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

Projekt bazy danych dla restauracji - etap 4

Szymon Gołębiowski
Dominika Bocheńczyk
Michał Gniadek

2 stycznia 2022

Spis treści

1 Działanie systemu **2**
- 1.1 Funkcje dla klientów . . . 2
- 1.2 Funkcje dla obsługi . . . 2
- 1.3 Funkcje dla kierownictwa . . . 2
- 1.4 Zasady przyznawania rabatów . . . 2

2 Schemat bazy danych **3**

3 Tabele **4**
- 3.1 Firmy . . . 4
- 3.2 Constants . . . 4
- 3.3 Customers . . . 4
- 3.4 Invoices . . . 5
- 3.5 Meals . . . 5
- 3.6 Menu . . . 5
- 3.7 MenuItems . . . 5
- 3.8 OrderDetails . . . 6
- 3.9 OrderDiscounts . . . 6
- 3.10 Orders . . . 6
- 3.11 PrivateCustomers . . . 7
- 3.12 Reservations . . . 7
- 3.13 TableDetails . . . 7
- 3.14 Tables . . . 8

4 Procedury **8**
- 4.1 AddCompanyCustomer(...) . . . 8
- 4.2 AddPrivateCustomer(...) . . . 9
- 4.3 NewMenuInProgress(StartDate, EndData, MenuID OUTPUT) . . . 9
- 4.4 ChangeMenuDates(MenuID, StartDate, EndDate) . . . 9
- 4.5 SetMenuItem(MenuID, MealID, Price) . . . 10
- 4.6 RemoveMenuItem(MenuID, MealID) . . . 10
- 4.7 ActivateMenu(MenuID) . . . 11
- 4.8 CreateOrderInvoice(OrderID) . . . 11
- 4.9 CreateMonthlyInvoice(CustomerID, Month, Year) . . . 12
- 4.10 UpdateConstants(...) . . . 13

5 Widoki **14**
- 5.1 MenusInProgress . . . 14
- 5.2 CurrentOrders . . . 14
- 5.3 OrderHist . . . 14
- 5.4 ReservationsToAccept . . . 14
- 5.5 SeafoodOrders . . . 14

6 Funkcje **15**
- 6.1 TotalOrderAmount(OrderID) . . . 15

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

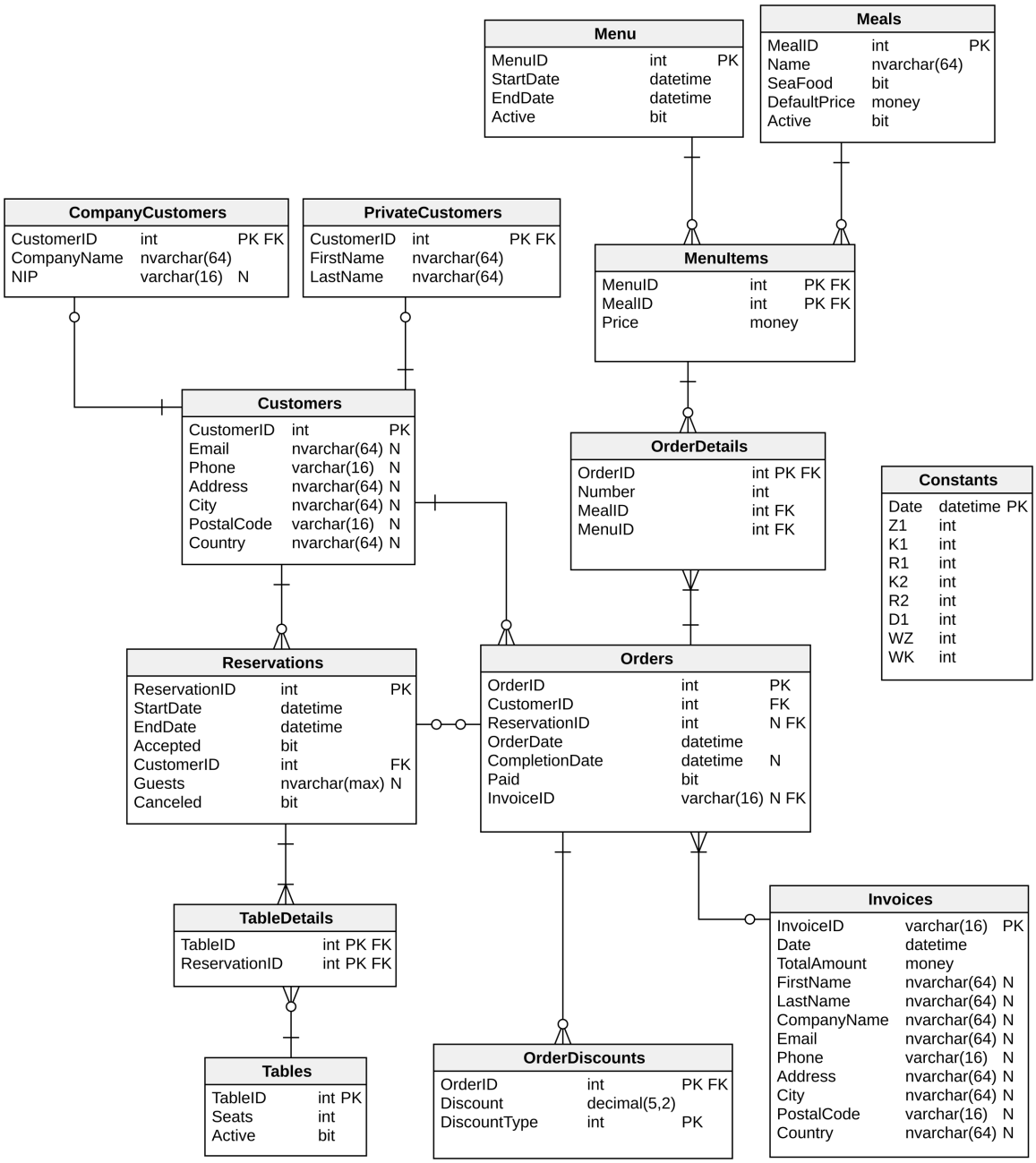
1.3 Funkcje dla kierownictwa

- 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,
    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)
);
```

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,
    Number int  NOT NULL,
    MealID int  NOT NULL,
    MenuID int  NOT NULL,
    CONSTRAINT PositiveMenuNumber CHECK (Number > 0),
    CONSTRAINT OrderDetails_pk PRIMARY KEY  (OrderID)
);

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,
    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.

```sql
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

```sql
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.

```sql
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)

```sql
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.

```sql
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.

```sql
CREATE 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
```

```
    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)
END
GO
```

4.2 AddPrivateCustomer(...)

Dodaje osobę prywatną jako klienta.

```
CREATE 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
    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)
END
GO
```

4.3 NewMenuInProgress(StartDate, EndData, MenuID OUTPUT)

Tworzy nowe nieaktywne menu.

```
CREATE 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
```

4.4 ChangeMenuDates(MenuID, StartDate, EndDate)

Zmienia daty niaktywnego menu.

```
CREATE 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

    SELECT @PrevStartDate = StartDate, @PrevEndDate = EndDate
    FROM Menu WHERE MenuID = @MenuID

    UPDATE Menu
    SET StartDate = ISNULL(@StartDate, @PrevStartDate),
        EndDate = ISNULL(@EndDate, @PrevEndDate)
    WHERE MenuID = @MenuID
END
GO
```

4.5 SetMenuItem(MenuID, MealID, Price)

Dodaje posiłek do nieaktywnego menu.

```
CREATE PROCEDURE SetMenuItem(@MenuID int, @MealID int, @Price money = NULL)
AS BEGIN
    IF (SELECT Active FROM Menu WHERE MenuID = @MenuID) = 1
    BEGIN
        ;THROW 52000, 'Menu is active', 1
        RETURN
    END

    IF (SELECT Active FROM Meals WHERE MealID = @MealID) = 0
    BEGIN
        ;THROW 52000, 'Meal is not active', 1
        RETURN
    END

    DECLARE @DefaultPrice money = (SELECT DefaultPrice FROM Meals WHERE MealID =
    ↪  @MealID)

    INSERT INTO MenuItems(MenuID, MealID, Price)
    VALUES (@MenuID, @MealID, ISNULL(@Price, @DefaultPrice))
END
GO
```

4.6 RemoveMenuItem(MenuID, MealID)

Usuwa posiłek z nieaktynego menu.

```
CREATE PROCEDURE RemoveMenuItem(@MenuID int, @MealID int)
AS BEGIN
    IF (SELECT Active FROM Menu WHERE MenuID = @MenuID) = 1
    BEGIN
        ;THROW 52000, 'Menu is active', 1
        RETURN
    END

    DELETE MenuItems
    WHERE MenuID = @MenuID AND MealID = @MealID
END
GO
```

4.7 ActivateMenu(MenuID)

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

```
CREATE 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
```

4.8 CreateOrderInvoice(OrderID)

Generuje fakturę w tabeli Invoices dla danego zamówienia.

```sql
CREATE 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

    INSERT INTO Invoices(
        Date, TotalAmount, FirstName, LastName, CompanyName, Email, Phone, Address,
        ↪ City, PostalCode, Country
    )
    SELECT GETDATE(), 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
END
GO
```

4.9 CreateMonthlyInvoice(CustomerID, Month, Year)

Generuje fakturę dla danego klienta, dla danego miesiąca.

```sql
CREATE 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

    INSERT INTO Invoices(
        Date, TotalAmount, FirstName, LastName, CompanyName, Email, Phone, Address,
        ↪ City, PostalCode, Country
    )
    SELECT GETDATE(), 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
END
GO
```

4.10 UpdateConstants(...)

Aktualizuje podane stałe (nie zmieniając pozostałych).

```
CREATE 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
```

5 Widoki

5.1 MenusInProgress

Pokazuje nieaktywne menu.

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

5.2 CurrentOrders

Pokazuje zamówienia w trakcie realizacji.

```
CREATE VIEW CurrentOrders
AS SELECT * FROM Orders
WHERE OrderDate <= GETDATE() AND GETDATE() < CompletionDate
GO
```

5.3 OrderHist

Pokazuje historię zamówień.

```
CREATE VIEW OrderHist
AS SELECT * FROM Orders WHERE CompletionDate <= GETDATE()
GO
```

5.4 ReservationsToAccept

Pokazuje rezerwacje, które nie zostały zaakceptowane.

```
CREATE VIEW ReservationsToAccept
AS SELECT * FROM Reservations WHERE Accepted = 0
GO
```

5.5 SeafoodOrders

Pokazuje zamówienia, które zawierają dania z owocami morza.

```
CREATE VIEW SeafoodOrders
AS SELECT *
    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
```

6 Funkcje

6.1 TotalOrderAmount(OrderID)

Zwraca całkowitą cenę zamówienia biorąc pod uwagę rabaty.

```
CREATE 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
```