Podstawy Baz Danych

Dokumentacja projektu

Styczeń 2021

Autorzy:

Jakub Janicki

Hubert Giza

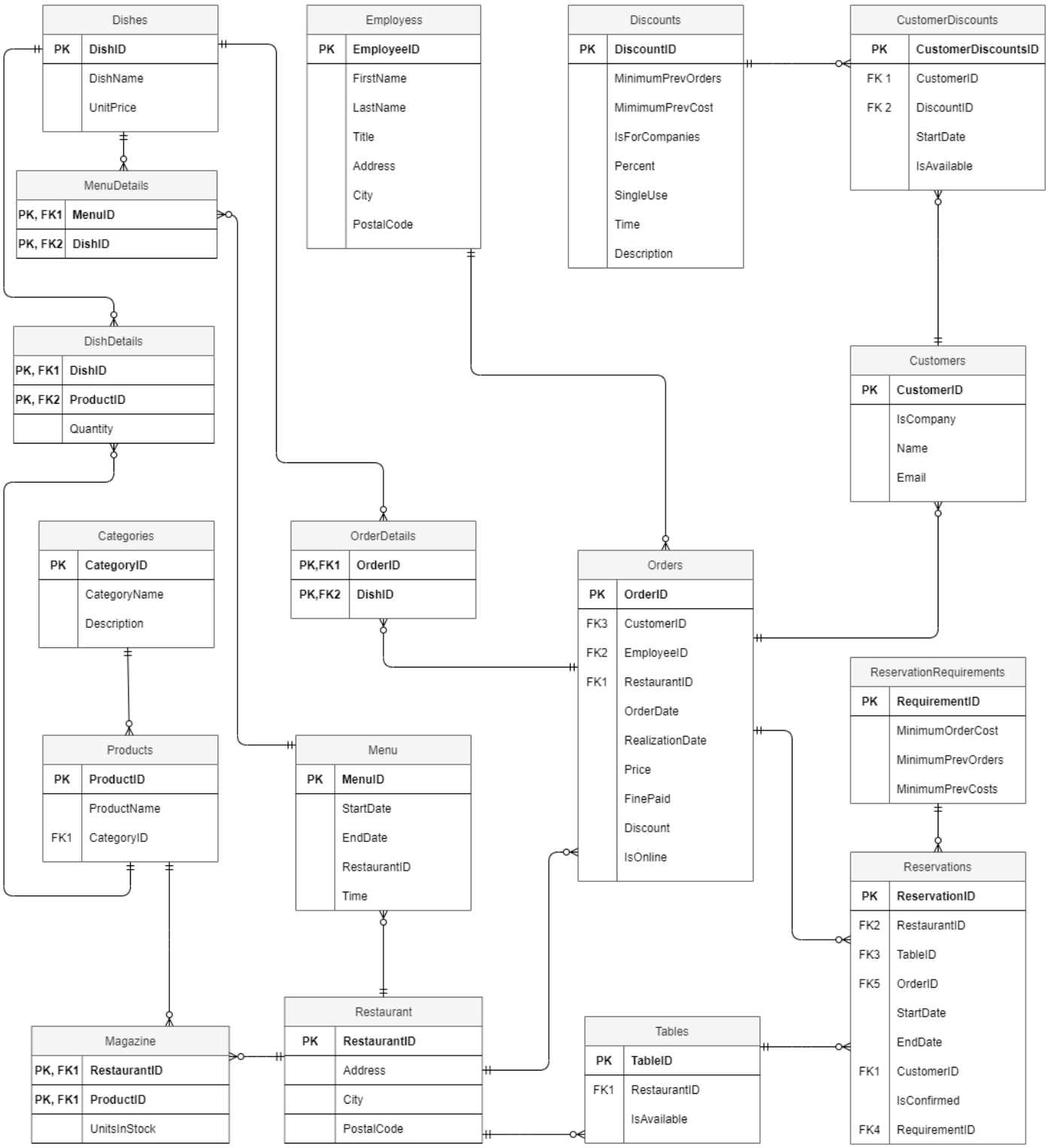
Prowadzący:

dr inż. Leszek Siwik

1. Wprowadzenie

Celem projektu było zaplanowanie systemu bazodanowego dla firmy zajmującej się obsługą sieci restauracji. Obsługa polega na przyjmowaniu i realizacji zamówień zarówno online jak i na miejscu. Obliczenie kosztu usług odbywa się z uwzględnieniem rabatów z rozróżnieniem między klientem indywidualnym a firmą. Możliwe jest także dokonanie rezerwacji.

2. Diagram bazy danych



3. Tabele

3.1. Categories

Przechowuje dane o kategoriach produktów.

CategoryID – identyfikator kategorii, założony indeks klastrowany

CategoryName – nazwa kategorii, założony indeks nieklastrowany

Description – opis kategorii

```
create table Categories
(
    CategoryID   int identity,
    CategoryName varchar(30),
    Description  varchar(200) default NULL,
    constraint CategoriesPK
        primary key (CategoryID),
    constraint CategoriesUnique
        unique (CategoryName),
)
go

create index IDXCategories_CategoryName
    on Categories (CategoryName)
go
```

3.2. CustomerDiscounts

Przechowuje informacje o przynależności do klienta określonych zniżek.

Referencje tabeli do Customers(CustomerID), Discoutns(DisocuntID)

CustomerDiscountsID – identyfikator pary klient,zniżka, założony indeks nieklastrowany

CustomerID – identyfikator klienta, założony indeks nieklastrowany(CustomerID, DiscoutnID)

DiscountID – identyfikator zniżki, założony indeks nieklastrowany(CustomerID, DiscoutnID)

StartDate – czas rozpoczęcia trwania zniżki

```
create table CustomerDiscounts
(
    CustomerDiscountsID int identity,
    CustomerID          int not null,
    DiscountID          int not null,
    StartDate           date,
    IsAvailable         bit,
    constraint CustomerDiscounts_pk
        primary key nonclustered (CustomerDiscountsID),
    constraint CustomerDiscounts_Customer
        foreign key (CustomerID) references Customers,
    constraint CustomerDiscounts_Discounts
        foreign key (DiscountID) references Discounts
)
go

create index IDXCustomerDiscounts_StartDate
    on CustomerDiscounts (StartDate)
go
```

3.3. Customers

Przechowuje informacje o klientach
CustomerID – identyfikator klienta, założony indeks klastrowany
IsCompany – informacja czy jest to klient indywidualny czy firma
Name – nazwa firmy lub imię i nazwisko klienta
Email – email klienta

```
create table Customers
(
    CustomerID int identity,
    IsCompany  bit default 0 not null,
    Name       varchar(50),
    Email      varchar(30),
    constraint CustomersPK
        primary key (CustomerID),
    constraint CustomersCheck
        check ([Email] like '%@%'),
)
go
```

3.4. Discounts

Przechowuje informacje o zniżkach
DiscountID – identyfikator zniżki, założony indeks klastrowany
Percent – procent zniżki
SingleUse – informacja czy zniżka jest jednorazowa
IsForCompanies – informacja czy zniżka jest dla firm
MinimumPrevOrders – minimalna liczba wcześniejszych zamówień klienta
MinimumPrevCost – minimalna wartość wcześniejszych zamówień klienta
Description – opis zniżki
Weeks – czas trwania zniżki w tygodniach

```
create table Discounts
(
    DiscountID        int identity,
    [Percent]         decimal(2, 2) not null,
    SingleUse         bit           not null,
    IsForCompanies    bit default 0 not null,
    MinimumPrevOrders int default 0 not null,
    MinimumPrevCost   int default 0 not null,
    Description       varchar(200),
    Weeks             int default 0,
    constraint DiscountsPK
        primary key (DiscountID),
    constraint DiscountCheck
        check ([MinimumPrevCost] >= 0 AND [MinimumPrevOrders] >= 0 AND
[Weeks] >= 0 AND [Percent] >= 0)
)
go
```

3.5. DishDetails

Przechowuje informacje jakie produkty zawiera danie
Referencje tabeli do Products(ProductID), Dish(DishID)
DishID – identyfikator dania, założony indeks klastrowany (DishID,ProductID)
ProductID – identyfikator produktu,
Quantity – ilość danego produktu użyta do przygotowania dania

```
create table DishDetails
(
    DishID    int not null,
    ProductID int not null,
    Quantity  int,
    constraint DishDetailsPK
        primary key (DishID, ProductID),
    constraint DishDetails_Dishes
        foreign key (DishID) references Dishes,
    constraint DishDetails_Products
        foreign key (ProductID) references Products
)
go
```

3.6. Dishes

Przechowuje informacje o daniu
DishID – identyfikator dania, założony indeks klastrowany
DishName – nazwa dania, założony indeks nieklastrowany
UnitPrice – cena dania
IsVegan – informacja czy danie jest wegańskie

```
create table Dishes
(
    DishID    int identity,
    DishName  varchar(30)   not null,
    UnitPrice decimal(8, 2) not null,
    isVegan   bit default 0 not null,
    constraint DishesPK
        primary key (DishID),
    constraint DishesCheck
        check ([UnitPrice] >= 0),
)
go

create index IDXDishes_DishName
    on Dishes (DishName)
go
```

3.7. Employess

Przechowuje informacje o pracownikach
EmployeeID – identyfikator pracownika, założony indeks klastrowany
FirstName – imię pracownika, założony indeks nieklastrowany (FirstName,LastName)
LastName – nazwisko pracownika, założony indeks nieklastrowany (FirstName,LastName)
Title – tytuł pracownika
Address – adres pracownika
City – miasto zameldowania pracownika
PostalCode – kod pocztowy miasta zameldowania pracownika

```
create table Employess
(
    EmployeeID int identity,
    FirstName  varchar(30) not null,
    LastName   varchar(30) not null,
    Title      varchar(30) default NULL,
    Address    varchar(40) default NULL,
    City       varchar(30) default NULL,
    PostalCode varchar(30) default NULL,
    constraint EmployessPK
        primary key (EmployeeID),
    constraint EmployessCheck
        check ([PostalCode] like '%-%')
go

create index IDXEmployess_Name
    on Employess (FirstName, LastName)
go
```

3.8. Magazine

Przechowuje aktualną ilość produktów w restauracjach
Referencje tabeli do Products(ProductID), Restaurants(RestaurantID)
RestaurantID – identyfikator restauracji, indeks klastrowany (ProductID,RestaurantID)
ProductID – identyfikator produktu, indeks klastrowany (ProductID,RestaurantID)
UnitsInStock – ilość produktów ma magazynie

```
create table Magazine
(
    RestaurantID int not null,
    ProductID    int not null,
    UnitsInStock int,
    constraint MagazinePK
        primary key (RestaurantID, ProductID),
    constraint Magazine_Products
        foreign key (ProductID) references Products,
    constraint Magazine_Restaurants
        foreign key (RestaurantID) references Restaurants
)
go
```

3.9. Menu

Przechowuje informacje o menu
Referencje tabeli do Restaurants(RestaurantID)
MenuID – identyfikator menu , założony indeks klastrowany
StartDate – data początku obowiązywania menu
EndDate – data końca obowiązywania menu
RestaurantID – identyfikator restauracji, założony indeks nieklastrowany
Time – określa czy menu jest poprzednim(p), aktualnym(n), przyszłym(f) menu

```
create table Menu
(
    MenuID       int identity,
    StartDate    date,
    EndDate      date,
    RestaurantID int,
    Time         varchar default 'n' not null,
    constraint MenuPK
        primary key (MenuID),
    constraint Menu_Restaurant
        foreign key (RestaurantID) references Restaurants,
    constraint MenuCheck
        check ([StartDate] <= [EndDate]),
    constraint MenuTimeCheck
        check ([Time] = 'f' OR [Time] = 'n' OR [Time] = 'p'),
)
go

create index IDXMenu_RestaurantID
    on Menu (RestaurantID)
go
```

3.10. MenuDetails

Przechowuje skład danego menu
Referencje tabeli do Menu(MenuID), Dish(DishID)
MenuID – identyfiaktor menu, założony indeks klastrowany (MenuID,DishID)
DishID – identyfiaktor dania, założony indeks klastrowany (MenuID,DishID)

```
create table MenuDetails
(
    MenuID int not null,
    DishID int not null,
    constraint MenuDetailsPK
        primary key (MenuID, DishID),
    constraint MenuDetails_Dishes
        foreign key (DishID) references Dishes,
    constraint MenuDetails_Menu
        foreign key (MenuID) references Menu
)
go
```

3.11. OrderDetails

Przechowuje informacje o składzie zamówienia
Referencje tabeli do Orders(OrderID), Dish(DishID)
OrderID – identyfikator zamówienia, założony indeks klastrowany (OrderID, DishID)
DishID – identyfikator dania, założony indeks klastrowany(OrderID, DishID)

```
create table OrderDetails
(
    OrderID int not null,
    DishID  int not null,
    constraint OrderDetailsPK
        primary key (OrderID, DishID),
    constraint OrderDetails_Dishes
        foreign key (DishID) references Dishes,
    constraint OrderDetails_Order
        foreign key (OrderID) references Orders
)
go
```

3.12. Orders

Przechowuje informacje o zamówieniu, do tabeli przypisane są dwa triggery UpdateCustomerDiscounts, UpdateMagazine przedstawione w sekcji *Triggery* poniżej. Referencje tabeli do Customers(CustomerID), Employee(EmployeeID), Restaurants(RestaurantID)

OrderID – identyfikator zamówienia, założony indeks klastrowany
CustormerID – identyfikator zamawiającego klienta, założony indeks nieklastrowany
EmployeeID – identyfikator pracownika obsługującego, założony indeks nieklastrowany
RestaurantID – identyfikator restauracji w której zamówiono, założony indeks nieklastrowany
ReservationID – identyfikator rezerwacji która złożono, założony indeks nieklastrowany
OrderDate – czas złożenia zamówienia, założony indeks nieklastrowany
RealizationDate – czas realizacji zamówienia
Price – cena zamówienia
FinePaid – kwota uiszczonej zapłaty za zamówienie
isOnline – informacja czy zamówienie zostało złożone online

```
create table Orders
(
    OrderID         int identity,
    CustomerID      int                         not null,
    EmployeeID      int                         not null,
    RestaurantID    int                         not null,
    OrderDate       datetime default getdate()  not null,
    RealizationDate datetime,
    Price           decimal(6, 2),
    FinePaid        decimal(6, 2),
    isOnline        bit                         not null,
    constraint OrdersPK
        primary key (OrderID),
    constraint Orders_Customers
        foreign key (CustomerID) references Customers,
    constraint Orders_Employess
        foreign key (EmployeeID) references Employess,
    constraint Orders_Restaurant
        foreign key (RestaurantID) references Restaurants,
    constraint OrdersCheck
        check ([Price] >= 0 AND [FinePaid] >= 0),
)
go

create index IDXOrders_OrderDate
    on Orders (OrderDate)
go

create index IDXOrders_CustomerID
    on Orders (CustomerID)
go

create index IDXOrders_EmployeeID
    on Orders (EmployeeID)
go

create index IDXOrders_RestaurantID
    on Orders (RestaurantID)
go
```

3.13. Products

Przechowuje informacje o produktach
Referencje tabeli do Categories(CategoryID)
ProductID – identyfikator produktu, założony indeks klastrowany
ProductName – nazwa produktu, założony indeks nieklastrowany
CategoryID – identyfikator kategorii produktu, założony indeks nieklastrowany

```
create table Products
(
    ProductID   int identity,
    ProductName varchar(30) not null,
    CategoryID  int         not null,
    constraint ProductsPK
        primary key (ProductID),
    constraint ProductsUnique
        unique (ProductName),
    constraint Products_Categories
        foreign key (CategoryID) references Categories
)
go
```

3.14. Reservation Requirements

Przechowuje informacje o wymaganiach do uzyskania rezerwacji
RequirementID – identyfikator wymagania, założony indeks klastrowany
MinimumOrderCost – minimalna kwota zamówienia do uzyskania rezerwacji
MinimumPrevOrders – minimalna ilość zamówień klienta do uzyskania rezerwacji
MinimumPrevCost – minimalna kwota jaką klient musi wydać aby dostać rezerwacje

```
create table ReservationRequirements
(
    RequirementID     int identity,
    MinimumOrderCost  int default 0 not null,
    MinimumPrevOrders int default 0 not null,
    MinimumPrevCost   int default 0 not null,
    constraint ReservationRequirementsPK
        primary key (RequirementID),
    constraint ReservationRequirementsCheck
        check ([MinimumPrevOrders] >= 0 AND [MinimumPrevCost] >= 0 AND
[MinimumOrderCost] >= 0)
)
go
```

3.15. Reservations

Przechowuje informacje o rezerwacjach
Referencje tabeli Restaurants(RestaurantID), Tables(TableID), Customers(CustomerID), Requirement(RequirementID), Orders(OrderID)
ReservationID – identyfikator rezerwacji, założony indeks klastrowany
RestaurantID – identyfikator restauracji w której zarezerwowano, założony indeks nieklastrowany
TableID – identyfikator stolika który zarezerwowano, założony indeks nieklastrowany
StartDate – czas rozpoczęcia rezerwacji stolika, założony indeks nieklastrowany (StartDate,EndDate)
EndDate czas zakończenia rezerwacji stolika, założony indeks nieklastrowany (StartDate,EndDate)
CustomerID – identyfikator klienta przez którego jest zamawiane, założony indeks nieklastrowany
IsConfirmed – informacja czy rezerwacja jest potwierdzona
RequirementID – identyfikator wymagań rezerwacji, założony indeks nieklastrowany

```
create table Reservations
(
    ReservationID int identity,
    RestaurantID  int           not null,
    TableID       int,
    StartDate     datetime      not null,
    EndDate       datetime      not null,
    CustomerID    int           not null,
    IsConfirmed   bit default 0 not null,
    RequirementID int           not null,
    OrderID       int,
    constraint ReservationsPK
        primary key (ReservationID),
    constraint Reservation_Tables
        foreign key (TableID) references Tables,
    constraint Reservations_Customers
        foreign key (CustomerID) references Customers,
    constraint Reservations_Orders
        foreign key (OrderID) references Orders,
    constraint Reservations_ReservationsRequirement
        foreign key (RequirementID) references ReservationRequirements,
    constraint Reservations_Restaurant
        foreign key (RestaurantID) references Restaurants,
    constraint ReservationsCheck
        check ([StartDate] <= [EndDate])
)
go

create index IDXReservations_Date
    on Reservations (StartDate, EndDate)
go

create index IDXReservations_OrderID
    on Reservations (OrderID)
go

create index IDXReservations_RequirementID
    on Reservations (RequirementID)
go

create index IDXReservations_CustomerID
    on Reservations (CustomerID)
go

create index IDXReservations_TableID
    on Reservations (TableID)
go

create index IDXReservations_RestaurantID
    on Reservations (RestaurantID)
go
```

3.16. Restaurants

Przechowuje informacje o restauracjach
RestaurantID – identyfikator restauracji, założony indeks klastrowany
RestaurantName – nazwa restauracji
Address – adres restauracji
City – miejscowość w której jest restauracja
PostalCode – kod pocztowy restauracji

```
create table Restaurants
(
    RestaurantID   int identity,
    RestaurantName varchar(40),
    Address        varchar(40),
    City           varchar(30),
    PostalCode     varchar(30),
    constraint RestaurantPK
        primary key (RestaurantID),
    constraint RestaurantCheck
        check ([PostalCode] like '%-%')
)
go
```

3.17. Tables

Przechowuje informacje o stolikach w restauracjach
Referencje tabeli do Restaurants(RestaurantID)
TableID – identyfikator stolika, założony indeks klastrowany
RestaurantID – identyfikator restauracji w której jest stolik, założony indeks nieklastrowany
isAvailable – informacja czy stolik nie jest wyłączony z użytku

```
create table Tables
(
    TableID      int identity,
    RestaurantID int           not null,
    IsAvailable  bit default 1 not null,
    constraint TablesPK
        primary key (TableID),
    constraint Tables_Restaurant
        foreign key (RestaurantID) references Restaurants,
)
go

create index IDXTables_RestaurantID
    on Tables (RestaurantID)
go
```

4. Widoki

4.1. v_AllNeededProductsForMenus

Ten widok, który pokazuje listę wszystkich produktów, które są potrzebne, dla wszystkich menu.

```
CREATE VIEW [dbo].[v_AllNeededProductsForMenus]
AS
SELECT DISTINCT DD.ProductID
from MenuDetails MD
	INNER JOIN DishDetails DD on MD.DishID = DD.DishID
go
```

4.2. v_CountOfDishesOrdered

Ten widok zwraca listę ile razy dane danie było zamawiane.

```
create view [dbo].[v_CountOfDishesOrdered]
as
select DishID 'DishID', COUNT(OrderID) 'Counted occurrences'
from OrderDetails
group by DishID
go
```

4.3. v_CurrentReservations

Ten widok zwraca listę dzisiejszych rezerwacji.

```
create view [dbo].[v_CurrentReservations]
as
select *
from Reservations
where day(getdate()) = day(StartDate)
go
```

4.4. v_NotAvailableProductsForMenus

Ten widok zwraca listę produktów, których nie ma w magazynie, a były używane w menu.

```
CREATE VIEW [dbo].[v_NotAvailableProductsForMenus]
AS
SELECT MD.MenuID, DD.ProductID
from MenuDetails MD
	INNER JOIN DishDetails DD ON MD.DishID = DD.DishID
	INNER JOIN (select ProductID from Magazine where UnitsInStock = 0) NA ON DD.ProductID =
NA.ProductID
group by MenuID, DD.ProductID
go
```

4.5. v_Orders

Widok, który zwraca listę wszystkich zamówień.

```
create view [dbo].[v_Orders]
as
select *
from Orders
go
```

5. Procedury

5.1. AddCategory

Procedura, która pozwala dodać do tabeli Category.

```
CREATE PROCEDURE [dbo].[addCategory]
   @CategoryName varchar(30),
   @Description varchar(200)=null
AS BEGIN
   SET NOCOUNT ON

   BEGIN TRY
   BEGIN TRANSACTION
      IF @CategoryName IS NULL OR LEN(@CategoryName)<3
         THROW 51000, '@CategoryName is null or too short',1
      INSERT INTO Categories
      values(@CategoryName,@Description)
   COMMIT TRANSACTION
   end try
   begin catch
      ROLLBACK TRANSACTION
      THROW
   end catch
end
go
```

5.2. AddCustomer

Procedura, która dodaje do tabeli Customer, sprawdzając przedtem, czy email jest prawdziwy.

```
CREATE PROCEDURE [dbo].[addCustomer]
   @Name varchar(50),
   @Email varchar(30),
   @IsCompany bit=FALSE
AS
BEGIN
   SET NOCOUNT ON

   BEGIN TRY
   BEGIN TRANSACTION
      IF @Email NOT LIKE '^([a-zA-Z0-9_\-\.]+)@([a-zA-Z0-9_\-\.]+)\.([a-zA-Z]{2,5})$'
         THROW 51000,'Provide real email address',1
   INSERT Customers
   values (@IsCompany,@Name,@Email)
   COMMIT TRANSACTION
   END TRY
   BEGIN CATCH
      ROLLBACK TRANSACTION
      THROW
   end catch
END
go
```

5.3. AddCustomerDiscount

Procedura, która dodaje do tabeli CustomerDiscount.

```
CREATE PROCEDURE [dbo].[addCustomerDiscount]
   @CustomerID int,
   @DiscountID int,
   @StartDate date
AS BEGIN
   SET NOCOUNT ON

   BEGIN TRY
   BEGIN TRANSACTION
      IF  (select CustomerID from Customers where CustomerID=@CustomerID) IS NULL
         THROW 51000,'Nie ma klienta o takim CustomerID',1
      IF  (select DiscountID from Discounts where DiscountID=@DiscountID) IS NULL
         THROW 51000,'Nie ma znizki o takim DiscountID',1
      INSERT INTO CustomerDiscounts
      values (@CustomerID,@DiscountID,@StartDate)
   COMMIT TRANSACTION
   END TRY
   BEGIN CATCH
      ROLLBACK TRANSACTION
      THROW
   end catch
END
go
```

5.4. AddDiscount

Procedura, która pozwala dodać kupon zniżkowy klientowi.

```
CREATE PROCEDURE [dbo].[addDiscount]
   @Percent decimal(2,2),
   @SingleUse bit,
   @IsForCompanies bit=FALSE,
   @MinimumPrevOrders int=0,
   @MinimumPrevCost int=0,
   @Description varchar(200)='',
   @Weeks int=0
AS BEGIN
   SET NOCOUNT ON
   BEGIN TRY
   BEGIN TRANSACTION
      IF @MinimumPrevOrders<1
         THROW 51000,'Number of previous orders must be greater than 0',1
      IF @MinimumPrevCost<1
         THROW 51000,'Number of previous orders cost must be greater than 0',1
      IF @Weeks<1
         THROW 51000,'Number of weeks must be greater than 0',1
      INSERT INTO Discounts
values(@Percent,@SingleUse,@IsForCompanies,@MinimumPrevOrders,@MinimumPrevCost,@Description,@
Weeks)
   COMMIT TRANSACTION
   END TRY
   BEGIN CATCH
      ROLLBACK TRANSACTION
      THROW
   end catch
end
go
```

5.5. AddDiscountForCompanies

Procedura pomocnicza używana głównie przy działaniu triggera dodającego zniżki dla danego zamówienia. Jej zadaniem jest dodać zniżki dla klienta firmy, jeśli spełnia on warunki zawarte w tabeli Discounts.

```
CREATE   PROCEDURE addDiscountForCompanies (@OrderID int, @CustomerID int, @DiscountID int )

AS
BEGIN
   declare @lastDiscount date, @duration int, @PrevCost int, @PrevOrders int
   set @lastDiscount = (select top 1 StartDate from CustomerDiscounts
                        inner join Discounts D2 on D2.DiscountID = CustomerDiscounts.DiscountID
                        where @CustomerID = CustomerDiscounts.CustomerID and D2.DiscountID = @DiscountID
                        order by StartDate desc)

   set @duration = (select Weeks from Discounts where DiscountID = @DiscountID)

   if (@lastDiscount is null or (datediff(week , getdate(),@lastDiscount)) > @duration)

      set @PrevCost = (select sum(Price) from Orders where CustomerID = @CustomerID and RealizationDate >
dateadd(week, @duration,getdate()))
      set @PrevOrders = (select count(OrderID) from Orders where CustomerID = @CustomerID and RealizationDate >
dateadd(week , @duration,getdate()))

      if  @PrevCost > (select MinimumPrevCost from Discounts where DiscountID = @DiscountID)
         and @PrevOrders > (select MinimumPrevCost from Discounts where DiscountID = @DiscountID)
         insert into CustomerDiscounts values (@CustomerID,@DiscountID,GETDATE(),default)


   else if (datediff(week , getdate(),@lastDiscount) < @duration)
      set @PrevCost = (select sum(Price) from Orders where CustomerID = @CustomerID and (RealizationDate
between @lastDiscount and getdate()))
      set @PrevOrders = (select count(OrderID) from Orders where CustomerID = @CustomerID and (RealizationDate
between @lastDiscount and getdate()))

      if  @PrevCost > (select MinimumPrevCost from Discounts where DiscountID = @DiscountID)
         and @PrevOrders > (select MinimumPrevCost from Discounts where DiscountID = @DiscountID)
         insert into CustomerDiscounts values
(@CustomerID,@DiscountID,dateadd(week,@duration,@lastDiscount),default)

end
go
```

5.6. AddDish

Procedura, która pozwala dodać nowe danie do tabeli Dishes

```
CREATE PROCEDURE [dbo].[addDish] @DishName varchar(30),@UnitPrice decimal(8,2),@IsVegan bit=0
AS BEGIN
  SET NOCOUNT ON

  BEGIN TRY
  BEGIN TRANSACTION
    IF @DishName IS NULL OR LEN(@DishName)<3
      THROW 51000,'@DishName is null or too short',1
    INSERT Dishes
    values (@DishName,@UnitPrice,@IsVegan)
  COMMIT TRANSACTION
  END TRY
  BEGIN CATCH
    ROLLBACK TRANSACTION
    THROW
  end catch
end
go
```

5.7. AddDishDetails

Procedura, która pozwala uzupełnić szczegóły dla danego dania w tabeli DishDetails.

```
CREATE PROCEDURE [dbo].[addDishDetails] @DishID int, @ProductID int
AS BEGIN
  SET NOCOUNT ON
  BEGIN TRY
  BEGIN TRANSACTION
    IF(select DishID from Dishes where DishID=@DishID) IS NULL
      THROW 51000,'There is no such dish',1
    IF(select ProductID from Products where ProductID=@ProductID) IS NULL
      THROW 51000,'There is no such product',1
    INSERT INTO DishDetails
    values (@DishID,@ProductID)
  COMMIT TRANSACTION
  END TRY
  BEGIN CATCH
    ROLLBACK TRANSACTION
    THROW
  end catch
end
go
```

5.8. AddEmployee

Procedura, która pozwala dodać nowego pracownika w tabeli Employees.

```
CREATE PROCEDURE [dbo].[addEmployee]
   @FirstName varchar(30),
   @LastName varchar(30),
   @Title varchar(30)=null,
   @Address varchar(40)=null,
   @City varchar(30)=null,
   @PostalCode varchar(30)=null
AS BEGIN
   SET NOCOUNT ON

   BEGIN TRY
   BEGIN TRANSACTION
      IF LEN(@FirstName)<3
         THROW 51000,'@FirstName is too short',1
      IF LEN(@LastName)<3
         THROW 51000,'@LastName is too short',1
      INSERT Employess
      values (@FirstName,@LastName,@Title,@Address,@City,@PostalCode)
   COMMIT TRANSACTION
   END TRY
   BEGIN CATCH
      ROLLBACK TRANSACTION
      THROW
   end catch
end
go
```

5.9. AddMenu

Procedura, która pozwala dodać nowe Menu, z domyślnie ustawioną wartością time na ‘n’.

```
CREATE PROCEDURE [dbo].[addMenu]
   @StartDate date,
   @EndDate date,
   @RestaurantID int,
   @Time varchar(1)='n'
AS BEGIN
   SET NOCOUNT ON

   BEGIN TRY
   BEGIN TRANSACTION
      IF @EndDate<getdate()
         THROW 51000,'You cant add menu into past',1
      IF (select RestaurantID from Restaurants where RestaurantID=@RestaurantID) IS NULL
         THROW 51000,'There is no such restaurant',1
      INSERT [Menu]
      values (@StartDate,@EndDate,@RestaurantID,@Time)
   COMMIT TRANSACTION
   END TRY
   BEGIN CATCH
      ROLLBACK TRANSACTION
      THROW
   end catch
end
go
```

5.10. AddMenuDetails

Procedura, która dodaje informacje o daniu, które ma zostać dodane do menu.

```
CREATE PROCEDURE [dbo].[addMenuDetails] @MenuID int,
                        @DishID int
AS
BEGIN
  SET NOCOUNT ON

  BEGIN TRY
    BEGIN TRANSACTION
      IF (SELECT MenuID from Menu where MenuID = @MenuID) IS NULL
        THROW 51000,'There is no such menu',1
      IF (SELECT DishID from Dishes where DishID = @DishID) IS NULL
        THROW 51000,'There is no such dish',1
      IF @DishID IN (select MD.DishID
            from [dbo].showMenusInLastMonth() MLM
                inner join MenuDetails MD on MLM.MenuID = MD.MenuID)
        THROW 51000,'This dish was in menu during this month',1
      INSERT [MenuDetails]
      values (@MenuID, @DishID)
    COMMIT TRANSACTION
  END TRY
  BEGIN CATCH
    ROLLBACK TRANSACTION
    THROW
  end catch
end
go
```

5.11. AddOrder

Procedura, która dodaje najważniejsze informacje o zamówieniu.

```
CREATE PROCEDURE [dbo].[addOrder]
  @CustomerID int,
  @EmployeeID int,
  @RestaurantID int,
  @isOnline bit=0
AS BEGIN
  SET NOCOUNT ON

  BEGIN TRY
  BEGIN TRANSACTION
    IF @CustomerID IS NOT NULL AND (SELECT CustomerID from Customers where CustomerID=@CustomerID) IS NULL
      THROW 51000,'There is no such customer',1
    IF (SELECT EmployeeID from Employess where EmployeeID=@EmployeeID) IS NULL
      THROW 51000,'There is no such employee',1
    IF (SELECT RestaurantID from Restaurants where RestaurantID=@RestaurantID) IS NULL
      THROW 51000,'There is no such restaurant',1
    INSERT INTO Orders(CustomerID,EmployeeID,RestaurantID,isOnline)
    values (@CustomerID,@EmployeeID,@RestaurantID,@isOnline)
  COMMIT TRANSACTION
  END TRY
  BEGIN CATCH
    ROLLBACK TRANSACTION
    THROW
  end catch
end
go
```

5.12. addOrderDetails

Procedura, która dodaje informacje o zamówieniu (poszczególne dania) do tabeli OrderDetails

```
CREATE PROCEDURE [dbo].[addOrderDetails]
   @OrderID int,
   @DishID int
AS BEGIN
   SET NOCOUNT ON

   BEGIN TRY
   BEGIN TRANSACTION
      IF(SELECT OrderID from Orders where OrderID=@OrderID)IS NULL
         THROW 51000,'There is no such order',1
      IF(SELECT DishID from Dishes where DishID=@DishID) IS NULL
         THROW 51000,'There is no such dish',1
      IF EXISTS (SELECT * FROM Dishes inner join DishDetails DD on Dishes.DishID = DD.DishID
                        inner join Products P on P.ProductID = DD.ProductID
                        inner join Categories C on C.CategoryID = P.CategoryID
                        WHERE CategoryName = 'Seafood')
         IF (SELECT OrderDate FROM Orders WHERE OrderID = @OrderID) < dateadd(week,1,getdate())
            THROW 51000, 'It is to late to order seafood',1

      INSERT [OrderDetails]
      values (@OrderID, @DishID)
   COMMIT TRANSACTION
   END TRY
   BEGIN CATCH
      ROLLBACK TRANSACTION
      THROW
   end catch
end
```

5.13. AddProduct

Procedura, która dodaje produkt do tabeli Products

```
CREATE PROCEDURE [dbo].[addProduct]
   @ProductName varchar(30),
   @CategoryID int
AS BEGIN
   SET NOCOUNT ON

   BEGIN TRY
   BEGIN TRANSACTION
      IF LEN(@ProductName)<3
         THROW 51000,'Provide valid name', 1
      IF (select CategoryID from Categories where CategoryID=@CategoryID) IS NULL
         THROW 51000,'There is no such category',1
      INSERT Products
      values (@ProductName,@CategoryID)
   COMMIT TRANSACTION
   END TRY
   BEGIN CATCH
      ROLLBACK TRANSACTION
      THROW
   end catch
end
go
```

5.14. AddProductToMagazine

Procedura, która dodaje lub odejmuje aktualną ilość produktu w magazynie, tzn można dodać ujemną liczbę, a wtedy wartość liczbowa odpowiadająca za ilość przechowywanego towaru ulegnie zmniejszeniu.

```
CREATE PROCEDURE [dbo].[addProductToMagazine]
   @RestaurantID int,
   @ProductID int,
   @UnitsInStock int
AS BEGIN
   SET NOCOUNT ON

   BEGIN TRY
   BEGIN TRANSACTION
      DECLARE @Quantity int

      IF(SELECT RestaurantID from Restaurants where RestaurantID=@RestaurantID)IS NULL
         THROW 51000,'There is no such restaurant',1
      IF(select ProductID from Products where ProductID=@ProductID) IS NULL
         THROW 51000,'There is no such product',1
      SET @Quantity = (select UnitsInStock from Magazine where RestaurantID=@RestaurantID and
ProductID=@ProductID)
      IF @Quantity IS NOT NULL
         IF @Quantity-@UnitsInStock<0
            THROW 51000, 'You are decreasing more Units than there are in magazine',1
         ELSE
            UPDATE Magazine
            SET UnitsInStock=@Quantity+@UnitsInStock where RestaurantID=@RestaurantID and
ProductID=@ProductID
      ELSE
         IF @UnitsInStock<0
            THROW 51000, 'You can not decrease product that was not available before',1
         ELSE
            UPDATE Magazine
            SET UnitsInStock=@UnitsInStock where RestaurantID=@RestaurantID and ProductID=@ProductID
   COMMIT TRANSACTION
   END TRY
   BEGIN CATCH
      ROLLBACK TRANSACTION
      THROW
   end catch
end
go
```

5.15. AddReservation

Procedura, która dodaje do bazy dane odnośnie rezerwacji.

```
CREATE PROCEDURE [dbo].[addReservation]
   @RestaurantID int,
   @TableID int,
   @StartDate datetime,
   @EndDate datetime,
   @CustomerID int,
   @OrderID int,
   @RequirementID int
AS BEGIN
   SET NOCOUNT ON

   BEGIN TRY
   BEGIN TRANSACTION
      IF (select RestaurantID from Restaurants where RestaurantID=@RestaurantID) IS NULL
         THROW 51000,'There is no such restaurant',1
      IF (select TableID from Tables where TableID=@TableID) IS NULL
         THROW 51000,'There is no such table',1
      IF (select IsAvailable from Tables where TableID=@TableID and IsAvailable=0) IS NOT NULL
         THROW 51000,'You have selected unavailable table',1
      IF (select CustomerID from Customers where CustomerID=@CustomerID) IS NULL
         THROW 51000, 'There is no such customer',1
      IF (select RequirementID from ReservationRequirements where RequirementID=@RequirementID)IS
NULL
         THROW 51000, 'There is no requirement with such ID',1
      INSERT INTO Reservations
      VALUES (@RestaurantID,@TableID,@StartDate,@EndDate,@CustomerID,0,@RequirementID,@OrderID)
   COMMIT TRANSACTION
   END TRY
   BEGIN CATCH
      ROLLBACK TRANSACTION
      THROW
   end catch
end
go
```

5.16. AddReservationRequirement

Procedura, dzięki której do tabeli ReservationRequirements możemy dodać kolejne wymagania dla rezerwacji.

```
CREATE PROCEDURE [dbo].[addReservationRequirement]
   @MinimumOrderCost int=0,
   @MinimumPrevOrders int=0,
   @MinimumPrevCost int=0
AS
BEGIN
   SET NOCOUNT ON

   BEGIN TRY
   BEGIN TRANSACTION
      IF @MinimumPrevOrders<0 or @MinimumPrevCost<0 or @MinimumOrderCost<0
         THROW 51000,'Provide positive numbers',1
      INSERT ReservationRequirements
      values (@MinimumOrderCost,@MinimumPrevOrders,@MinimumPrevCost)
   COMMIT TRANSACTION
   END TRY
   BEGIN CATCH
      ROLLBACK TRANSACTION
      THROW
   end catch
end
go
```

5.17. AddRestaurant

Procedura, która dodaje do bazy kolejną restaurację wraz z jej podstawowymi danymi.

```
CREATE PROCEDURE [dbo].[addRestaurant]
   @RestaurantName varchar(40),
   @Address varchar(40),
   @City varchar(30),
   @PostalCode varchar(30)
AS BEGIN
   SET NOCOUNT ON
      INSERT Restaurants
      values (@RestaurantName,@Address,@City,@PostalCode)
end
go
```

5.18. AddTable

Procedura, która dodaje do tabeli Tables stolik dla danej restauracji.

```
CREATE PROCEDURE [dbo].[addTable]
   @RestaurantID int,
   @IsAvailable bit=1
AS BEGIN
   SET NOCOUNT ON

   BEGIN TRY
   BEGIN TRANSACTION
      IF(select RestaurantID from Restaurants where RestaurantID=@RestaurantID)IS NULL
         THROW 51000,'There is no such restaurant',1
      INSERT Tables
      values (@RestaurantID,@IsAvailable)
   COMMIT TRANSACTION
   END TRY

   begin catch
      ROLLBACK TRANSACTION
      THROW
   end catch
end
go
```

5.19. ChangeTablesAvailability

Procedura, dzięki której można zmienić dostępność danego stolika.

```
CREATE PROCEDURE [dbo].[changeTablesAvailability] @TableID int, @availability bit
AS
BEGIN
   BEGIN TRY
      BEGIN TRANSACTION
         SET NOCOUNT ON

         IF (select TableID from Tables where TableID = @TableID) IS NULL
            THROW 51000,'There is no such table',1
         UPDATE Tables
         SET IsAvailable=@availability
         WHERE TableID=@TableID
      COMMIT TRANSACTION
   end try
   begin catch
      ROLLBACK TRANSACTION
      THROW
   end catch
end
go
```

5.20. ComputeDiscountedPrice

Procedura oblicza cenę zamówienia z uwzględnieniem rabatów. Pobiera aktualne rabaty z tabeli CustomerDiscounts i modyfikuje wartość funkcji computePrice(cenę bez rabatów).

```
CREATE OR ALTER PROCEDURE computeDiscountedPrice @OrderID int
AS
BEGIN
   DECLARE @CustomerID int, @IsCompany bit, @Percent decimal(8,2), @StartDate date

   set @CustomerID = (select CustomerID from Orders where OrderID = @OrderID)
   set @IsCompany = (select IsCompany from Customers where CustomerID = @CustomerID)
   set @Percent = 0
   IF (@IsCompany = 0)
      --Discount 1
      IF EXISTS (select * from CustomerDiscounts where DiscountID = 1)
        set @Percent = (select [Percent] from Discounts where DiscountID =1)
      --Discount 2
      IF EXISTS (select * from CustomerDiscounts where DiscountID = 2)
        set @Percent += (select [Percent] from Discounts where DiscountID =2)
      --Discount 3
      IF EXISTS (select * from CustomerDiscounts
                  inner join Discounts D on CustomerDiscounts.DiscountID = D.DiscountID
                  where CustomerID = @CustomerID and D.DiscountID = 3 and IsAvailable = 1 and datediff(week
,getdate(),StartDate)<Weeks)

        set @Percent += (select [Percent] from Discounts where DiscountID =2)
        update CustomerDiscounts set IsAvailable = 0 where CustomerID = @CustomerID and DiscountID = 3 and
IsAvailable = 1
      --Discount 4
      IF EXISTS (select * from CustomerDiscounts
                  inner join Discounts D on CustomerDiscounts.DiscountID = D.DiscountID
                  where CustomerID = @CustomerID and D.DiscountID = 4 and IsAvailable = 1 and datediff(week
,getdate(),StartDate)<Weeks)

        set @Percent += (select [Percent] from Discounts where DiscountID =4)
        update CustomerDiscounts set IsAvailable = 0 where CustomerID = @CustomerID and DiscountID = 4 and
IsAvailable = 1

   IF (@IsCompany = 1)
      --Discount 5
      IF EXISTS (select * from CustomerDiscounts
                  inner join Discounts D on CustomerDiscounts.DiscountID = D.DiscountID
                  where CustomerID = @CustomerID and D.DiscountID = 5 and IsAvailable = 1 and datediff(week
,getdate(),StartDate)<Weeks)

        set @StartDate = (select top 1 StartDate from CustomerDiscounts
                  inner join Discounts D on CustomerDiscounts.DiscountID = D.DiscountID
                  where CustomerID = @CustomerID and D.DiscountID = 5 and IsAvailable = 1 and datediff(week
,getdate(),StartDate)<Weeks
                  order by StartDate desc)

        set @Percent += [dbo].[countDiscountsInRowRec](@CustomerID,@StartDate,1) * (select [Percent] from Discounts
where DiscountID =6)

      --Discount 6
      IF EXISTS (select * from CustomerDiscounts
                  inner join Discounts D on CustomerDiscounts.DiscountID = D.DiscountID
                  where CustomerID = @CustomerID and D.DiscountID = 6 and IsAvailable = 1 and datediff(week
,getdate(),StartDate)<Weeks)
        set @Percent =  (select [Percent] from Discounts where DiscountID =6)

   update Orders set Price = ([dbo].[computePrice](@OrderID)*(1-@Percent)) where OrderID = @OrderID
end
go
```

5.21. ConfirmReservation

Procedura, która pozwala na szybkie sprawdzenie, czy rezerwacja spełnia wymagania i potwierdzenie jej w przypadku pozytywnej odpowiedzi funkcji pomocniczych.

```
CREATE PROCEDURE confirmReservation @ReservationID int
AS
BEGIN
   SET NOCOUNT ON
   DECLARE @bool1 bit,@bool2 bit, @OrderID int,@CustomerID int,@TableID int,@StartDate
datetime,@EndDate datetime
   set @OrderID = (select OrderID from Reservations where ReservationID = @ReservationID)
   set @CustomerID = (select CustomerID from Reservations where ReservationID = @ReservationID)
   set @bool1 = [dbo].[checkOnlineOrderRequirements]([dbo].[computePrice](@OrderID), @CustomerID)
   set @TableID = (select TableID from Reservations where ReservationID = @ReservationID)
   set @StartDate = (select StartDate from Reservations where ReservationID = @ReservationID)
   set @EndDate = (select EndDate from Reservations where ReservationID = @ReservationID)
   set @bool2 = [dbo].[checkReservationRequirements](@TableID,@StartDate,@EndDate)
   IF @bool1=1 and @bool2=1
      UPDATE Reservations
      SET IsConfirmed=1
      WHERE ReservationID = @ReservationID
   ELSE
      EXEC deleteOrderFromDatabase @OrderID
      DELETE FROM Reservations
      where ReservationID=@ReservationID
end
go
```

5.22. DeleteDishFromMenu

Procedura, która pozwala usunąć daną pozycję z menu.

```
CREATE PROCEDURE [dbo].[deleteDishFromMenu] @MenuID int, @DishID int
AS BEGIN
   DELETE FROM MenuDetails
   WHERE MenuID=@MenuID and DishID=@DishID
end
go
```

5.23. DeleteOrderFromDatabase

Procedura, która pozwala usunąć wybrane zamówienie z bazy danych.

```
CREATE PROCEDURE [dbo].[deleteOrderFromDatabase] @OrderID int
AS BEGIN
      DELETE FROM OrderDetails
      where OrderID=@OrderID

      DELETE FROM Orders
      where OrderID=@OrderID
end
go
```

5.24 FillOrderInformation

Procedura, dzięki której możemy uzupełnić informacje o zamówieniu o pola RealizationDate oraz FinePaid

```
create procedure [dbo].[fillOrderInformation] @OrderID int, @RealizationDate datetime=null, @FinePaid
decimal(6, 2)=null
as
begin
   SET NOCOUNT ON

   BEGIN TRY
      BEGIN TRANSACTION
         IF (select OrderID from Orders where OrderID = @OrderID) IS NULL
            THROW 51000,'There is no such order',1
         IF @RealizationDate IS NOT NULL
            UPDATE Orders
            SET RealizationDate=@RealizationDate
            where OrderID = @OrderID
         IF @FinePaid IS NOT NULL
            UPDATE Orders
            SET FinePaid=@FinePaid
            where OrderID = @OrderID
      COMMIT TRANSACTION
   END TRY
   BEGIN CATCH
      ROLLBACK TRANSACTION
      THROW
   end catch
end
go
```

6. Funkcje

6.1. CheckIfNewMenuIsValid

Funkcja, która sprawdza, czy dane menu spełnia wszystkie narzucone wymagania. W skrócie: porównuje pozycje między danym menu, a aktualnym menu i liczy stosunek między wspólnymi do wszystkich pozycji

```
CREATE FUNCTION [dbo].[checkIfNewMenuIsValid](@MenuID int)
   RETURNS BIT
AS
BEGIN
   DECLARE @numberOfDishes int,@numberOfCommonDishes int
   set @numberOfDishes = (select COUNT(*) from [dbo].getCurrentMenu())
   set @numberOfCommonDishes = (select COUNT(*)
                        from [dbo].getCurrentMenu() M
                              inner join [dbo].getMenuFromID(@MenuID) NM on NM.DishID = M.DishID)
   IF @numberOfCommonDishes / @numberOfDishes >= 1 / 2
      RETURN 0
   RETURN 1
end
go
```

6.2. CheckOnlineOrderRequirements

Funkcja, która oblicza cenę zamówienia i sprawdza wymagania dla prawidłowej rezerwacji online.

```
CREATE   FUNCTION [dbo].[checkOnlineOrderRequirements]
   (@OrderID int,
    @CustomerID int)
   RETURNS bit
AS
BEGIN
   DECLARE @minPrevCost int,@minOrderCost int, @minPrevOrders int

   set @minPrevCost = (select max(MinimumPrevCost) from ReservationRequirements)
   set @minOrderCost = (select max(MinimumOrderCost) from ReservationRequirements)
   set @minPrevOrders = (select max(MinimumPrevOrders) from ReservationRequirements)

   IF([dbo].computePrice (@OrderID)<@minOrderCost)
      RETURN 0
      IF(select COUNT(OrderID) from Orders where [dbo].computePrice(OrderID)>=@minPrevCost
)<@minPrevOrders
      RETURN 0
   RETURN 1
end
go
```

6.3. CheckReservationRequirements

Funkcja, która sprawdza, czy możemy zarezerwować dany stolik w wybranych godzinach.

```
CREATE FUNCTION [dbo].[checkReservationRequirements]
   (@TableID int,
   @StartDate datetime,
   @EndDate datetime)
   returns bit
AS BEGIN
   IF (SELECT ReservationID from Reservations where TableID=@TableID and @StartDate BETWEEN StartDate
AND EndDate) IS NOT NULL
      RETURN 0
   IF (SELECT ReservationID from Reservations where TableID=@TableID and @EndDate BETWEEN StartDate
AND EndDate) IS NOT NULL
      RETURN 0
   RETURN 1
end
go
```

6.4. ComputePrice

Funkcja pomocnicza, która oblicza cenę danego zamówienia.

```
CREATE FUNCTION [dbo].[computePrice](@OrderID int)
   RETURNS decimal(8, 2)
AS
BEGIN
   RETURN (select SUM(UnitPrice)
         from Dishes
         where DishID in (select DishID from OrderDetails where OrderID = @OrderID))
END
go
```

6.5. countDiscountsInRowRec

Funkcja pomocnicza triggera UpdateOrderDiscounts. Zwraca liczbę okresowych zniżek występujących po sobie.

```
CREATE FUNCTION [dbo].[countDiscountsInRowRec]
   (@CustomerID int,
   @StartDate date,
   @Counter int)
   returns int
AS BEGIN
   declare @newDate date
   IF NOT EXISTS (select StartDate from CustomerDiscounts
      where CustomerID = @CustomerID and DiscountID = 5 and datediff(month, @StartDate,StartDate) be-
tween 0 and 1)
      return @Counter

   set @newDate = (select top 1 StartDate from CustomerDiscounts
      where CustomerID = 2 and DiscountID = 5 and (datediff(month, getdate(),StartDate)<1)
      order by StartDate)

   return [dbo].[countDiscountsInRowRec](@CustomerID,@StartDate,@Counter + 1)

end
go
```

6.6. GenerateCustomerReportOrders

Funkcja, która generuje raport dla użytkownika o jego zamówieniach.

```
create function [dbo].[generateCustomerReportOrders](@CustomerID int)
  returns table
    as return
    select OrderID, RestaurantName, RealizationDate, Price
    from Orders
        inner join Restaurants R2 on Orders.RestaurantID = R2.RestaurantID
    where Orders.CustomerID = @CustomerID
go
```

6.7. GenerateInvoiceFromOneMonth

Funkcja, która generuję fakturę z całego miesiąca.

```
create function [dbo].[generateInvoiceFromOneMonth](@CustomerID int)
  returns table
    as return select O.OrderID, C.Name, O.RestaurantID, O.Price, O.isOnline, O.OrderDate
        from Orders O
            inner join Customers C on C.CustomerID = O.CustomerID
        where O.CustomerID = @CustomerID
          and C.IsCompany = 1
          and datediff(month, O.OrderDate, getdate()) < 1
go
```

6.8. GenerateInvoiceOfOneOrder

Funkcja, która generuje fakturę na dane zamówienie.

```
create function [dbo].[generateInvoiceOfOneOrder](@CustomerID int, @OrderID int)
  returns table
    as return select OrderID,C.Name, RestaurantID, Price, isOnline, OrderDate
        from Orders
            inner join Customers C on Orders.CustomerID = C.CustomerID
        where Orders.CustomerID = @CustomerID
          and Orders.OrderID = @OrderID
          and C.IsCompany = 1
go
```

6.9. GenerateMonthReportClient

Funkcja, która generuje informacje dla klient odnośnie zamówien z jednego miesiąca, wraz z informacją, czy zostały użyte rabaty.

```
create   function [dbo].[generateMonthReportClient](@CustomerID int)
  returns table
    as return select OrderID, [dbo].computePrice(OrderID) as 'original price', Price as 'price after
discounts',OrderDate
        from Orders
        where CustomerID = @CustomerID
          and DATEDIFF(month, RealizationDate, getdate()) < 1
go
```

6.10. GenerateMonthReportDiscounts

Funkcja, która zwraca informacje o aktualnie dostępnych rabatach klientów, które są nie starsze niż miesiąc.

```
create function [dbo].[generateWeekReportDiscounts]()
  returns table
    as return select CustomerID, DiscountID, StartDate
          from CustomerDiscounts
          where datediff(week, StartDate, getdate()) < 1
           and IsAvailable = 1
go
```

6.11. GenerateMonthReportMenus

Funkcja, która zwraca tabelę informacji o tym jakie menu, kiedy i z jakimi potrawami się pojawiły.

```
create function [dbo].[generateMonthReportMenus]()
  returns table
    as return select M.MenuID, M.StartDate, MD.DishID
          from Menu M
               inner join MenuDetails MD on M.MenuID = MD.MenuID
          where datediff(month, StartDate, getdate()) < 1
go
```

6.12. GenerateMonthReportTables

Funkcja, która zwraca informacje o tym, jaki stolik, kiedy i przez jakiego klienta był rezerwowany.

```
create function [dbo].[generateMonthReportTables]()
  returns table
    as return select TableID, StartDate, CustomerID
          from Reservations
          where datediff(month , StartDate, getdate()) < 1
go
```

6.13. GenerateWeekReportClient

Tygodniowy odpowiednik wcześniejszej funkcji.

```
create   function [dbo].[generateWeekReportClient](@CustomerID int)
  returns table
    as return select OrderID, [dbo].computePrice(OrderID) as 'original price', Price as 'price after
discounts',OrderDate
          from Orders
          where CustomerID = @CustomerID
           and DATEDIFF(week, RealizationDate, getdate()) < 1
go
```

6.14. GenerateWeekReportDiscounts

Tygodniowy odpowiednik wcześniejszej funkcji.

```
create function [dbo].[generateWeekReportDiscounts]()
  returns table
    as return select CustomerID, DiscountID, StartDate
          from CustomerDiscounts
          where datediff(week, StartDate, getdate()) < 1
           and IsAvailable = 1
go
```

6.15. GenerateWeekReportMenus

Tygodniowy odpowiednik wcześniejszej funkcji.

```
create function [dbo].[generateWeekReportMenus]()
  returns table
    as return select M.MenuID, M.StartDate, MD.DishID
          from Menu M
               inner join MenuDetails MD on M.MenuID = MD.MenuID
          where datediff(week, StartDate, getdate()) < 1
go
```

6.16. GenerateWeekReportTables

Tygodniowy odpowiednik wcześniejszej funkcji.

```
create function [dbo].[generateWeekReportTables]()
  returns table
    as return select TableID, StartDate, CustomerID
          from Reservations
          where datediff(week, StartDate, getdate()) < 1
go
```

6.17. GetCurrentMenu

Funkcja pomocnicza, która zwraca aktualnie obowiązujące menu

```
CREATE
  FUNCTION [dbo].[getCurrentMenu]()
  RETURNS TABLE
    AS
    RETURN SELECT MenuID,DishID
        from MenuDetails
        where MenuID = (select MenuID from Menu where Time = 'n')
go
```

6.18. GetMenuFromID

Funkcja pomocnicza, która zwraca pozycje danego menu.

```
CREATE FUNCTION [dbo].[getMenuFromID](@MenuID int)
RETURNS TABLE
AS RETURN SELECT * from MenuDetails where MenuID=@MenuID
go
```

6.19. ShowCustomerDetails

Funkcja, która zwraca informacje o danym kliencie.

```
create function [dbo].[showCustomerDetails](@CustomerID int)
returns table
as return select * from Customers where CustomerID=@CustomerID
go
```

6.20. ShowCustomerReservations

Funkcja, która zwraca rezerwacje danego klienta.

```
create function [dbo].[showCustomerReservations](@CustomerID int)
  returns table
    as return select *
          from Reservations
          where CustomerID = @CustomerID
           and StartDate >= getdate()
go
```

6.21. ShowCustomersOrdersNumberDetails

Funkcja, która zwraca liczbę zamówien oraz ich łączną wartość dla danego klienta.

```
create function [dbo].[showCustomersOrdersNumberDetails](@CustomerID int)
  returns table
    as return select count(*) 'number of orders', sum(Price) 'sum of price of orders'
          from Orders
          where CustomerID = @CustomerID
go
```

6.22. ShowCustomersOrdersNumberDetailsLastMonth

Odpowiednik poprzedniej funkcji, z ograniczeniem do zeszłego miesiąca

```
create function [dbo].[showCustomersOrdersNumberDetailsLastMonth](@CustomerID int)
  returns table
    as return select count(*) 'number of orders', sum(Price) 'sum of price of orders'
          from Orders
          where CustomerID = @CustomerID and datediff(month,RealizationDate,getdate())<1
go
```

6.23. showCustomersOrdersNumberDetailsLastQuarter

Odpowiednik wcześniejszej funkcji z ograniczeniem do zeszłego kwartału

```
create  function [dbo].[showCustomersOrdersNumberDetailsLastQuarter](@CustomerID int)
  returns table
    as return select count(*) 'number of orders', sum(Price) 'sum of price of orders'
          from Orders
          where CustomerID = @CustomerID and datediff(month,RealizationDate,getdate())<3
go
```

6.24. ShowDishIngredients

Funkcja pomocnicza, która zwraca komponenty danego dania.

```
CREATE FUNCTION [dbo].[showDishIngredients](@DishID int)
  RETURNS TABLE
    AS RETURN
    select ProductName
    from Products P
        INNER JOIN DishDetails DD on P.ProductID = DD.ProductID
    where DD.DishID = @DishID
go
```

6.25. ShowFutureOrdersWithDish

Funkcja, która zwraca przyszłe zamówienia, które zarezerwowały dane danie.

```
create function [dbo].[showFutureOrdersWithDish](@DishID int)
   returns table
      as return select O.OrderID
            from Orders O
                  inner join OrderDetails OD on O.OrderID = OD.OrderID
            where OD.DishID = @DishID
             and RealizationDate is null
go
```

6.26. ShowMenusInLastMonth

Funkcja, która zwraca menu z ostatniego miesiąca

```
CREATE FUNCTION [dbo].[showMenusInLastMonth]()
RETURNS TABLE
AS RETURN
   select MenuID from Menu where StartDate<=getdate() and datediff(month,StartDate,getdate())<=30
go
```

6.27. ShowMenusWithDish

Funkcja, która zwraca wszystkie menu, które miały w sobie dane danie.

```
CREATE FUNCTION [dbo].[showMenusWithDish](@DishID int)
returns table
as return
   select MenuID from MenuDetails where DishID=@DishID
go
```

6.28. ShowRestaurantTables

Funkcja, która zwraca wszystkie stoliki dla danej restauracji.

```
CREATE FUNCTION [dbo].[showRestaurantTables] (@RestaurantID int)
RETURNS TABLE
AS
RETURN
   SELECT TableID,IsAvailable from Tables where RestaurantID=@RestaurantID
go
```

7. Triggery

7.1. UpdateMagazine

Trigger odpowiada za aktualizacje stanu magazynu. W momencie dodania zamówienia pobiera z magazynu określoną ilość danych produktów, potrzebnych do przygotowania dań wchodzących w skład zamówienia.

```
create   trigger UpdateMagazine
on Orders after update as
begin
   declare @OrderID int, @RestaurantID int, @Quantity int, @Counter int, @ProductID int
   set @OrderID = (select OrderID FROM inserted)
   set @RestaurantID = (select RestaurantID FROM inserted)


   select ProductID as ProductID, sum(Quantity) as Quantity into #temporary
   from DishDetails
                  inner join Dishes D on D.DishID = DishDetails.DishID
                  inner join OrderDetails OD on D.DishID = OD.DishID
                  inner join Orders O on O.OrderID = OD.OrderID
                  where O.OrderID = @OrderID
                  group by ProductID

   select @Counter = (-1) + count(*) from #temporary

   while @Counter >= 0
   begin
      set @ProductID = (select ProductID from #temporary order by ProductID offset @Counter rows fetch
next 1 rows only )
      set @Quantity = (select Quantity from #temporary order by ProductID offset @Counter rows fetch next 1
rows only )
      update Magazine set UnitsInStock -= @Quantity where ProductID = @ProductID
      set @Counter -=1
   end

   If(OBJECT_ID('tempdb..#temporary') Is Not Null)
      drop table #temporary

end
go
```

7.2. UpdateCustomerDiscounts

Trigger odpowiada za aktualizowanie rabatów w tabeli CustomerDiscounts. Rozpoczyna działanie gdy zmieni się kolumna Price tabeli Orders, to znaczy w momencie finalizacji usługi. Przypisuje klientowi zniżki sprawdzając czy spełnia określone warunki

```
on Orders after update as
begin
  declare @PrevCost int, @PrevOrders int, @CustomerID int, @OrderID int, @lastDiscount date, @duration
int,@DiscountDate date
  set @CustomerID = (select CustomerID FROM inserted)
  set @OrderID = (select OrderID FROM inserted)
  set @PrevCost = (select sum(Price) from Orders where CustomerID=@CustomerID)
  set @PrevOrders = (select sum(OrderID) from Orders where CustomerID=@CustomerID)
  set @DiscountDate = getdate()

  if update(Price)

    if (select IsCompany from Customers where CustomerID = @CustomerID) = 0
      --Discount 1
      if @PrevCost > (select MinimumPrevCost from Discounts where DiscountID =1) and
        @PrevOrders > (select MinimumPrevOrders from Discounts where DiscountID =1)
        exec addCustomerDiscount @CustomerID,1,@DiscountDate


      --Discount 2
      if @PrevCost > (select sum(MinimumPrevCost) from Discounts where DiscountID in(1,2)) and
        @PrevOrders > (select sum(MinimumPrevOrders) from Discounts where DiscountID in (1,2))
        exec addCustomerDiscount @CustomerID,2,@DiscountDate

      --Discount 3
      if @PrevCost > (select sum(MinimumPrevCost) from Discounts where DiscountID =3)
        and not EXISTS (select * from CustomerDiscounts where @CustomerID = CustomerID and DiscountID
= 3)
        insert into CustomerDiscounts values (@CustomerID,3,GETDATE(),default)
        exec addCustomerDiscount @CustomerID,3,@DiscountDate

      --Discount 4
      if @PrevCost > (select sum(MinimumPrevCost) from Discounts where DiscountID =4)
        and not EXISTS (select * from CustomerDiscounts where @CustomerID = CustomerID and DiscountID
= 4)
        exec addCustomerDiscount @CustomerID,4,@DiscountDate

    --For Companies
    if (select IsCompany from Customers where CustomerID = @CustomerID) = 1

      --Discount 5
      exec addDiscountForCompanies @OrderID = @OrderID,@CustomerID = @CustomerID, @DiscountID = 5

      --Discount 6
      exec addDiscountForCompanies @OrderID = @OrderID,@CustomerID = @CustomerID, @DiscountID = 6


end
go
```

8. Indeksy

Oprócz indeksów tworzonych automatycznie w związku z poleceniami CONSTRAINT PRIMARY KEY i CONSTRAINT UNIQUE, stworzone zostały indeksy nieklastrowe dla:

- wszystkich kluczy obcych
- kolumn danych często występujących w warunku WHERE, takich jak DishName, FirstName, LastName, ProductName itd..
- kolumn danych często występujących w klauzuli ORDER BY, GROUP BY, takich jak StartDate, EndDate itd..

Polecenia inicjalizujące indeksy są widoczne w sekcji 3.Tabele.

9. Funkcjonalności systemu

Nasz system bazy danych spełnia następujące funkcjonalności:

- złożenie zamówienia - na miejscu i na wynos (klient)
- rezerwacja stolika z wcześniejszym złożeniem zamówienia (klient)
- dodawanie informacji o dacie realizacji oraz zapłaconej kwocie zamówienia (obsługa)
- sprawdzenie warunków poprawnej rezerwacji i zamówienia oraz ich zatwierdzenie (obsługa)
- możliwość dodawania warunków rezerwacji, ograniczeń na zamówienia, zniżek oraz warunków ich przyznawania (manager)
- możliwość usunięcia zamówienia, rezerwacji (manager)
- zmiana dostępności stolika w restauracji (manager)
- przyznawanie klientom postępowych zniżek, tj. po zamówieniu sprawdzane jest, czy klient spełnia warunki na jakąś zniżkę
- sprawdzenie historii menu oraz aktualnie dostępnego menu (obsługa)
- usunięcie pozycji w menu (manager)
- sprawdzenie, czy dane menu spełnia warunki poprawności (manager)
- dodanie rabatu do zamówienia na prośbę klienta (w przypadku kuponów jednorazowych) oraz automatyczne dodawanie stałych zniżek (obsługa/system)
- wyświetlanie rabatów w danej restauracji (klient, obsługa)
- wyświetlanie historii zamówień klienta wraz z kwotami (klient, obsługa)
- obliczanie ile zamówień oraz na jaką łączną kwotę zostało złożonych przez klienta w ciągu miesiąca, kwartału lub od pierwszego zamówienia (obsługa)
- wyświetlanie przyszłych zamówień z daną pozycją z menu (obsługa)
- wyświetlanie informacji o zamówieniach - kto, kiedy, co (manager)
- generowanie danych do faktury do zamówienia lub zbiorczej - dla klienta indywidualnego i firmy (obsługa)
- generowanie raportów dotyczących zamówień, rezerwacji, rabatów i klientów (obsługa)
- dodawanie produktów do magazynu i zmienianie ilości przechowywanych towarów (obsługa)
- wyświetlanie aktualnego menu (klient, obsługa)

10. Uprawnienia do korzystania z danych

Wyróżniamy 3 role określające uprawnienia do danych

10.1. Klient

Ma dostęp do danych w restauracji, której jest klientem takich jak:

10.1.1. Podgląd aktualnego Menu, wraz z cenami
10.1.2. Podgląd na przyznane oraz oferowane zniżki
10.1.3. Podgląd składników dań
10.1.4. Podgląd złożonego przez niego zamówienia lub rezerwacji

10.2. Obsługa

Ma dostęp do danych w restauracji w której pracuje takich jak:

10.2.1. Podgląd obsługiwanych zamówień
10.2.2. Podgląd statusu stolików
10.2.3. Podgląd dostępnych rabatów
10.2.4. Podgląd aktualnego Menu
10.2.5. Podgląd składników dań
10.2.6. Podgląd rezerwacji

10.3. Manager

Ma dostęp do wszystkich danych.

11. Generator

W projekcie wykorzystaliśmy SQL Data Generator firmy RedGate (*http://www.redgate.com/products/sql-development/sql-data-generator*) do wygenerowania danych. Część nieregularnych danych zostało dodane ręcznie lub wygenerowane za pomocą skryptu w języku Python. Zostały wygenerowane dane odpowiadające 3 letniej działalności firmy.