemporaryDiscountDetails WHERE @discountID = DiscountID)
          SET @money = dbo.SumOrderValue(@orderID)
          SET @restaurantPrice = (SELECT TemporaryValueDiscount FROM
DiscountDetails WHERE @restaurantID = RestaurantID)
          IF(@temporaryPrice + @money >=  @restaurantPrice)
          BEGIN
              UPDATE TemporaryDiscountDetails
              SET OrdersPrice = 0
              WHERE DiscountID = @discountID
              DECLARE @days TINYINT
              SET @days = (SELECT TemporaryNumberOfDays FROM
DiscountDetails WHERE RestaurantID = @restaurantID)
              UPDATE Discount
              SET DiscountBeginning = CAST(GETDATE() AS DATE),
                  DiscountEnd = DATEADD(DAY ,@days, CAST(GETDATE() AS
DATE))
              WHERE @discountID = DiscountID
              PRINT 'Temporary discount activated'
          END
      END
```

7. CheckMinOrderValue

Sprawdzenie czy wartość zamówienia jest większa niż wartość, która umożliwia zwiększenie ilości zamówień, po których można dokonywać rezerwacji razem ze złożeniem zamówienia.

```
CREATE TRIGGER [dbo].[CheckMinOrderValue] ON [dbo].[Order]
AFTER UPDATE
AS
BEGIN
    DECLARE @customerID INT
    DECLARE @orderID INT
    SET @customerID = (SELECT CustomerID FROM inserted)
    SET @orderID = (SELECT OrderID FROM inserted)
    IF (dbo.IsIndividualCustomer(@customerID) = 1 AND
dbo.IsTakeSiteOrder(@orderID) = 1)
    BEGIN
        DECLARE @reservationID INT
        DECLARE @startReservation TIME
        DECLARE @dateReservation DATE
        SET @reservationID = (SELECT ReservationID FROM OnSiteOrder
WHERE OrderID = @orderID)
        SET @dateReservation = (SELECT DateReservation FROM Reservation
WHERE ReservationID = @reservationID)
        SET @startReservation = (SELECT StartReservation FROM
Reservation WHERE ReservationID = @reservationID)
        IF (@dateReservation != CAST(GETDATE() as DATE) or
@startReservation > CAST(GETDATE() AS TIME))
        BEGIN
            DECLARE @minOrderValue SMALLMONEY
            DECLARE @price SMALLMONEY
            SET @minOrderValue = (SELECT MinOrderValue FROM Restaurant
WHERE RestaurantID = (SELECT RestaurantID FROM Customers WHERE
CustomerID = @customerID))
            SET @price  = dbo.SumOrderValue(@orderID)

            IF (@price >= @minOrderValue)
            BEGIN
                RAISERROR('NOT ENOUGH MONEY', 16, 1)
                ROLLBACK TRANSACTION
            END
        END
    END
END
GO
```

8. SeaFoodOrder

Sprawdzenie czy zamówienie z owocami morza jest zrobione w odpowiednim terminie

```
CREATE TRIGGER [dbo].[SeaFoodOrder] ON [dbo].[OrderDetails]
AFTER INSERT
AS
BEGIN
    DECLARE @orderID INT
    DECLARE @mealID SMALLINT
    DECLARE @orderDate DATE
    DECLARE @orderFulfillment DATE
    DECLARE @categoryName NVARCHAR(20)

    SET @orderID = (SELECT OrderID FROM inserted)
    SET @mealID = (SELECT MealID FROM inserted)
    SET @orderDate = (SELECT OrderDate FROM OnSiteOrder WHERE OrderID =
@orderID)
    SET @orderFulfillment = (SELECT DateReservation FROM Reservation
        WHERE (SELECT ReservationID FROM OnSiteOrder WHERE OrderID =
@orderID) = ReservationID)
    SET @categoryName = (SELECT CategoryName FROM Categories
        WHERE (SELECT CategoryID FROM Meals WHERE MealID = @mealID) =
CategoryID)

    IF @CategoryName LIKE 'Seafood' AND (DATEPART(dw, @orderFulfillment)
NOT IN (5,6,7)
        OR DATEDIFF(DAY, @orderDate, @orderFulfillment) < 3)
    BEGIN
        RAISERROR('Cannot place an order! You are late', 16, 1)
        ROLLBACK TRANSACTION :(
    END
ALTER TABLE [dbo].[OrderDetails] ENABLE TRIGGER [SeaFoodOrder]
```

9.Role w systemie

1. Administrator

- dodanie nowej restauracji
- dodanie nowego menadżera

2. Menadżer restauracji

- dodanie nowego pracownika
- generowanie raportów
- dodawanie nowych dań do bazy wszystkich dań
- sprawdzanie stanu magazynu
- zmiana daty aktywności dań
- dodanie nowych dostawców
- dodanie nowych produktów

3. Pracownik

- tworzenie zamówień na wynos i na miejscu
- dodanie nowych klientów zarówno indywidualnych jak i firm
- anulowanie zamówień
- akceptowanie rezerwacji

- sprawdzenie rabatów dla klientów
- generowanie faktur
- sprawdzanie stanu magazynu
- dodawanie pozycji do zamówień
- Sprawdzanie stałych klientów

4. Klient Indywidualny

- złożenie zamówienia
- dokonanie rezerwacji z możliwością jednoczesnego zamówienia
- generowanie raportu dla klienta indywidualnego dotyczącego historii zamówień

5. Klient firmowy

- złożenie zamówienia
- dokonanie rezerwacji firmowej i imiennej dla pracownika
- generowanie raportu dla firmy

6. Funkcje systemowe

- obliczanie wartości zamówienia
- sprawdzanie poprawności zamówień zawierających owoce morza
- kontrolowanie ilości produktów w magazynie
- przydzielanie stolika do rezerwacji
- przydzielenie rabatów dla klientów