Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie
Wydział Informatyki, Elektroniki i Telekomunikacji

Projekt bazy danych dla restauracji - etap 5

Szymon Gołębiowski
Dominika Bocheńczyk
Michał Gniadek

11 stycznia 2022

Spis treści

1 Działanie systemu

1.1 Funkcje dla klientów

- Złożenie zamówienia na wynos (przez internet; z limitem czasu do kiedy ustalone jest menu)
- Rezerwacja stolika (przez internet) + ew. złożenie zamówienia (z limitem czasu do kiedy ustalone jest menu) - system weryfikuje czy w podanym czasie i dla podanej liczby osób jest miejsce, a w przypadku zamówienia czy dane danie jest dostępne (a dla owoców morza czy zamówienie jest składane z odpowiednim wyprzedzeniem)
- Anulowanie rezerwacji i zamówienia
- Sprawdzenie statusu rezerwacji
- Wygenerowanie faktury za pojedyncze zamówienie i zbiorczej za cały miesiąc
- Przeglądanie historii zamówień i dostępnych rabatów i wygenerowanie raportów z historią

1.2 Funkcje dla obsługi

- Podgląd aktualnych zamówień
- Akceptacja oczekujących zamówień
- W przypadku kiedy klient zamawia na miejscu, możliwość wprowadzenia do systemu zamówienia (również na wynos) i zajęcia stolika
- Zmiana informacji dotyczących zamówienia i rezerwacji (np. jeśli klient wyjdzie wcześniej)
- Wygenerowanie faktury za zamówienie

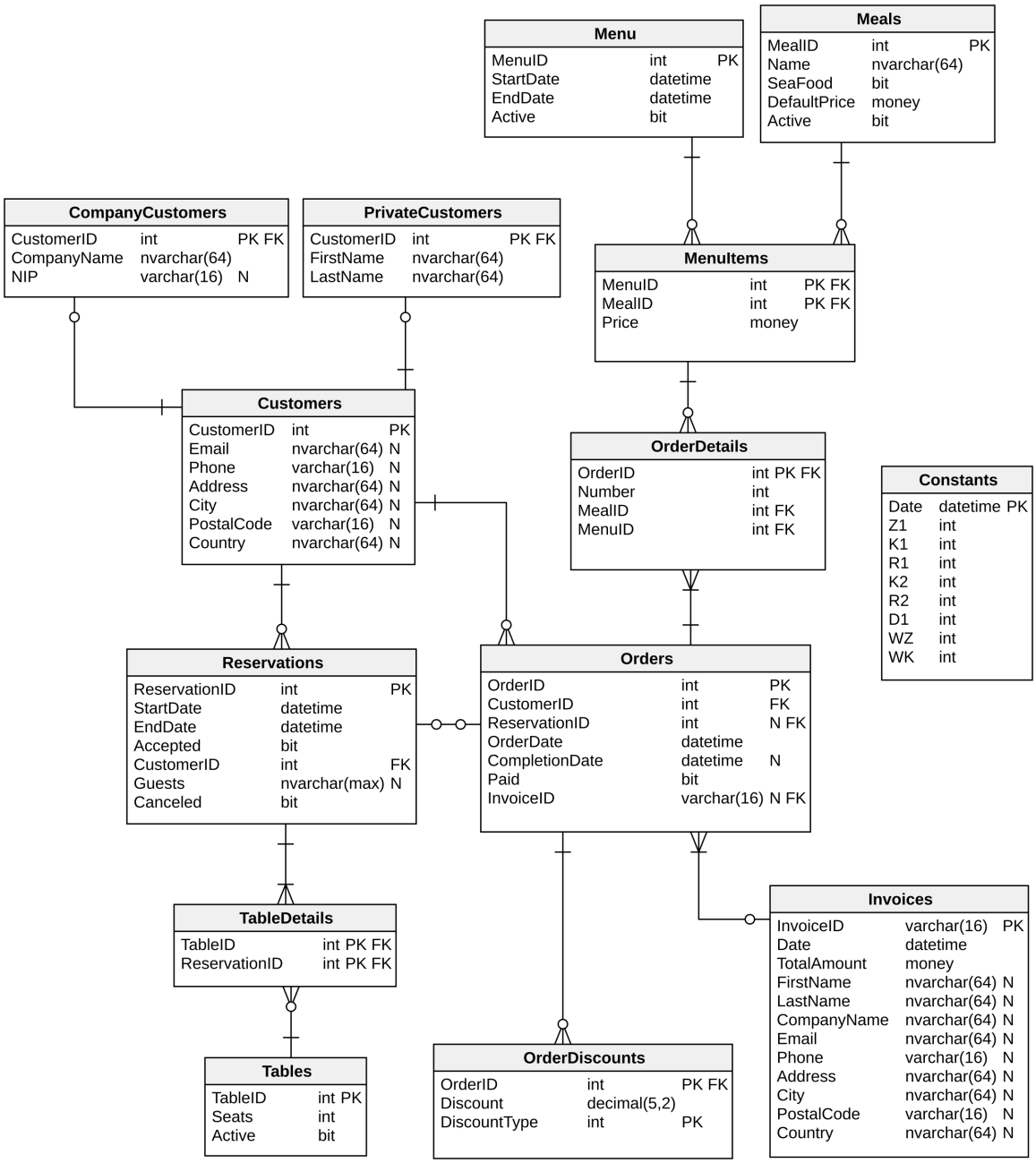
1.3 Funkcje dla kierownictwa

- Generowanie raportów (miesięcznych i tygodniowych) dotyczących rezerwacji, rabatów, menu, statystyk zamówień (kwoty, terminy czy zamówienie zostało złożone przez klienta indywidualnego, czy przez firmę)
- Modyfikacje menu — system sprawdza, czy menu jest zgodne z zasadami
- Możliwość zmiany parametrów rabatów
- Podgląd zamówień z owocami morza

1.4 Zasady przyznawania rabatów

- Zniżka typu pierwszego — po realizacji Z1 zamówień, każde za co najmniej kwotę K1 klient dostaje stałą zniżkę R1% na wszystkie zamówienia.
- Zniżka typu drugiego — po realizacji zamówień za łączną kwotę K2, przez następne D1 dni każde zamówienie otrzymuje zniżkę R2%. Jeśli ktoś spełni ponownie warunek, to zniżki nie połączą się, tylko okres zostanie przedłużony.

2 Schemat bazy danych



3 Tabele

3.1 Firmy

Przechowuje informacje o firmach: numer firmy, nazwa firmy, (opcjonalny) NIP.

```
CREATE TABLE CompanyCustomers (
    CustomerID int  NOT NULL,
    CompanyName nvarchar(64)  NOT NULL,
    NIP varchar(16)  NULL,
    CONSTRAINT CompanyCustomers_pk PRIMARY KEY (CustomerID)
);
```

3.2 Constants

Zawiera informacje o wartościach stałych potrzebnych do wyznaczenia rabatów w danym okresie:
Z1 - minimalna liczba zamówień dla rabatu 1,
K1 - minimalna wydana kwota dla rabatu 1,
R1 - procent zniżki na wszystkie zamówienia po udzieleniu rabatu 1,
K2 - minimalna wydana kwota dla rabatu 2,
R2 - procent zniżki na zamówienie po udzieleniu rabatu 2,
D1 - maksymalna liczba dni na wykorzystanie rabatu 2 począwszy od dnia przyznania zniżki,
WZ - minimalna wartość zamówienia w przypadku wypełniania formularza do rezerwacji,
WK - minimalna ilość wykonanych zamówień w przypadku wypełniania formularza do rezerwacji.

```
CREATE TABLE Constants (
    Date datetime  NOT NULL,
    Z1 int  NOT NULL,
    K1 int  NOT NULL,
    R1 int  NOT NULL,
    K2 int  NOT NULL,
    R2 int  NOT NULL,
    D1 int  NOT NULL,
    WZ int  NOT NULL,
    WK int  NOT NULL,
    CONSTRAINT ConstantChecks CHECK (Z1 >= 0 AND  K1 >= 0 AND  R1 >= 0 AND R1 <= 100
    ↪  AND K2 >= 0 AND R2 >= 0 AND R2 <= 100 AND D1 >= 0 AND WZ >= 0 AND WK >= 0 ),
    CONSTRAINT Constants_pk PRIMARY KEY  (Date)
);
```

3.3 Customers

Przechowuje informacje wspólne o klientach indywidualnych i firmach. Informacje adresowe są opcjonalne (w przypadku kiedy są potrzebne, można je uzupełnić później).

```
CREATE TABLE Customers (
    CustomerID int  NOT NULL IDENTITY(1, 1),
    Email nvarchar(64)  NULL,
    Phone varchar(16)  NULL,
    Address nvarchar(64)  NULL,
    City nvarchar(64)  NULL,
    PostalCode varchar(16)  NULL,
    Country nvarchar(64)  NULL,
    CONSTRAINT Customers_pk PRIMARY KEY  (CustomerID)
);
```

3.4 Invoices

Zawiera informacje o fakturach: numer faktury, data wystawienia faktury, łączna kwota oraz dane klienta.

```
CREATE TABLE Invoices (
    InvoiceID varchar(16)  NOT NULL,
    CustomerID int NOT NULL,
    Date datetime  NOT NULL,
    TotalAmount money  NOT NULL,
    FirstName nvarchar(64)  NULL,
    LastName nvarchar(64)  NULL,
    CompanyName nvarchar(64)  NULL,
    Email nvarchar(64)  NULL,
    Phone varchar(16)  NULL,
    Address nvarchar(64)  NULL,
    City nvarchar(64)  NULL,
    PostalCode varchar(16)  NULL,
    Country nvarchar(64)  NULL,
    CONSTRAINT PositiveTotalAmount CHECK (TotalAmount > 0),
    CONSTRAINT Invoices_pk PRIMARY KEY  (InvoiceID)
);

ALTER TABLE Invoices ADD CONSTRAINT Invoices_Customers
    FOREIGN KEY (CustomerID)
    REFERENCES Customers (CustomerID);
```

3.5 Meals

Lista dań możliwych do użycia podczas tworzenia menu. Zawiera informację o domyślnej cenie oraz oznaczenie dań z owocami morza.

```
CREATE TABLE Meals (
    MealID int  NOT NULL IDENTITY(1, 1),
    Name nvarchar(64)  NOT NULL,
    SeaFood bit  NOT NULL,
    DefaultPrice money  NOT NULL,
    Active bit  NOT NULL,
    CONSTRAINT PositiveDefaultPrice CHECK (DefaultPrice > 0),
    CONSTRAINT Meals_pk PRIMARY KEY  (MealID)
);
```

3.6 Menu

Przechowuje informacje o menu dostępnych w różnych okresach.

```
CREATE TABLE Menu (
    MenuID int  NOT NULL IDENTITY(1, 1),
    StartDate datetime  NOT NULL,
    EndDate datetime  NOT NULL,
    Active bit  NOT NULL,
    CONSTRAINT MenuStartBeforeEnd CHECK (StartDate < EndDate),
    CONSTRAINT Menu_pk PRIMARY KEY  (MenuID)
);
```

3.7 MenuItems

Zawiera wszystkie posiłki dostępne w co najmniej jednym z menu wraz z ich cenami.

```
CREATE TABLE MenuItems (
    MenuID int  NOT NULL,
    MealID int  NOT NULL,
    Price money  NOT NULL,
    CONSTRAINT PositivePrice CHECK (Price > 0),
    CONSTRAINT MenuItems_pk PRIMARY KEY  (MenuID,MealID)
);

ALTER TABLE MenuItems ADD CONSTRAINT MenuItems_Meals
    FOREIGN KEY (MealID)
    REFERENCES Meals (MealID);

ALTER TABLE MenuItems ADD CONSTRAINT Menu_MenuItems
    FOREIGN KEY (MenuID)
    REFERENCES Menu (MenuID);
```

3.8 OrderDetails

Zawiera wszystkie pozycje ze wszystkich złożonych zamówień. Każda pozycja jest przypisana do dokładnie jednego zamówienia i może obejmować kilka sztuk tego samego produktu.

```
CREATE TABLE OrderDetails (
    OrderID int  NOT NULL,
    Quantity int  NOT NULL,
    MealID int  NOT NULL,
    MenuID int  NOT NULL,
    CONSTRAINT PositiveQuantity CHECK (Quantity > 0),
    CONSTRAINT OrderDetails_pk PRIMARY KEY  (OrderID, MealID)
);

ALTER TABLE OrderDetails ADD CONSTRAINT MenuItems_OrderDetails
    FOREIGN KEY (MenuID,MealID)
    REFERENCES MenuItems (MenuID,MealID);

ALTER TABLE OrderDetails ADD CONSTRAINT Orders_OrderDetails
    FOREIGN KEY (OrderID)
    REFERENCES Orders (OrderID);
```

3.9 OrderDiscounts

Zawiera listę udzielonych rabatów. Każdy rabat jest przypisany do dokładnie jednego zamówienia.

```
CREATE TABLE OrderDiscounts (
    OrderID int  NOT NULL,
    Discount decimal(5,2)  NOT NULL,
    DiscountType int  NOT NULL CHECK (DiscountType IN (1, 2)),
    CONSTRAINT DiscountRange CHECK (Discount >= 0 AND Discount <= 1),
    CONSTRAINT OrderDiscounts_pk PRIMARY KEY  (OrderID,DiscountType)
);

ALTER TABLE OrderDiscounts ADD CONSTRAINT OrdersDiscounts_Orders
    FOREIGN KEY (OrderID)
    REFERENCES Orders (OrderID);
```

3.10 Orders

Lista złożonych zamówień wraz z informacją o ich statusie.

```
CREATE TABLE Orders (
    OrderID int  NOT NULL IDENTITY(1, 1),
    CustomerID int  NOT NULL,
    ReservationID int  NULL,
    OrderDate datetime  NOT NULL,
    CompletionDate datetime  NULL,
    Paid bit  NOT NULL,
    InvoiceID varchar(16)  NULL,
    CONSTRAINT OrderedBeforeCompleted CHECK (CompletionDate >= OrderDate),
    CONSTRAINT Orders_pk PRIMARY KEY  (OrderID)
);

ALTER TABLE Orders ADD CONSTRAINT Order_Reservations
    FOREIGN KEY (ReservationID)
    REFERENCES Reservations (ReservationID);

ALTER TABLE Orders ADD CONSTRAINT Orders_Customers
    FOREIGN KEY (CustomerID)
    REFERENCES Customers (CustomerID);
```

3.11 PrivateCustomers

Przechowuje informacje o klientach indywidualnych: imię i nazwisko

```
CREATE TABLE PrivateCustomers (
    CustomerID int  NOT NULL,
    FirstName nvarchar(64)  NOT NULL,
    LastName nvarchar(64)  NOT NULL,
    CONSTRAINT PrivateCustomers_pk PRIMARY KEY  (CustomerID)
);
```

3.12 Reservations

Przechowuje listę rezerwacji stolików.

```
CREATE TABLE Reservations (
    ReservationID int  NOT NULL IDENTITY(1, 1),
    StartDate datetime  NOT NULL,
    EndDate datetime  NOT NULL,
    Accepted bit  NOT NULL,
    CustomerID int  NOT NULL,
    Guests nvarchar(max)  NULL,
    Canceled bit  NOT NULL,
    CONSTRAINT ReservationStartBeforeEnd CHECK (StartDate < EndDate),
    CONSTRAINT Reservations_pk PRIMARY KEY  (ReservationID)
);

ALTER TABLE Reservations ADD CONSTRAINT Reservations_Customers
    FOREIGN KEY (CustomerID)
    REFERENCES Customers (CustomerID);
```

3.13 TableDetails

Zawiera szczegóły rezerwacji poszczególnych stolików (przypisanie stolika do rezerwacji)

```
CREATE TABLE TableDetails (
    TableID int  NOT NULL,
    ReservationID int  NOT NULL,
    CONSTRAINT TableDetails_pk PRIMARY KEY  (TableID,ReservationID)
);

ALTER TABLE TableDetails ADD CONSTRAINT Reservations_TableDetails
    FOREIGN KEY (ReservationID)
    REFERENCES Reservations (ReservationID);

ALTER TABLE TableDetails ADD CONSTRAINT TableDetails_Tables
    FOREIGN KEY (TableID)
    REFERENCES Tables (TableID);
```

3.14 Tables

Lista stolików dostępnych w restauracji.

```
CREATE TABLE Tables (
    TableID int  NOT NULL IDENTITY(1, 1),
    Seats int  NOT NULL,
    Active bit  NOT NULL,
    CONSTRAINT PositiveSeats CHECK (Seats > 0),
    CONSTRAINT Tables_pk PRIMARY KEY  (TableID)
);

ALTER TABLE CompanyCustomers ADD CONSTRAINT Customers_CompanyCustomers
    FOREIGN KEY (CustomerID)
    REFERENCES Customers (CustomerID);

ALTER TABLE PrivateCustomers ADD CONSTRAINT Customers_PrivateCustomers
    FOREIGN KEY (CustomerID)
    REFERENCES Customers (CustomerID);

ALTER TABLE Orders ADD CONSTRAINT Invoices_Orders
    FOREIGN KEY (InvoiceID)
    REFERENCES Invoices (InvoiceID);
```

4 Procedury

4.1 AddCompanyCustomer(...)

Dodaje firmę jako klienta.

```
CREATE OR ALTER PROCEDURE AddCompanyCustomer(
    @Email nvarchar(64),
    @Phone nvarchar(16),
    @Address nvarchar(64),
    @City nvarchar(64),
    @PostalCode varchar(16),
    @Country nvarchar(64),
    @CompanyName nvarchar(64),
```

```
    @NIP varchar(16)
)
AS BEGIN
    BEGIN TRANSACTION;
        INSERT INTO Customers (Email, Phone, Address, City, PostalCode, Country)
        VALUES (@Email, @Phone, @Address, @City, @PostalCode, @Country)
        INSERT INTO CompanyCustomers (CustomerID, CompanyName, NIP)
        VALUES (@@IDENTITY, @CompanyName, @NIP)
    COMMIT;
END
GO
```

4.2 AddPrivateCustomer(...)

Dodaje osobę prywatną jako klienta.

```
CREATE OR ALTER PROCEDURE AddPrivateCustomer(
    @Email nvarchar(64),
    @Phone nvarchar(16),
    @Address nvarchar(64),
    @City nvarchar(64),
    @PostalCode varchar(16),
    @Country nvarchar(64),
    @FirstName nvarchar(64),
    @LastName nvarchar(64)
)
AS BEGIN
    BEGIN TRANSACTION;
        INSERT INTO Customers (Email, Phone, Address, City, PostalCode, Country)
        VALUES (@Email, @Phone, @Address, @City, @PostalCode, @Country)
        INSERT INTO PrivateCustomers (CustomerID, FirstName, LastName)
        VALUES (@@IDENTITY, @FirstName, @LastName)
    COMMIT;
END
GO
```

4.3 AddTable(Seats)

Dodaje nowy stolik

```
CREATE OR ALTER PROCEDURE AddTable (@Seats int)
AS BEGIN
    INSERT INTO Tables(Seats, Active)
    VALUES (@Seats, 1)
END
GO
```

4.4 DisableTable(TableID)

Oznacza stolik jako nieaktywny

```
CREATE OR ALTER PROCEDURE DisableTable (@TableID int)
AS BEGIN

    IF(SELECT Active FROM Tables WHERE TableID = @TableID) = 0
    BEGIN
```

```
        ;THROW 52000, 'Table already inactive', 1
        RETURN
    END

    UPDATE Tables SET Active = 0
    WHERE TableID = @TableID
END
GO
```

4.5 EnableTable(TableID)

Oznacza stolik jako aktywny

```
CREATE OR ALTER PROCEDURE EnableTable (@TableID int)
AS BEGIN

    IF(SELECT Active FROM Tables WHERE TableID = @TableID) = 1
    BEGIN
        ;THROW 52000, 'Table already active', 1
        RETURN
    END

    UPDATE Tables SET Active = 1
    WHERE TableID = @TableID
END
GO
```

4.6 NewMenuInProgress(StartDate, EndData, MenuID OUTPUT)

Tworzy nowe nieaktywne menu.

```
CREATE OR ALTER PROCEDURE NewMenuInProgress(@StartDate datetime, @EndDate datetime,
↪  @MenuID int OUTPUT)
AS BEGIN
    INSERT INTO Menu(StartDate, EndDate, Active)
    VALUES(@StartDate, @EndDate, 0)

    SET @MenuID = @@IDENTITY
END
GO
```

4.7 ChangeMenuDates(MenuID, StartDate, EndDate)

Zmienia daty niaktywnego menu.

```
CREATE OR ALTER PROCEDURE ChangeMenuDates(@MenuID int, @StartDate datetime = NULL,
↪  @EndDate datetime = NULL)
AS BEGIN
    IF (SELECT Active FROM Menu WHERE MenuID = @MenuID) = 1
    BEGIN
        ;THROW 52000, 'Menu is active', 1
        RETURN
    END

    DECLARE @PrevStartDate datetime
    DECLARE @PrevEndDate datetime
```

```
11
12      SELECT @PrevStartDate = StartDate, @PrevEndDate = EndDate
13      FROM Menu WHERE MenuID = @MenuID
14
15      UPDATE Menu
16      SET StartDate = ISNULL(@StartDate, @PrevStartDate),
17          EndDate = ISNULL(@EndDate, @PrevEndDate)
18      WHERE MenuID = @MenuID
19  END
20  GO
```

4.8 SetMenuItem(MenuID, MealID, Price)

Dodaje posiłek do nieaktywnego menu.

```
1   CREATE OR ALTER PROCEDURE SetMenuItem(@MenuID int, @MealID int, @Price money = NULL)
2   AS BEGIN
3       IF (SELECT Active FROM Menu WHERE MenuID = @MenuID) = 1
4       BEGIN
5           ;THROW 52000, 'Menu is active', 1
6           RETURN
7       END
8
9       IF (SELECT Active FROM Meals WHERE MealID = @MealID) = 0
10      BEGIN
11          ;THROW 52000, 'Meal is not active', 1
12          RETURN
13      END
14
15      DECLARE @DefaultPrice money = (SELECT DefaultPrice FROM Meals WHERE MealID =
        ↪  @MealID)
16
17      INSERT INTO MenuItems(MenuID, MealID, Price)
18      VALUES (@MenuID, @MealID, ISNULL(@Price, @DefaultPrice))
19  END
20  GO
```

4.9 RemoveMenuItem(MenuID, MealID)

Usuwa posiłek z nieaktynego menu.

```
1   CREATE OR ALTER PROCEDURE RemoveMenuItem(@MenuID int, @MealID int)
2   AS BEGIN
3       IF (SELECT Active FROM Menu WHERE MenuID = @MenuID) = 1
4       BEGIN
5           ;THROW 52000, 'Menu is active', 1
6           RETURN
7       END
8
9       DELETE MenuItems
10      WHERE MenuID = @MenuID AND MealID = @MealID
11  END
12  GO
```

4.10 ActivateMenu(MenuID)

Próbuje aktywować menu biorąc pod uwagę niepowtarzanie się posiłków i nienachodzenie dat.

```sql
CREATE OR ALTER PROCEDURE ActivateMenu(@MenuID int)
AS BEGIN
    -- Check if not active
    IF (SELECT Active FROM Menu WHERE MenuID = @MenuID) = 1
    BEGIN
        ;THROW 52000, 'Menu is active', 1
        RETURN
    END

    -- Check if dates do not overlap
    DECLARE @StartDate datetime = (SELECT StartDate FROM Menu WHERE MenuID = @MenuID)
    DECLARE @LastMenuDate datetime = (SELECT MAX(EndDate) FROM Menu WHERE Active = 1)

    if DATEDIFF(day, @LastMenuDate, @StartDate) <= 0
    BEGIN
        ;THROW 52000, 'Overlapping dates', 1
        RETURN
    END

    -- Check if the menu items are legal
    DECLARE @Count int
    DECLARE @NotChangedCount int

    SELECT @Count = Count(MealID)
    FROM Menu
    JOIN MenuItems ON MenuItems.MenuID = Menu.MenuID
    WHERE Menu.MenuID = @MenuID

    SELECT @NotChangedCount = Count(MealID)
    FROM Menu
    JOIN MenuItems ON MenuItems.MenuID = Menu.MenuID
    WHERE Menu.MenuID = @MenuID AND MenuItems.MealID IN (
        SELECT MI2.MealID
        FROM MenuItems AS MI2
        JOIN Menu AS M2 ON M2.MenuID = MI2.MenuID
        WHERE M2.Active = 1 AND DATEDIFF(day, M2.EndDate, Menu.StartDate) < 14
    )

    IF (@NotChangedCount * 2) > @Count
    BEGIN
        ;THROW 52000, 'Menu is not legal', 1
        RETURN
    END

    -- Everything is correct
    UPDATE Menu SET Active = 1
    WHERE MenuID = @MenuID
END
GO
```

4.11 CreateOrderInvoice(OrderID)

Generuje fakturę w tabeli Invoices dla danego zamówienia.

```sql
CREATE OR ALTER PROCEDURE CreateOrderInvoice(@OrderID int)
AS BEGIN
    IF (SELECT InvoiceID FROM Orders WHERE OrderID = @OrderID) IS NOT NULL
```

```
    BEGIN
        ;THROW 5200, 'Order already has an invoice', 1
        RETURN
    END
    IF (SELECT Paid FROM Orders WHERE OrderID = @OrderID) = 0
    BEGIN
        ;THROW 5200, 'Order has not been paid yet', 1
        RETURN
    END

    BEGIN TRANSACTION;
        INSERT INTO Invoices(
            Date, CustomerID, TotalAmount, FirstName, LastName, CompanyName, Email,
            ↪ Phone, Address, City, PostalCode, Country
        )
        SELECT GETDATE(), Customers.CustomerID, dbo.TotalOrderAmount(@OrderID),
        ↪ FirstName, LastName, CompanyName,
                Email, Phone, Address, City, PostalCode, Country
        FROM Orders
            JOIN Customers ON Customers.CustomerID = Orders.OrderID
            LEFT JOIN CompanyCustomers ON CompanyCustomers.CustomerID =
            ↪ Customers.CustomerID
            LEFT JOIN PrivateCustomers ON PrivateCustomers.CustomerID =
            ↪ Customers.CustomerID
        WHERE Orders.OrderID = @OrderID;

        UPDATE Orders SET InvoiceID = @@IDENTITY
        WHERE OrderID = @OrderID
    COMMIT;
END
GO
```

4.12 CreateMonthlyInvoice(CustomerID, Month, Year)

Generuje fakturę dla danego klienta, dla danego miesiąca.

```
CREATE OR ALTER PROCEDURE CreateMonthlyInvoice(@CustomerID Int, @Month int, @Year int)
AS BEGIN
    IF DATEFROMPARTS(@Year, @Month, DAY(EOMONTH(DATEFROMPARTS(@Year, @Month, 1)))) >=
    ↪ GETDATE()
    BEGIN
        ;THROW 5200, 'The month hasnt passed yet', 1
        RETURN
    END

    BEGIN TRANSACTION;
        INSERT INTO Invoices(
            Date, CustomerID, TotalAmount, FirstName, LastName, CompanyName, Email,
            ↪ Phone, Address, City, PostalCode, Country
        )
        SELECT GETDATE(), Customers.CustomerID
        ↪ SUM(dbo.TotalOrderAmount(Orders.OrderID)), MAX(FirstName), MAX(LastName),
            MAX(CompanyName), MAX(Email), MAX(Phone), MAX(Address), MAX(City),
            ↪ MAX(PostalCode), MAX(Country)
        FROM Customers
            LEFT JOIN Orders ON Orders.CustomerID = Customers.CustomerID AND
            ↪ Orders.InvoiceID IS NULL
```

```
                                    AND MONTH(Orders.CompletionDate) = @Month AND
                                    ↪  YEAR(Orders.CompletionDate) = @Year
            LEFT JOIN CompanyCustomers ON CompanyCustomers.CustomerID =
            ↪  Customers.CustomerID
            LEFT JOIN PrivateCustomers ON PrivateCustomers.CustomerID =
            ↪  Customers.CustomerID
        WHERE Customers.CustomerID = @CustomerID
        GROUP BY Customers.CustomerID

        UPDATE Orders SET InvoiceID = @@IDENTITY
        WHERE Orders.CustomerID = @CustomerID AND Orders.InvoiceID IS NULL
                AND MONTH(Orders.CompletionDate) = @Month AND
                ↪  YEAR(Orders.CompletionDate) = @Year
    COMMIT;
END
GO
```

4.13 UpdateConstants(...)

Aktualizuje podane stałe (nie zmieniając pozostałych).

```
CREATE OR ALTER PROCEDURE UpdateConstants(
    @Z1 INT = NULL,
    @K1 INT = NULL,
    @R1 INT = NULL,
    @K2 INT = NULL,
    @R2 INT = NULL,
    @D1 INT = NULL,
    @WZ INT = NULL,
    @WK INT = NULL
) AS BEGIN
    DECLARE @PREV_Z1 INT
    DECLARE @PREV_K1 INT
    DECLARE @PREV_R1 INT
    DECLARE @PREV_K2 INT
    DECLARE @PREV_R2 INT
    DECLARE @PREV_D1 INT
    DECLARE @PREV_WZ INT
    DECLARE @PREV_WK INT

    SELECT
        @PREV_Z1 = Z1,
        @PREV_K1 = K1,
        @PREV_R1 = R1,
        @PREV_K2 = K2,
        @PREV_R2 = R2,
        @PREV_D1 = D1,
        @PREV_WZ = WZ,
        @PREV_WK = WK
    FROM Constants

    INSERT INTO Constants(Date, Z1, K1, R1, K2, R2, D1, WZ, WK)
    VALUES (
        GETDATE(),
        ISNULL(@Z1, @PREV_Z1),
        ISNULL(@K1, @PREV_K1),
        ISNULL(@R1, @PREV_R1),
```

```
        ISNULL(@K2, @PREV_K2),
        ISNULL(@R2, @PREV_R2),
        ISNULL(@D1, @PREV_D1),
        ISNULL(@WZ, @PREV_WZ),
        ISNULL(@WK, @PREV_WK)
    )
END
GO
```

4.14 AddReservation(StartDate, EndDate, CustomerID, Guests)

Dodaje nową rezerwację

```
CREATE OR ALTER PROCEDURE AddReservation (@StartDate datetime, @EndDate datetime,
↪  @CustomerID int, @Guests nvarchar(max), @Tables ReservationTablesList READONLY)
AS BEGIN

    -- IF (AreTablesAvailable (@StartDate, @EndDate, @Tables))

    INSERT INTO Reservations(StartDate, EndDate, Accepted, CustomerID, Guests,
    ↪  Canceled)
    VALUES (@StartDate, @EndDate, 0, @CustomerID, @Guests, 0)

    DECLARE @ReservationID int = @@IDENTITY

    INSERT INTO TableDetails(TableID, ReservationID)
    SELECT @TableID, @ReservationID FROM @Tables
END
GO
```

4.15 AcceptReservation(ReservationID)

Akceptuje wybraną rezerwację

```
CREATE OR ALTER PROCEDURE AcceptReservation (@ReservationID int)
AS BEGIN
    IF(SELECT Accepted FROM Reservations WHERE ReservationID = @ReservationID) = 1
    BEGIN
        ;THROW 52000, 'Reservation already accepted', 1
        RETURN
    END

    IF(SELECT Canceled FROM Reservations WHERE ReservationID = @ReservationID) = 1
    BEGIN
        ;THROW 52000, 'Reservation cancelled before acceptation', 1
        RETURN
    END

    UPDATE Reservations SET Accepted = 1
    WHERE ReservationID = @ReservationID
END
GO
```

4.16 CancelReservation(ReservationID)

Anuluje wybraną rezerwację

```sql
CREATE OR ALTER PROCEDURE CancelReservation (@ReservationID int)
AS BEGIN

    IF(SELECT Canceled FROM Reservations WHERE ReservationID = @ReservationID) = 1
    BEGIN
        ;THROW 52000, 'Reservation already cancelled', 1
        RETURN
    END

    UPDATE Reservations SET Canceled = 1
    WHERE ReservationID = @ReservationID

END
GO
```

4.17 CreateOrder(CustomerID, CompletionDate, OrderedItems)

Tworzy nowe zamówienie w systemie. Zamówienie jest przypisane do konkretnego klienta i ma datę odbioru.

```sql
CREATE OR ALTER PROCEDURE CreateOrder(@CustomerID int, @CompletionDate datetime,
↪  @OrderedItems OrderedItemsListT READONLY)
AS BEGIN

    SET XACT_ABORT ON

    -- check if the customer exists
    IF NOT EXISTS (SELECT * FROM Customers WHERE CustomerID = @CustomerID)
    BEGIN
        ;THROW 52000, 'The customer does not exist', 1
        RETURN
    END

    DECLARE @MenuID int = dbo.GetMenuForDay(@CompletionDate)
    IF @MenuID IS NULL
    BEGIN
        ;THROW 52000, 'The menu does not exist', 1
        RETURN
    END

    -- check if all items belong to the proper menu
    IF (SELECT count(1) FROM @OrderedItems) != (SELECT count(1) FROM MenuItems WHERE
    ↪  MenuID = @MenuID AND MealID IN (SELECT MealID FROM @OrderedItems))
    BEGIN
        ;THROW 52000, 'The ordered items list is incorrect', 1
        RETURN
    END

    BEGIN TRANSACTION
    -- BEGIN TRY

        INSERT INTO Orders(CustomerID, OrderDate, Paid)
        VALUES (@CustomerID, GETDATE(), 0)

        DECLARE @OrderID int = @@IDENTITY

        INSERT INTO OrderDetails(OrderID, Quantity, MealID, MenuID)
```

```
        SELECT @OrderID, Quantity, MealID, @MenuID FROM @OrderedItems

    COMMIT TRANSACTION
    -- END TRY
    -- BEGIN CATCH
    --     ROLLBACK TRANSACTION
    -- END CATCH

END
GO
```

4.18 PayForOrder(OrderID)

Zapisuje informację, że klient zapłacił za dane zamówienie.

```
CREATE OR ALTER PROCEDURE PayForOrder (@OrderID int)
AS BEGIN

    -- check if the order exists
    IF NOT EXISTS (SELECT * FROM Orders WHERE OrderID = @OrderID)
    BEGIN
        ;THROW 52000, 'The order does not exist', 1
        RETURN
    END

    UPDATE Orders
    SET Paid = 1
    WHERE OrderID = @OrderID

END
GO
```

4.19 CompleteOrder(OrderID)

Zapisuje informację, że zamówienie zostało wydane klientowi.

```
CREATE OR ALTER PROCEDURE PayForOrder (@OrderID int)
AS BEGIN

    -- check if the order exists
    IF NOT EXISTS (SELECT * FROM Orders WHERE OrderID = @OrderID)
    BEGIN
        ;THROW 52000, 'The order does not exist', 1
        RETURN
    END

    UPDATE Orders
    SET CompletionDate = GETDATE()
    WHERE OrderID = @OrderID

END
GO
```

5 Widoki

5.1 MenusInProgress

Pokazuje nieaktywne menu.

```
CREATE OR ALTER VIEW MenusInProgress AS
SELECT MenuID FROM Menu WHERE Active = 0
GO
```

5.2 CurrentOrders

Pokazuje zamówienia w trakcie realizacji.

```
CREATE OR ALTER VIEW CurrentOrders
AS SELECT OrderID, CustomerID, ReservationID, Paid, InvoiceID FROM Orders
WHERE OrderDate <= GETDATE() AND GETDATE() < CompletionDate
GO
```

5.3 OrderHist

Pokazuje historię zamówień.

```
CREATE OR ALTER VIEW OrderHist
AS SELECT OrderID, CustomerID, ReservationID, Paid, InvoiceID FROM Orders WHERE
↪  CompletionDate <= GETDATE()
GO
```

5.4 ReservationsToAccept

Pokazuje rezerwacje, które nie zostały zaakceptowane.

```
CREATE OR ALTER VIEW ReservationsToAccept
AS SELECT ReservationID, CustomerID, Guests, Canceled FROM Reservations WHERE Accepted
↪  = 0
GO
```

5.5 SeafoodOrders

Pokazuje zamówienia, które zawierają dania z owocami morza.

```
CREATE OR ALTER VIEW SeafoodOrders
AS SELECT O.OrderID, M.MealID, MI.MenuID, OD.Quantity, O.CustomerID, O.ReservationID,
↪  O.OrderDate, O.CompletionDate, O.Paid, O.InvoiceID
    FROM Orders O
    INNER JOIN OrderDetails OD ON O.OrderID = OD.OrderID
    INNER JOIN MenuItems MI ON OD.MenuID = MI.MenuID AND OD.MealID = MI.MealID
    INNER JOIN Meals M ON M.MealID = MI.MealID
WHERE SeaFood = 1
GO
```

5.6 CurrentConstants

Zwraca aktualne wartości stałych w systemie

```
CREATE OR ALTER VIEW CurrentConstants
AS SELECT TOP 1 c.Z1, c.K1, c.R1, c.K2, c.R2, c.D1, c.WZ, c.WK
    FROM Constants c
    ORDER BY c.[Date] DESC
GO
```

5.7 CurrentMenu

Zwraca aktualne menu dla zamówień na ten sam dzień

```
CREATE OR ALTER VIEW CurrentMenu
AS SELECT MI.MenuID, MI.MealID, MI.Price
FROM MenuItems MI INNER JOIN Menu M ON M.MenuID = MI.MenuID
WHERE StartDate <= GETDATE() AND GETDATE() <= EndDate
GO
```

5.8 TablesAvailableNow

Zwraca aktualnie dostępne stoliki

```
CREATE OR ALTER VIEW TablesAvailableNow
AS SELECT T.TableID, Seats
FROM Tables T INNER JOIN TableDetails TD ON T.TableID = TD.TableID
INNER JOIN Reservations R ON TD.ReservationID = R.ReservationID
WHERE Active = 1 AND
(T.TableID NOT IN (SELECT TableID FROM TableDetails)
OR T.TableID NOT IN (SELECT TableID FROM TableDetails TD1
    INNER JOIN Reservations R1 ON TD1.ReservationID = R1.ReservationID
    WHERE R1.StartDate <= GETDATE() AND GETDATE() <= EndDate AND R1.Canceled = 0))
GO
```

6 Funkcje

6.1 MealsStatistics(Monthly, Date)

Raport dotyczący posiłków, pokazujący ile razy został zamówiony i ile na niego wydano. Jeśli Monthly jest ustawione na 1, raport jest miesięczny, a w przeciwnym wypadku jest tygodniowy. Raport dotyczący posiłków, pokazujący ile razy został zamówiony i ile na niego wydano. Jeśli Monthly jest ustawione na 1, raport jest miesięczny, a w przeciwnym wypadku jest tygodniowy.

```
CREATE OR ALTER FUNCTION MealsStatistics(
    @Monthly bit,
    @Date datetime
)RETURNS @Statistics TABLE ([Name] nvarchar(64), Quantity int, [TotalAmount] money)
BEGIN
    DECLARE @EndDate datetime = CASE @Monthly
        WHEN 0 THEN DATEADD(week, -1, @Date)
        ELSE DATEADD(month, -1, @Date)
    END

    INSERT @Statistics
```

```
        SELECT [Name], ISNULL(Sum(OD.Quantity), 0), ISNULL(Sum(OD.Quantity *
        ↪  MI.Price), 0)
        FROM Meals
        LEFT JOIN OrderDetails OD ON OD.MealID = Meals.MealID
        LEFT JOIN MenuItems MI ON MI.MealID = OD.MealID AND MI.MenuID = OD.MenuID
        LEFT JOIN Orders ON Orders.OrderID = OD.OrderID
        WHERE Orders.CompletionDate IS NULL OR Orders.CompletionDate BETWEEN @Date AND
        ↪  @EndDate
        GROUP BY Meals.MealID, Meals.Name

    RETURN
END
GO

CREATE OR ALTER FUNCTION MealsStatistics(
    @Monthly bit,
    @Date datetime
)RETURNS @Statistics TABLE ([Name] nvarchar(64), [Number] int, [TotalAmount] money)
BEGIN
    DECLARE @EndDate datetime = CASE @Monthly
        WHEN 0 THEN DATEADD(week, -1, @Date)
        ELSE DATEADD(month, -1, @Date)
    END

    INSERT @Statistics
        SELECT [Name], ISNULL(Sum(OD.Number), 0), ISNULL(Sum(OD.Number * MI.Price), 0)
        FROM Meals
        LEFT JOIN OrderDetails OD ON OD.MealID = Meals.MealID
        LEFT JOIN MenuItems MI ON MI.MealID = OD.MealID AND MI.MenuID = OD.MenuID
        LEFT JOIN Orders ON Orders.OrderID = OD.OrderID
        WHERE Orders.CompletionDate IS NULL OR Orders.CompletionDate BETWEEN @Date AND
        ↪  @EndDate
        GROUP BY Meals.MealID, Meals.Name

    RETURN
END
GO
```

6.2 CustomerStatistics(CustomerID)

Raport dotyczący danego klienta, wyświetla dla każdego zamówienia końcowa cenę, czas w którym zamówienie spłyneło i datę na które jest to zamówienie.

```
CREATE OR ALTER FUNCTION CustomerStatistics(
    @CustomerID int
)RETURNS @Statistics TABLE(Amount money, OrderDate datetime, CompletionDate datetime)
BEGIN
    INSERT @Statistics
    SELECT dbo.TotalOrderAmount(OrderID), OrderDate, CompletionDate
    FROM Orders
    WHERE CustomerID = @CustomerID
    RETURN
END
GO
```

6.3 OrderStatistics(CustomerID)

Raport dotyczący zamówień, wyświetla dla każdego zamówienia końcowa cenę, czas w którym zamówienie spłynęło i datę na które jest to zamówienie, a także nazwę klienta (imię i nazwisko w przypadku klienta indywidualnego).

```
CREATE OR ALTER FUNCTION OrderStatistics()
RETURNS @Statistics TABLE(Amount money, OrderDate datetime, CompletionDate datetime,
↪  Who nvarchar(64))
BEGIN
    INSERT @Statistics
    SELECT dbo.TotalOrderAmount(OrderID), OrderDate, CompletionDate,
    ↪  ISNULL(CompanyCustomers.CompanyName, PrivateCustomers.FirstName + ' ' +
    ↪  PrivateCustomers.LastName)
    FROM Orders
    LEFT JOIN CompanyCustomers ON CompanyCustomers.CustomerID = Orders.CustomerID
    LEFT JOIN PrivateCustomers ON PrivateCustomers.CustomerID = Orders.CustomerID
    RETURN
END
GO
```

6.4 CustomerStatistics(CustomerID, Monthly, Date)

Raport dotyczący danego klienta, wyświetla dla każdego zamówienia końcowa cenę, czas w którym zamówienie spłynęło i datę na które jest to zamówienie. Jeśli Monthly jest ustawione na 1, raport jest miesięczny, a w przeciwnym wypadku jest tygodniowy.

```
CREATE OR ALTER FUNCTION CustomerStatistics(
    @CustomerID int,
    @Monthly bit,
    @Date datetime
)RETURNS @Statistics TABLE(Amount money, OrderDate datetime, CompletionDate datetime)
BEGIN
    DECLARE @EndDate datetime = CASE @Monthly
        WHEN 0 THEN DATEADD(week, -1, @Date)
        ELSE DATEADD(month, -1, @Date)
    END

    INSERT @Statistics
    SELECT dbo.TotalOrderAmount(OrderID), OrderDate, CompletionDate
    FROM Orders
    WHERE CustomerID = @CustomerID AND CompletionDate BETWEEN @Date AND @EndDate

    RETURN
END
GO
```

6.5 OrderStatistics(Monthly, Date)

Raport dotyczący zamówień, wyświetla dla każdego zamówienia końcowa cenę, czas w którym zamówienie spłynęło i datę na które jest to zamówienie, a także nazwę klienta (imię i nazwisko w przypadku klienta indywidualnego). Jeśli Monthly jest ustawione na 1, raport jest miesięczny, a w przeciwnym wypadku jest tygodniowy.

```
CREATE OR ALTER FUNCTION OrderStatistics(
    @Monthly bit,
    @Date datetime
```

```
)RETURNS @Statistics TABLE(Amount money, OrderDate datetime, CompletionDate datetime,
    Who nvarchar(64))
BEGIN
    DECLARE @EndDate datetime = CASE @Monthly
        WHEN 0 THEN DATEADD(week, -1, @Date)
        ELSE DATEADD(month, -1, @Date)
    END

    INSERT @Statistics
    SELECT dbo.TotalOrderAmount(OrderID), OrderDate, CompletionDate,
    ISNULL(CompanyCustomers.CompanyName, PrivateCustomers.FirstName + ' ' +
    PrivateCustomers.LastName)
    FROM Orders
    LEFT JOIN CompanyCustomers ON CompanyCustomers.CustomerID = Orders.CustomerID
    LEFT JOIN PrivateCustomers ON PrivateCustomers.CustomerID = Orders.CustomerID
    WHERE CompletionDate BETWEEN @Date AND @EndDate

    RETURN
END
GO
```

6.6 TableStatistics(Monthly, Date)

Raport dotyczący stolików, dla każdego pokazuje ilość miejsc, to czy jest aktywny a także ile razy został zarezerowany w danym okresie. Jeśli Monthly jest ustawione na 1, raport jest miesięczny, a w przeciwnym wypadku jest tygodniowy.

```
CREATE OR ALTER FUNCTION TableStatistics (
    @Monthly bit,
    @Date datetime
) RETURNS @Statistics TABLE(TableID int, Seats int, Active bit, TimesUsed int)
BEGIN
    DECLARE @EndDate datetime = CASE @Monthly
        WHEN 0 THEN DATEADD(week, -1, @Date)
        ELSE DATEADD(month, -1, @Date)
    END

    INSERT @Statistics
    SELECT Tables.TableID, Seats, Active, COUNT(Reservations.ReservationID)
    FROM Tables
    LEFT JOIN TableDetails ON Tables.TableID = TableDetails.TableID
    LEFT JOIN Reservations ON TableDetails.ReservationID = Reservations.ReservationID
    WHERE StartDate BETWEEN @Date AND @EndDate
    GROUP BY Tables.TableID, Seats, Active

    RETURN
END
GO
```

6.7 DiscountsStatistics(Monthly, Date)

Raport dotyczący rabatów, zawiera typ rabatu, ilość w procentach, a także ile razy został wykorzystany w danym okresie. Jeśli Monthly jest ustawione na 1, raport jest miesięczny, a w przeciwnym wypadku jest tygodniowy.

```
CREATE OR ALTER FUNCTION DiscountsStatistics(
    @Monthly bit,
```

```
    @Date datetime
) RETURNS @Statistics TABLE(DiscountType int, Amount decimal(5,2), TimesUsed int)
BEGIN
    DECLARE @EndDate datetime = CASE @Monthly
        WHEN 0 THEN DATEADD(week, -1, @Date)
        ELSE DATEADD(month, -1, @Date)
    END

    INSERT @Statistics
    SELECT DiscountType, Discount, Count(*)
    FROM OrderDiscounts
    JOIN Orders ON OrderDiscounts.OrderID = Orders.OrderID
    WHERE Orders.CompletionDate BETWEEN @Date AND @EndDate
    GROUP BY DiscountType, Discount

    RETURN
END
GO
```

6.8 AreTablesAvailable(StartDate, EndDate, Tables)

Sprawdza czy stoliki z listy są dostępne w danym przedziale czasowym

```
CREATE OR ALTER FUNCTION AreTablesAvailable(@StartDate datetime, @EndDate datetime,
↪  @Tables ReservationTablesList READONLY)
RETURNS BIT
BEGIN
    IF ( @Tables.TableID IN
    (SELECT TableID FROM TableDetails TD
    INNER JOIN Reservations R ON TD.ReservationID = R.ReservationID
    WHERE TableID = @Tables.TableID
    AND ((NOT EndDate <= @StartDate OR StartDate >= @EndDate) AND Canceled = 0)))
    BEGIN
        ;THROW 52000, 'Not all selected tables will be available', 1
        RETURN 0
    END
    RETURN 1
END
GO
```

6.9 TotalOrderAmount(OrderID)

Zwraca całkowitą cenę zamówienia biorąc pod uwagę rabaty.

```
CREATE OR ALTER FUNCTION TotalOrderAmount(@OrderID int) RETURNS money
BEGIN
    RETURN (
        SELECT SUM(OD.Number * MI.Price) * (1-SUM(Discounts.Discount)) FROM Orders
        JOIN OrderDetails AS OD ON OD.OrderID = Orders.OrderID
        JOIN MenuItems AS MI ON MI.MenuID = OD.MenuID AND MI.MealID = OD.MealID
        JOIN OrderDiscounts AS Discounts ON Discounts.OrderID = Orders.OrderID
        WHERE Orders.OrderID = @OrderID
        GROUP BY Orders.OrderID
    )
END
GO
```

6.10 IsDiscountType1(CustomerID)

Sprawdza czy klientowi przysługuje w tej chwili rabat typu pierwszego (co najmniej Z1 zamówień za kwotę przynajmniej K1)

```
CREATE OR ALTER FUNCTION IsDiscountType1(@CustomerID int) Returns bit
BEGIN
    DECLARE @MinOrdersNumber int = (SELECT Z1 FROM CurrentConstants)
    DECLARE @MinSingleOrderAmount int  = (SELECT K1 FROM CurrentConstants)

    DECLARE @BigOrdersNumber money = (
        SELECT COUNT(1)
        FROM Orders o
        WHERE o.CustomerID = @CustomerID AND dbo.TotalOrderAmount(o.OrderID) >
        ↪  @MinSingleOrderAmount
    )

    RETURN CASE WHEN (@BigOrdersNumber >= @MinOrdersNumber) THEN 1 ELSE 0 END
END
GO
```

6.11 IsDiscountType2(CustomerID)

Sprawdza czy klientowi przysługuje w tej chwili rabat typu drugiego (zamówienia za co najmniej K2 w ciągu ostatich D1 dni)

```
CREATE OR ALTER FUNCTION IsDiscountType2(@CustomerID int) RETURNS bit
BEGIN
    DECLARE @MinTotalAmount int = (SELECT K2 FROM CurrentConstants)
    DECLARE @LastDays int  = (SELECT D1 FROM CurrentConstants)

    DECLARE @TotalAmount money = (
        SELECT SUM(dbo.TotalOrderAmount(o.OrderID))
        FROM Orders o
        WHERE o.CustomerID = @CustomerID AND DATEDIFF(DAY, o.OrderDate, GETDATE()) <=
        ↪  @LastDays
    )

    RETURN CASE WHEN @TotalAmount >= @MinTotalAmount THEN 1 ELSE 0 END
END
GO
```

6.12 GetMenuForDay(Day)

Zwraca ID menu obowiązującego w podanym czasie.

```
CREATE OR ALTER FUNCTION GetMenuForDay(@Day datetime) RETURNS int
BEGIN
    RETURN (SELECT MenuID FROM Menu WHERE Active = 1 AND @Day BETWEEN StartDate AND
    ↪  EndDate)
END
GO
```