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

16 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

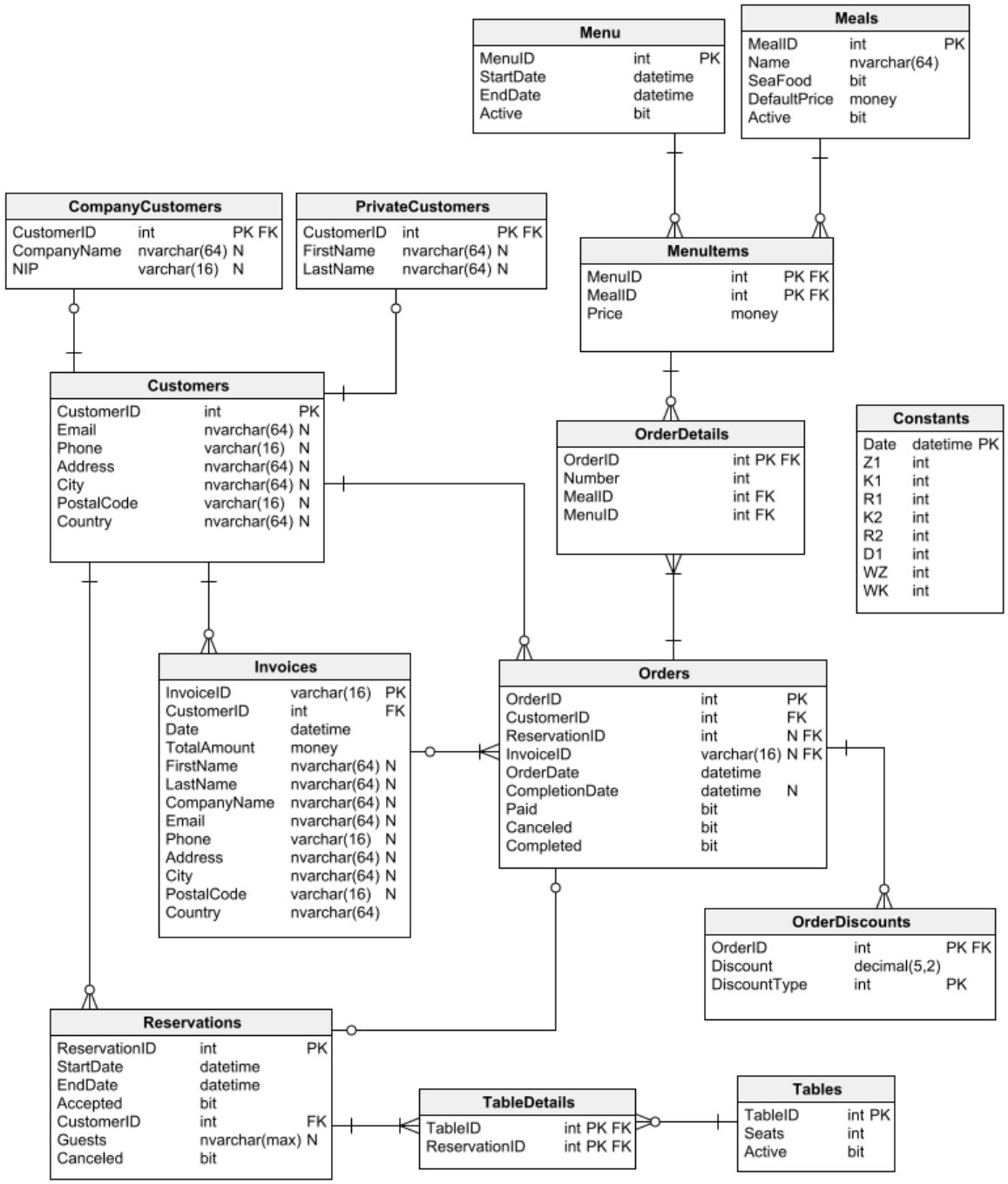
1.3 Funkcje dla kierownictwa

- 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 Uprawnienia

3.1 Manager

Rola Kierownictwa zapewnia największe możliwości działania w bazie danych, zmiany menu, generowanie raportów, możliwość zmiany paramatrów, a także wszystko co może reszta obsługi.

```
CREATE ROLE manager;
GRANT SELECT ON SCHEMA::dbo TO manager;
```

3.2 Staff

Posiada uprawnienia do wykonywania dowolnych operacji które nie modyfikują bazy danych, a także do do procedur obsługujących rezerwacje i zamówienia.

```
CREATE ROLE staff;
GRANT SELECT ON SCHEMA::dbo TO staff;
```

3.3 Customer

Posiada uprawnienia tylko do składania rezerwacji i zamówień, generowaniu niektórych raportów i faktur.

```
CREATE ROLE customer;
```

4 Tabele

4.1 CompanyCustomers

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

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

4.2 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)
);
```

4.3 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)
);
```

4.4 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) NOT NULL,
    City nvarchar(64) NOT NULL,
    PostalCode varchar(16) NOT NULL,
    Country nvarchar(64) NOT 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);
```

4.5 Meals

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

```sql
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)
);
```

4.6 Menu

Przechowuje informacje o menu dostępnych w różnych okresach.

```sql
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)
);
```

4.7 MenuItems

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

```sql
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);
```

4.8 OrderDetails

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

```sql
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);
```

4.9 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);
```

4.10 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,
    Canceled bit NOT NULL,
    Completed 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);
```

4.11 PrivateCustomers

Przechowuje informacje o klientach indywidualnych: imię i nazwisko

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

4.12 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);
```

4.13 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);
```

4.14 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);
```

5 Indeksy

5.1 MenuIndex

Indeks wykorzystujący zakres dostępności menu.

```
CREATE INDEX MenuIndex ON Menu(StartDate, EndDate);
GO
```

5.2 CompanyCustomersIndex

Indeks wykorzystujący nazwę firmy.

```
CREATE INDEX CompanyCustomersIndex ON CompanyCustomers(CompanyName);
GO
```

5.3 PrivateCustomersIndex

Indeks wykorzystujący imię i nazwisko klienta.

```
CREATE INDEX PrivateCustomersIndex ON PrivateCustomers(FirstName, LastName);
GO
```

6 Widoki

6.1 CurrentConstants

Zwraca aktualne wartości stałych rabatowych 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
```

6.2 ReservationsToAccept

Pokazuje rezerwacje, które nie zostały zaakceptowane.

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

6.3 TodaysReservations

Pokazuje rezerwacje na aktualny dzień.

```sql
CREATE OR ALTER VIEW TodaysReservations
AS SELECT ReservationID, CustomerID, StartDate, EndDate, Guests
FROM Reservations
WHERE DAY(StartDate) = DAY(GETDATE())
AND MONTH(StartDate) = MONTH(GETDATE())
AND YEAR(StartDate) = YEAR(GETDATE())
GO
```

6.4 ReservationsDetails

Pokazuje szczegóły rezerwacji.

```sql
CREATE OR ALTER VIEW ReservationsDetails
AS
    ((SELECT ReservationID, CustomerID, StartDate, EndDate, Guests, 'not accepted' as
    ↪  Status
    FROM Reservations
    WHERE Accepted = 0)
    UNION
    (SELECT ReservationID, CustomerID, StartDate, EndDate, Guests, 'accepted'
    FROM Reservations
    WHERE Accepted = 1 AND Canceled = 0 AND GETDATE() <= EndDate)
    UNION
    (SELECT ReservationID, CustomerID, StartDate, EndDate, Guests, 'finished'
    FROM Reservations
    WHERE Accepted = 1 AND Canceled = 0 AND EndDate < GETDATE())
    UNION
    (SELECT ReservationID, CustomerID, StartDate, EndDate, Guests, 'cancelled'
    FROM Reservations
    WHERE Accepted = 1 AND Canceled = 1))
GO
```

6.5 CurrentTables

Podgląd aktualnego stanu stolików.

```sql
CREATE OR ALTER VIEW CurrentTables
AS ((SELECT T.TableID, Seats, 'Available' as Availability, null as ReservationID, null
↪  as EndOfReservation
FROM Tables T
WHERE dbo.TableAvailableAtTime(T.TableID, GETDATE(), GETDATE()) = 1)
UNION
(SELECT T.TableID, Seats, 'Occupied', CurrentTableReservation(T.TableID),
↪  dbo.EndOfTableOccupationTime(T.TableID)
FROM Tables T
```

```
WHERE dbo.TableAvailableAtTime(T.TableID, GETDATE(), GETDATE()) = 0))
GO
```

6.6 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
```

6.7 OrderHist

Pokazuje historię zamówień.

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

6.8 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
```

6.9 CalculatedOrders

Pokazuje wszystkie zamówienia wraz z kwotami.

```
CREATE OR ALTER VIEW CalculatedOrders
AS SELECT
        OrderID,
        CustomerID,
        ReservationID,
        InvoiceID,
        OrderDate,
        CompletionDate,
        (CASE
            WHEN Canceled = 1 THEN 'anulowane'
            WHEN (Paid = 1 AND Completed = 0) THEN 'oplacone przed realizacja'
            WHEN (Paid = 0 AND Completed = 1) THEN 'zrealizowane, do zaplaty'
            WHEN (Paid = 1 AND Completed = 1) THEN 'zrealizowane i oplacone'
            ELSE 'przyjete'
        END
        ) Status,
```

```
        dbo.TotalOrderAmount(OrderID) TotalAmount
    FROM
        Orders
GO
```

6.10 OrdersToCompleteToday

Pokazuje wszystkie zamówienia na dzisiaj, które jeszcze nie zostały zrealizowane.

```
CREATE OR ALTER VIEW OrdersToCompleteToday
AS SELECT
        CustomerID,
        OrderID,
        CompletionDate,
        dbo.TotalOrderAmount(OrderID) TotalAmount
    FROM
        Orders
    WHERE
        DATEDIFF(day, CompletionDate, GETDATE()) = 0
        AND Completed = 0 AND Canceled = 0
GO
```

6.11 MenusInProgress

Pokazuje nieaktywne menu.

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

6.12 CurrentMenu

Zwraca aktualne menu dla zamówień na ten sam dzień

```
CREATE OR ALTER VIEW CurrentMenu
AS
    SELECT MealID, Name, Price
    FROM dbo.GetMenuForDay(GETDATE())
GO
```

6.13 CustomersFullNames

Zwraca CustomerID razem z nazwą firmy lub imieniem i nazwiskiem, zależnie od typu klienta. Podaje także dane adresowe.

```
CREATE OR ALTER VIEW CustomersFullNames
AS SELECT
    Customers.CustomerID, ISNULL(CompanyName, FirstName + ' ' + LastName) AS [Name],
    Email, Phone, Address, City, PostalCode, Country
FROM Customers
    LEFT JOIN CompanyCustomers ON CompanyCustomers.CustomerID = Customers.CustomerID
    LEFT JOIN PrivateCustomers ON PrivateCustomers.CustomerID = Customers.CustomerID
WHERE
    Email IS NOT NULL
GO
```

7 Procedury

7.1 UpdateConstants(...)

Aktualizuje podane stałe (nie zmieniając pozostałych). Domyślnie stałe wchodzą w życie natychmiastowo, ale może zostać podana określona data.

```
CREATE OR ALTER PROCEDURE UpdateConstants(
    @Date datetime = NULL,
    @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 CurrentConstants

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

7.2 AddTable(Seats)

Dodaje nowy stolik.

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

7.3 DisableTable(TableID)

Oznacza stolik jako nieaktywny.

```sql
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
```

7.4 EnableTable(TableID)

Oznacza stolik jako aktywny.

```sql
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
```

7.5 AddReservation(StartDate, EndDate, CustomerID, Guests)

Dodaje nową rezerwację stolika na określony termin

```sql
CREATE OR ALTER PROCEDURE AddReservation (
    @StartDate datetime,
    @EndDate datetime,
    @Accepted bit = 1,
    @CustomerID int,
    @Guests nvarchar(max) = NULL,
    @Tables ReservationTablesListT READONLY,
    @ReservationID int = NULL OUTPUT
```

```
)
AS BEGIN
    BEGIN TRY
    BEGIN TRANSACTION
        IF dbo.AreTablesAvailable(@StartDate, @EndDate, @Tables) = 1
        BEGIN
            INSERT INTO Reservations(StartDate, EndDate, Accepted, CustomerID, Guests,
            ↪  Canceled)
            VALUES (@StartDate, @EndDate, @Accepted, @CustomerID, @Guests, 0)

            SET @ReservationID = @@IDENTITY

            INSERT INTO TableDetails(TableID, ReservationID)
            SELECT TableID, @ReservationID FROM @Tables
        END
    COMMIT
    END TRY
    BEGIN CATCH
        ROLLBACK;
        THROW;
    END CATCH
END
GO
```

7.6 TableReservationNow(CustomerID, EndDate, TableID)

Zarezerwowanie stolika w aktualnej chwili (rezerwacja rozpoczyna się natychmiastowo).

```
CREATE OR ALTER PROCEDURE TableReservationNow (
    @CustomerID int,
    @EndDate datetime,
    @Accepted bit = 1,
    @TableID int,
    @Guests nvarchar(max) = NULL,
    @ReservationID int = NULL OUTPUT
)
AS BEGIN
    BEGIN TRY
    BEGIN TRANSACTION

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

        SET @ReservationID = @@IDENTITY

        INSERT INTO TableDetails(TableID, ReservationID)
        VALUES (@TableID, @ReservationID)

    COMMIT
    END TRY
    BEGIN CATCH
        ROLLBACK;
        THROW;
    END CATCH
END
GO
```

7.7 PrivateOnlineReservation()

Tworzy rezerwację pojedynczego stolika dla klienta indywidualnego wraz ze złożeniem zamówienia

```
CREATE OR ALTER PROCEDURE PrivateOnlineReservation (
    @StartDate datetime,
    @EndDate datetime,
    @Accepted bit = 0,
    @CustomerID int,
    @Guests nvarchar(max) = NULL,
    @Tables ReservationTablesListT READONLY,
    @ReservationID int = NULL OUTPUT
)
AS BEGIN
    BEGIN TRY
    BEGIN TRANSACTION
        IF dbo.AreTablesAvailable(@StartDate, @EndDate, @Tables) = 1
        BEGIN
            INSERT INTO Reservations(StartDate, EndDate, Accepted, CustomerID, Guests,
            ↪  Canceled)
            VALUES (@StartDate, @EndDate, @Accepted, @CustomerID, @Guests, 0)

            SET @ReservationID = @@IDENTITY

            INSERT INTO TableDetails(TableID, ReservationID)
            SELECT TableID, @ReservationID FROM @Tables
        END
    COMMIT
    END TRY
    BEGIN CATCH
        ROLLBACK;
        THROW;
    END CATCH
END


GO
```

7.8 AcceptReservation(ReservationID)

Akceptuje rezerwację złożoną przez formularz internetowy

```
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
```

7.9 CancelReservation(ReservationID)

Anuluje wybraną rezerwację

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

        IF(@ReservationID NOT IN (SELECT ReservationID FROM Reservations))
        BEGIN
            ;THROW 52000, 'Reservation does not exist', 1
            RETURN
        END

        IF(SELECT EndDate FROM Reservations WHERE ReservationID = @ReservationID) <
        ↪ GETDATE()
        BEGIN
            ;THROW 52000, 'Reservation already finished', 1
            RETURN
        END

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

        IF(@ReservationID IN (SELECT ReservationID FROM Orders))
            BEGIN
                DECLARE @OrderID int;
                SET @OrderID = (SELECT OrderID FROM Orders WHERE ReservationID =
                ↪ @ReservationID)
                EXEC CancelOrder @OrderID = @OrderID;
            END

        UPDATE Reservations SET Canceled = 1
        WHERE ReservationID = @ReservationID

    COMMIT
    END TRY
    BEGIN CATCH
        ROLLBACK;
        THROW;
    END CATCH
END
GO
```

7.10 FinishCurrentReservation(ReservationID)

Zakończenie rezerwacji (jeśli klient opuścił restaurację przed końcem rezerwacji)

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

```
    IF NOT (@ReservationID IN (SELECT ReservationID FROM Reservations WHERE StartDate
    ↪  <= GETDATE() AND GETDATE() <= EndDate))
    BEGIN
        ;THROW 52000, 'Wrong ReservationID or reservation has already ended or not
        ↪  started yet', 1
        RETURN
    END
    ELSE
    BEGIN
        UPDATE Reservations SET EndDate = GETDATE()
        WHERE ReservationID = @ReservationID
    END
END
GO
```

7.11 ExtendCurrrentReservation(ReservationID, NewEndDate)

Wydłużenie czasu rezerwacji do preferowanej godziny jeśli to możliwe

```
CREATE OR ALTER PROCEDURE ExtendCurrentReservation(@ReservationID int, @NewEndDate
↪  datetime)
AS BEGIN
    IF NOT (@ReservationID IN (SELECT ReservationID FROM Reservations WHERE StartDate
    ↪  <= GETDATE() AND GETDATE() <= EndDate))
    BEGIN
        ;THROW 52000, 'Wrong ReservationID or reservation has already ended or not
        ↪  started yet', 1
        RETURN
    END
    ELSE
    BEGIN
        IF (( SELECT COUNT (*) FROM (
            (SELECT TableID FROM TableDetails WHERE ReservationID = @ReservationID)
            EXCEPT
            (SELECT TableID FROM TableDetails TD
            INNER JOIN Reservations R ON TD.ReservationID = R.ReservationID
            WHERE dbo.TableAvailableAtTime(TableID, EndDate, @NewEndDate) = 1)  ) as
            ↪  TTI) != 0)
        BEGIN
            ;THROW 52000, 'Extension is not possible for this amount of time', 1
            RETURN
        END
        ELSE
        BEGIN
            UPDATE Reservations SET EndDate = @NewEndDate
            WHERE ReservationID = @ReservationID
        END
    END
END
GO
```

7.12 CreateOrderInvoice(OrderID)

Generuje fakturę w tabeli Invoices dla danego zamówienia.

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

```
3
4       DECLARE @CustomerID int = (SELECT CustomerID FROM Orders WHERE OrderID = @OrderID)
5
6       IF NOT EXISTS (SELECT * FROM Customers WHERE CustomerID = @CustomerID) BEGIN
7           ;THROW 52000, 'The customer does not exist', 1
8           RETURN
9       END
10
11      IF 0 = dbo.CanCreateInvoice(@CustomerID)
12      BEGIN
13          ;THROW 5200, 'The customer have not provided indispensable data to create an
            ↪  invoice', 1
14          RETURN
15      END
16
17      IF (SELECT InvoiceID FROM Orders WHERE OrderID = @OrderID) IS NOT NULL
18      BEGIN
19          ;THROW 5200, 'Order already has an invoice', 1
20          RETURN
21      END
22
23      IF (SELECT Paid FROM Orders WHERE OrderID = @OrderID) = 0
24      BEGIN
25          ;THROW 5200, 'Order has not been paid yet', 1
26          RETURN
27      END
28
29      BEGIN TRY;
30      BEGIN TRANSACTION;
31          INSERT INTO Invoices(
32              Date, CustomerID, TotalAmount, FirstName, LastName, CompanyName, Email,
                ↪  Phone, Address, City, PostalCode, Country
33          )
34          SELECT GETDATE(), Customers.CustomerID, dbo.TotalOrderAmount(@OrderID),
            ↪  FirstName, LastName, CompanyName,
35                  Email, Phone, Address, City, PostalCode, Country
36          FROM Orders
37              JOIN Customers ON Customers.CustomerID = Orders.OrderID
38              LEFT JOIN CompanyCustomers ON CompanyCustomers.CustomerID =
                ↪  Customers.CustomerID
39              LEFT JOIN PrivateCustomers ON PrivateCustomers.CustomerID =
                ↪  Customers.CustomerID
40          WHERE Orders.OrderID = @OrderID;
41
42          UPDATE Orders SET InvoiceID = @@IDENTITY
43          WHERE OrderID = @OrderID
44          COMMIT;
45      END TRY
46      BEGIN CATCH;
47          ROLLBACK;
48          THROW;
49      END CATCH
50  END
51  GO
```

7.13 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 NOT EXISTS (SELECT * FROM Customers WHERE CustomerID = @CustomerID) BEGIN
        ;THROW 52000, 'The customer does not exist', 1
        RETURN
    END

    IF 0 = dbo.CanCreateInvoice(@CustomerID)
    BEGIN
        ;THROW 5200, 'The customer have not provided indispensable data to create an invoice', 1
        RETURN
    END

    IF DATEFROMPARTS(@Year, @Month, DAY(EOMONTH(DATEFROMPARTS(@Year, @Month, 1)))) >= GETDATE()
    BEGIN
        ;THROW 5200, 'The month has not passed yet', 1
        RETURN
    END

    BEGIN TRY;
    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
    END TRY
    BEGIN CATCH;
        ROLLBACK;
        THROW;
    END CATCH
END
GO
```

7.14 CreateOrder

Tworzy nowe zamówienie w systemie. Zamówienie jest przypisane do konkretnego klienta i ma ustaloną datę odbioru. Tworzy nowe zamówienie w systemie i ustawia je jako zrealizowane oraz opłacone. Funkcja jes wykorzystywana, gdy klient kupuje towar na miejscu.

```
CREATE OR ALTER PROCEDURE CreateOrder(
    @CustomerID int,
    @OrderDate datetime = NULL,
    @CompletionDate datetime,
    @OrderedItems OrderedItemsListT READONLY,
    @OrderID int = NULL OUTPUT
)
AS BEGIN
    BEGIN TRY
    BEGIN TRANSACTION

        IF @OrderDate IS NULL
            SET @OrderDate = GETDATE()

        IF @CompletionDate < @OrderDate BEGIN
            ;THROW 52000, 'The completion date is before the order date', 1
            RETURN
        END

        -- 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.GetMenuIDForDay(@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

        -- check if order incluing seafood is placed in enough advance
        IF EXISTS (SELECT * FROM @OrderedItems oi INNER JOIN Meals m ON m.MealID =
        ↪  oi.MealID WHERE m.SeaFood = 1)
        AND 0 = dbo.CanOrderSeafood(@OrderDate, @CompletionDate) BEGIN
            ;THROW 52000, 'The order including seafood must be placed in advance', 1
        END


        INSERT INTO Orders(CustomerID, OrderDate, CompletionDate, Paid, Canceled,
        ↪  Completed)
        VALUES (@CustomerID, @OrderDate, @CompletionDate, 0, 0, 0)

```

```
        SET @OrderID = @@IDENTITY

        INSERT INTO OrderDetails(OrderID, Quantity, MealID, MenuID)
        SELECT @OrderID, Quantity, MealID, @MenuID FROM @OrderedItems

    COMMIT
    END TRY
    BEGIN CATCH
        ROLLBACK;
        THROW;
    END CATCH
END
GO

CREATE OR ALTER PROCEDURE CreateInstantOrder(
    @CustomerID int,
    @CompletionDate datetime,
    @OrderedItems OrderedItemsListT READONLY,
    @OrderID int = NULL OUTPUT
)
AS BEGIN
    BEGIN TRY;
    BEGIN TRANSACTION;

        EXEC CreateOrder
            @CustomerID = @CustomerID,
            @OrderDate = @CompletionDate,
            @CompletionDate = @CompletionDate,
            @OrderedItems = @OrderedItems,
            @OrderID = @OrderID OUTPUT;

        UPDATE Orders SET Completed = 1 WHERE OrderID = @OrderID;
        EXEC PayForOrder @OrderID = @OrderID;

    COMMIT;
    END TRY
    BEGIN CATCH
        ROLLBACK;
        THROW;
    END CATCH
END
GO
```

7.15 CancelOrder

Anuluje zamówienie, które nie zostało jeszcze zrealizowane.

```
CREATE OR ALTER PROCEDURE CancelOrder (@OrderID int)
AS BEGIN
    BEGIN TRY
    BEGIN TRANSACTION

        -- 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
```

```
11
12          -- check if the order has a reservation
13          IF NULL != (SELECT ReservationID FROM Orders WHERE OrderID = @OrderID) BEGIN
14              ;THROW 52000, 'The order cannot be canceled because it has a reservation',
                ↪  1
15              RETURN
16          END
17
18          -- check if the order was completed
19          IF 1 = (SELECT Completed FROM Orders WHERE OrderID = @OrderID) BEGIN
20              ;THROW 52000, 'The order has been already completed', 1
21              RETURN;
22          END
23
24          -- check if the order was canceled
25          IF 1 = (SELECT Canceled FROM Orders WHERE OrderID = @OrderID) BEGIN
26              ;THROW 52000, 'The order has been already canceled', 1
27              RETURN;
28          END
29
30          -- set order as canceled
31          UPDATE Orders SET Canceled = 1 WHERE OrderID = @OrderID
32
33      COMMIT
34      END TRY
35      BEGIN CATCH
36          ROLLBACK;
37          THROW;
38      END CATCH
39  END
40  GO
```

7.16 PayForOrder

Dokonuje płatności za zamówienie i jednocześnie przydziela rabaty, jeśli zostały spełnione warunki.

```
1   CREATE OR ALTER PROCEDURE PayForOrder (@OrderID int)
2   AS BEGIN
3       BEGIN TRY
4       BEGIN TRANSACTION
5
6           -- check if the order exists
7           IF NOT EXISTS (SELECT * FROM Orders WHERE OrderID = @OrderID) BEGIN
8               ;THROW 52000, 'The order does not exist', 1
9               RETURN
10          END
11
12          -- check if the order was paid
13          IF 1 = (SELECT Paid FROM Orders WHERE OrderID = @OrderID) BEGIN
14              ;THROW 52000, 'The order has been already paid', 1
15              RETURN;
16          END
17
18          -- check if the order was canceled
19          IF 1 = (SELECT Canceled FROM Orders WHERE OrderID = @OrderID) BEGIN
20              ;THROW 52000, 'The order was canceled', 1
21              RETURN;
```

```
        END

        -- update status
        UPDATE Orders SET Paid = 1 WHERE OrderID = @OrderID

        DECLARE @CustomerID int = (SELECT CustomerID FROM Orders WHERE OrderID =
        ↪  @OrderID)
        DECLARE @CompletionDate datetime;

        SELECT
            @CustomerID = CustomerID,
            @CompletionDate = CompletionDate
        FROM Orders
        WHERE OrderID = @OrderID

        -- give discount type 1
        IF 1 = dbo.IsDiscountType1(@CustomerID, @CompletionDate) BEGIN
            INSERT INTO OrderDiscounts (OrderID, Discount, DiscountType)
            VALUES (@OrderID, (SELECT R1 FROM CurrentConstants) / 100.0, 1)
        END

        -- give discount type 2
        IF 1 = dbo.IsDiscountType2(@CustomerID, @CompletionDate) BEGIN
            INSERT INTO OrderDiscounts (OrderID, Discount, DiscountType)
            VALUES (@OrderID, (SELECT R2 FROM CurrentConstants) / 100.0, 2)
        END

    COMMIT
    END TRY
    BEGIN CATCH
        ROLLBACK;
        THROW;
    END CATCH
END
GO
```

7.17 CompleteOrder

Zapisuje informację, że zamówienie zostało wydane klientowi.

```
CREATE OR ALTER PROCEDURE CompleteOrder (@OrderID int, @CompletionDate datetime =
↪  NULL)
AS BEGIN
    BEGIN TRY
    BEGIN TRANSACTION

        -- 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

        -- check if the order was canceled
        IF 1 = (SELECT Canceled FROM Orders WHERE OrderID = @OrderID) BEGIN
            ;THROW 52000, 'The order was canceled', 1
            RETURN;
        END
```

```
17
18          -- check if the order was completed
19          IF 1 = (SELECT Completed FROM Orders WHERE OrderID = @OrderID) BEGIN
20              ;THROW 52000, 'The order has been already completed', 1
21              RETURN;
22          END
23
24          UPDATE Orders
25          SET Completed = 1,
26              CompletionDate = ISNULL(@CompletionDate, GETDATE())
27          WHERE OrderID = @OrderID
28
29      COMMIT
30      END TRY
31      BEGIN CATCH
32          ROLLBACK;
33          THROW;
34      END CATCH
35  END
36  GO
```

7.18 NewMeal(Name, IsSeaFood, DefaultPrice, Active = 1, MealID OUTPUT)

Tworzy nowy posiłek.

```
1   CREATE OR ALTER PROCEDURE NewMeal(@Name nvarchar(64), @IsSeaFood bit, @DefaultPrice
    ↪  money, @Active bit = 1, @MealID int OUTPUT)
2   AS BEGIN
3       INSERT INTO Meals([Name], SeaFood, DefaultPrice, Active)
4       VALUES(@Name, @IsSeaFood, @DefaultPrice, @Active)
5
6       SET @MealID = @@IDENTITY
7   END
8   GO
9
10  GRANT EXECUTE ON OBJECT::dbo.NewMeal TO manager
11  GO
```

7.19 SetMealActive(MealID, Active)

Aktywuje lub dezaktywuje posiłek.

```
1   CREATE OR ALTER PROCEDURE SetMealActive(@MealID int, @Active bit)
2   AS BEGIN
3       UPDATE Meals
4       SET Active = @Active
5       WHERE MealID = @MealID
6   END
7   GO
8
9   GRANT EXECUTE ON OBJECT::dbo.SetMealActive TO manager
10  GO
```

7.20 UpdateMealDefaultPrice(MealID, DefaultPrice)

Zmienia podstawową cenę posiłku.

```sql
CREATE OR ALTER PROCEDURE UpdateMealDefaultPrice(@MealID int, @DefaultPrice money)
AS BEGIN
    UPDATE Meals
    SET DefaultPrice = @DefaultPrice
    WHERE MealID = @MealID
END
GO

GRANT EXECUTE ON OBJECT::dbo.UpdateMealDefaultPrice TO manager
GO
```

7.21 NewMenuInProgress(StartDate, EndData, MenuID OUTPUT)

Tworzy nowe nieaktywne menu.

```sql
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

GRANT EXECUTE ON OBJECT::dbo.NewMenuInProgress TO manager
GO
```

7.22 ChangeMenuDates(MenuID, StartDate, EndDate)

Zmienia daty niaktywnego menu.

```sql
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

    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

GRANT EXECUTE ON OBJECT::dbo.ChangeMenuDates TO manager
GO
```

7.23 SetMenuItem(MenuID, MealID, Price)

Dodaje posiłek do nieaktywnego menu.

```
CREATE OR ALTER 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

GRANT EXECUTE ON OBJECT::dbo.SetMenuItem TO manager
GO
```

7.24 RemoveMenuItem(MenuID, MealID)

Usuwa posiłek z nieaktynego menu.

```
CREATE OR ALTER 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

GRANT EXECUTE ON OBJECT::dbo.RemoveMenuItem TO manager
GO
```

7.25 ActivateMenu(MenuID)

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

```
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

GRANT EXECUTE ON OBJECT::dbo.ActivateMenu TO manager
GO
```

7.26 DeactivateMenu(MenuID)

Dezaktywuje menu.

```
CREATE OR ALTER PROCEDURE DeactivateMenu(@MenuID int)
AS BEGIN
    UPDATE Menu SET Active = 0
    WHERE MenuID = @MenuID
```

```
END
GO

GRANT EXECUTE ON OBJECT::dbo.DeactivateMenu TO manager
GO
```

7.27 AddCompanyCustomer(...)

Dodaje firmę jako klienta. Konieczne jest podanie e-maila oraz nazwy firmy. Pozostałe dane mogą zostać uzupełnione później.

```
CREATE OR ALTER PROCEDURE AddCompanyCustomer(
    @Email nvarchar(64),
    @Phone nvarchar(16) = NULL,
    @Address nvarchar(64) = NULL,
    @City nvarchar(64) = NULL,
    @PostalCode varchar(16) = NULL,
    @Country nvarchar(64) = NULL,
    @CompanyName nvarchar(64),
    @NIP varchar(16) = NULL
)
AS BEGIN
    BEGIN TRY;
    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 TRY
    BEGIN CATCH;
        ROLLBACK;
        THROW;
    END CATCH
END
GO
```

7.28 AddPrivateCustomer(...)

Dodaje osobę prywatną jako klienta. Konieczne jest podanie imienia i nazwiska oraz e-maila. Pozostałe dane mogą zostać uzupełnione później.

```
CREATE OR ALTER PROCEDURE AddPrivateCustomer(
    @Email nvarchar(64),
    @Phone nvarchar(16) = NULL,
    @Address nvarchar(64) = NULL,
    @City nvarchar(64) = NULL,
    @PostalCode varchar(16) = NULL,
    @Country nvarchar(64) = NULL,
    @FirstName nvarchar(64),
    @LastName nvarchar(64)
)
AS BEGIN
    BEGIN TRY;
    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 TRY
    BEGIN CATCH;
        ROLLBACK;
        THROW;
    END CATCH
END
GO
```

7.29 UpdateCustomer(...)

Umożliwia zmianę danych klienta. Obsługuje dane wspólne dla klienta firmowego i indywidualnego.

```
CREATE OR ALTER PROCEDURE UpdateCustomer(
    @CustomerID int,

    @Email nvarchar(64) = NULL,
    @Phone nvarchar(16) = NULL,
    @Address nvarchar(64) = NULL,
    @City nvarchar(64) = NULL,
    @PostalCode varchar(16) = NULL,
    @Country nvarchar(64) = NULL
)
AS BEGIN
    BEGIN TRY;
    BEGIN TRANSACTION;

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

        DECLARE @PREV_Email nvarchar(64);
        DECLARE @PREV_Phone nvarchar(16);
        DECLARE @PREV_Address nvarchar(64);
        DECLARE @PREV_City nvarchar(64);
        DECLARE @PREV_PostalCode varchar(16);
        DECLARE @PREV_Country nvarchar(64);

        -- get previous values from Customers
        SELECT
            @PREV_Email = Email,
            @PREV_Phone = Phone,
            @PREV_Address = Address,
            @PREV_City = City,
            @PREV_PostalCode = PostalCode,
            @PREV_Country = Country
        FROM Customers
        WHERE CustomerID = @CustomerID

        -- set new values in Customers
        UPDATE Customers
```

```
        SET Email = ISNULL(@Email, @PREV_Email),
            Phone = ISNULL(@Phone, @PREV_Phone),
            Address = ISNULL(@Address, @PREV_Address),
            City = ISNULL(@City, @PREV_City),
            PostalCode = ISNULL(@PostalCode, @PREV_PostalCode),
            Country = ISNULL(@Country, @PREV_Country)
        WHERE CustomerID = @CustomerID

        COMMIT;
    END TRY
    BEGIN CATCH;
        ROLLBACK;
        THROW;
    END CATCH
END
GO
```

7.30 UpdateCompanyCustomer(...)

Umożliwia zmianę danych klienta firmowego. Pozostałe dane pozostają bez zmian.

```
CREATE OR ALTER PROCEDURE UpdateCompanyCustomer(
    @CustomerID int,

    @Email nvarchar(64) = NULL,
    @Phone nvarchar(16) = NULL,
    @Address nvarchar(64) = NULL,
    @City nvarchar(64) = NULL,
    @PostalCode varchar(16) = NULL,
    @Country nvarchar(64) = NULL,
    @CompanyName nvarchar(64) = NULL,
    @NIP varchar(16) = NULL
)
AS BEGIN
    BEGIN TRY;
    BEGIN TRANSACTION;

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

        -- update values in Customers (common part)
        EXEC UpdateCustomer
            @CustomerID = @CustomerID,
            @Email = @Email,
            @Phone = @Phone,
            @Address = @Address,
            @City = @City,
            @PostalCode = @PostalCode,
            @Country = @Country;

        -- update values in CompanyCustomers
        DECLARE @PREV_CompanyName nvarchar(64);
        DECLARE @PREV_NIP varchar(16);

        SELECT
```

```
            @PREV_CompanyName = CompanyName,
            @PREV_NIP = NIP
        FROM CompanyCustomers
        WHERE CustomerID = @CustomerID

        UPDATE CompanyCustomers
        SET CompanyName = ISNULL(@CompanyName, @PREV_CompanyName),
            NIP = ISNULL(@NIP, @PREV_NIP)
        WHERE CustomerID = @CustomerID

    COMMIT;
    END TRY
    BEGIN CATCH;
        ROLLBACK;
        THROW;
    END CATCH
END
GO
```

7.31 UpdatePrivateCustomer(...)

Umożliwia zmianę danych klienta indywidualnego. Pozostałe dane pozostają bez zmian.

```
CREATE OR ALTER PROCEDURE UpdatePrivateCustomer(
    @CustomerID int,

    @Email nvarchar(64) = NULL,
    @Phone nvarchar(16) = NULL,
    @Address nvarchar(64) = NULL,
    @City nvarchar(64) = NULL,
    @PostalCode varchar(16) = NULL,
    @Country nvarchar(64) = NULL,
    @FirstName nvarchar(64) = NULL,
    @LastName nvarchar(64) = NULL
)
AS BEGIN
    BEGIN TRY;
    BEGIN TRANSACTION;

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

        -- update values in Customers (common part)
        EXEC UpdateCustomer
            @CustomerID = @CustomerID,
            @Email = @Email,
            @Phone = @Phone,
            @Address = @Address,
            @City = @City,
            @PostalCode = @PostalCode,
            @Country = @Country;

        -- update values in PrivateCustomers
        DECLARE @PREV_FirstName nvarchar(64);
        DECLARE @PREV_LastName nvarchar(64);
```

```
35
36          SELECT
37              @PREV_FirstName = FirstName,
38              @PREV_LastName = LastName
39          FROM PrivateCustomers
40          WHERE CustomerID = @CustomerID
41
42          UPDATE PrivateCustomers
43          SET FirstName = ISNULL(@FirstName, @PREV_FirstName),
44              LastName = ISNULL(@LastName, @PREV_LastName)
45          WHERE CustomerID = @CustomerID
46
47      COMMIT;
48      END TRY
49      BEGIN CATCH;
50          ROLLBACK;
51          THROW;
52      END CATCH
53  END
54  GO
```

7.32 ForgetCustomer(...)

Umożliwia usunięcie danych klienta bez utraty spójności bazy danych (pozostaje tylko CustomerID)

```
1   CREATE OR ALTER PROCEDURE ForgetCustomer(@CustomerID INT)
2   AS BEGIN
3       BEGIN TRY;
4       BEGIN TRANSACTION;
5
6       IF NOT EXISTS (SELECT * FROM Customers WHERE CustomerID = @CustomerID) BEGIN
7           ;THROW 52000, 'The customer does not exist', 1
8           RETURN
9       END
10
11      UPDATE Customers
12      SET
13          Email = NULL,
14          Phone = NULL,
15          Address = NULL,
16          City = NULL,
17          PostalCode = NULL,
18          Country = NULL
19      WHERE CustomerID = @CustomerID
20
21      UPDATE PrivateCustomers
22      SET
23          FirstName = NULL,
24          LastName = NULL
25      WHERE CustomerID = @CustomerID
26
27      UPDATE CompanyCustomers
28      SET
29          CompanyName = NULL,
30          NIP = NULL
31      WHERE CustomerID = @CustomerID
32
```

```
    COMMIT;
    END TRY
    BEGIN CATCH;
        ROLLBACK;
        THROW;
    END CATCH
END
GO
```

8 Funkcje

8.1 AreTablesAvailable(StartDate, EndDate, Tables)

Sprawdza czy wszystkie stoliki z listy są dostępne w danym przedziale czasowym.

```
CREATE OR ALTER FUNCTION AreTablesAvailable(@StartDate datetime, @EndDate datetime,
↪  @Tables ReservationTablesListT READONLY)
RETURNS BIT
BEGIN
    IF (( SELECT COUNT (*) FROM ( (SELECT TableID FROM @Tables) EXCEPT (SELECT TableID
    ↪  FROM Tables WHERE dbo.TableAvailableAtTime(TableID, @StartDate, @EndDate) = 1)
    ↪  ) as TTI) != 0)
    BEGIN
        ;THROW 52000, 'Not all selected tables will be available', 1
        RETURN 0
    END
    RETURN 1
END
GO
```

8.2 ReservationDetails(ReservationID)

Zwraca szczegóły na temat danej rezerwacji.

```
CREATE OR ALTER FUNCTION ReservationDetails(@ReservationID int)
RETURNS @Details TABLE (ReservationID int, CustomerID int, StartDate datetime, EndDate
↪  datetime, Guests nvarchar(max), Status varchar(16))
BEGIN
    INSERT @Details
    SELECT ReservationID, CustomerID, StartDate, EndDate, Guests, Status FROM
    ↪  ReservationDetails WHERE ReservationID = @ReservationID
    ORDER BY StartDate
    RETURN
END
GO
```

8.3 AllClientReservations(CustomerID)

Zwraca informacje na temat wszystkich rezerwacji danego klienta wraz z informacją o ich statusie.

```
CREATE OR ALTER FUNCTION AllClientReservations(@CustomerID int)
RETURNS @AllReservations TABLE (CustomerID int, ReservationID int, StartDate datetime,
↪  EndDate datetime, Guests nvarchar(max), Status varchar(16))
BEGIN
    INSERT @AllReservations
```

```
    SELECT CustomerID, ReservationID, StartDate, EndDate, Guests, Status FROM
    ↪ ReservationDetails WHERE CustomerID = @CustomerID
    ORDER BY StartDate
    RETURN
END
GO
```

8.4 TableAvailableAtTime(TableID, StartDate, EndDate)

Sprawdza czy dany stolik jest dostępny w danym przedziale czasowym.

```
CREATE OR ALTER FUNCTION TableAvailableAtTime(@TableID int, @StartDate datetime,
↪ @EndDate datetime)
RETURNS BIT
BEGIN
    IF ( @TableID NOT IN
    (SELECT TableID FROM TableDetails TD
    INNER JOIN Reservations R ON TD.ReservationID = R.ReservationID
    WHERE ((NOT (EndDate <= @StartDate OR StartDate >= @EndDate)) AND Canceled = 0)) )
    RETURN 1
    ELSE
    RETURN 0
END
GO
```

8.5 EndOfTableOccupationTime(TableID)

Zwraca czas zakończenia rezerwacji jeśli stolik jest aktualnie zarezerwowany.

```
CREATE OR ALTER FUNCTION EndOfTableOccupationTime(@TableID int)
RETURNS datetime
BEGIN
    IF (TableAvailableAtTime(@TableID, GETDATE(), GETDATE()) = 1)
    BEGIN
        ;THROW 52000, 'Table is not occupied at the moment', 1
        RETURN 0
    END
    ELSE
    BEGIN
        RETURN (SELECT EndDate FROM Reservations R
        INNER JOIN TableDetails TD on R.ReservationID = TD.ReservationID
        WHERE TD.TableID = @TableID AND StartDate <= GETDATE() AND GETDATE() <=
        ↪ EndDate)
    END
END
GO
```

8.6 CurrentTableReservation(TableID)

Zwraca numer rezerwacji jeśli stolik jest aktualnie zarezerwowany.

```
CREATE OR ALTER FUNCTION CurrentTableReservation(@TableID int)
RETURNS int
BEGIN
    IF (TableAvailableAtTime(@TableID, GETDATE(), GETDATE()) = 1)
    BEGIN
```

```
        ;THROW 52000, 'Table is not occupied at the moment', 1
        RETURN 0
    END
    ELSE
    BEGIN
        RETURN (SELECT R.ReservationID FROM Reservations R
        INNER JOIN TableDetails TD on R.ReservationID = TD.ReservationID
        WHERE TD.TableID = @TableID AND StartDate <= GETDATE() AND GETDATE() <=
        ↪  EndDate)
    END
END
GO
```

8.7 TablesAvailableToReserve(StartDate, EndDate)

Zwraca tabelę zawierającą stoliki możliwe do zarezerwowania w danym przedziale czasowym.

```
CREATE OR ALTER FUNCTION TablesAvailableToReserve(@StartDate datetime, @EndDate
↪  datetime)
RETURNS @Tables TABLE (TableID int, Seats int)
BEGIN
    INSERT @Tables
        SELECT TableID, Seats FROM Tables WHERE dbo.TableAvailableAtTime(TableID,
        ↪  @StartDate, @EndDate) = 1
    RETURN
END
GO
```

8.8 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.

```
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
```

8.9 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
```

8.10 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
```

8.11 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
```

8.12 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
```

8.13 CanCreateInvoice

Sprawdza czy dany klient ma uzupełnione wszystkie dane konieczne do wygenerowania faktury.

```sql
CREATE OR ALTER FUNCTION CanCreateInvoice(@CustomerID int) RETURNS bit
BEGIN
    RETURN CASE WHEN EXISTS (
        SELECT *
        FROM Customers c
            LEFT JOIN CompanyCustomers cc ON cc.CustomerID = c.CustomerID
            LEFT JOIN PrivateCustomers pc ON pc.CustomerID = c.CustomerID
        WHERE
            ((pc.FirstName != NULL AND pc.LastName != NULL) OR cc.CompanyName != NULL)
            ↪  AND
            c.Address != NULL AND
            c.City != NULL AND
            c.PostalCode != NULL AND
            c.Country != NULL AND
            c.CustomerID = @CustomerID
        ) THEN 1 ELSE 0 END
END
GO
```

8.14 TotalDiscountForOrder

Zwraca całkowity rabat (w

```sql
CREATE OR ALTER FUNCTION TotalDiscountForOrder(@OrderID int) RETURNS decimal(5, 2)
BEGIN
    RETURN (
        SELECT COALESCE(SUM(Discount), 0)
        FROM OrderDiscounts
        WHERE OrderID = @OrderID
    )
END
GO
```

8.15 TotalOrderAmount(OrderID)

Zwraca całkowitą cenę zamówienia biorąc pod uwagę rabaty.

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

8.16 CanOrderSeafood(OrderDate, CompletionDate)

Zwraca informację czy w dniu OrderDate można złożyć zamówienie na owoce morza, które ma zostać odebrane w dniu CompletionDate

```sql
CREATE OR ALTER FUNCTION CanOrderSeafood(@OrderDate datetime, @CompletionDate
↪  datetime) RETURNS bit
BEGIN

    IF NOT DATENAME(weekday, @CompletionDate) IN ('Thursday', 'Friday', 'Saturday')
        RETURN 0;

    IF NOT (@OrderDate < @CompletionDate AND (
        DATENAME(week, @OrderDate) < DATENAME(week, @CompletionDate) OR
            DATENAME(weekday, @OrderDate) IN ('Sunday', 'Monday')))
        RETURN 0;

    RETURN 1;
END
GO


--> Funkcje
--# IsDiscountType1(CustomerID)
--- Sprawdza czy klientowi przysługuje w danej chwili rabat typu pierwszego (co
↪  najmniej Z1 zamówień za kwotę przynajmniej K1)
--- na zamówienie dokonane w danym terminie.
CREATE OR ALTER FUNCTION IsDiscountType1(@CustomerID int, @CheckDate datetime) 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 AND
            o.Completed = 1 AND
            o.CompletionDate < @CheckDate
    )

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

8.17 IsDiscountType2(CustomerID)

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

```sql
CREATE OR ALTER FUNCTION IsDiscountType2(@CustomerID int, @CheckDate datetime) 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 AND
            o.Completed = 1 AND
            o.CompletionDate < @CheckDate
    )

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

8.18 CustomerOrders

Pokazuje wszystkie zamówienia danego klienta

```
CREATE OR ALTER FUNCTION CustomerOrders(@CustomerID int)
RETURNS @MyOrders TABLE(
    ReservationID int,
    InvoiceID varchar(16),
    OrderDate datetime,
    CompletionDate datetime,
    Status nvarchar(64),
    TotalAmount money
)
BEGIN
    INSERT @MyOrders
        SELECT
            ReservationID,
            InvoiceID,
            OrderDate,
            CompletionDate,
            Status,
            TotalAmount
        FROM
            CalculatedOrders
        WHERE
            CustomerID = @CustomerID
        ORDER BY CompletionDate DESC
    RETURN
END
GO
```

8.19 GetMenuIDForDay(Day)

Zwraca ID menu obowiązującego w podanym czasie.

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

8.20 GetMenuOrders(MenuID)

Zwraca zamówienia korzystające z danego menu, a także klientów którzy je złożyli razem z danymi kontaktowymi.

```sql
CREATE OR ALTER FUNCTION GetMenuOrders(@MenuID int)
RETURNS @MenuOrders TABLE(
    OrderID int,
    CompletionDate datetime,
    CustomerID int,
    [Name] nvarchar(256),
    Phone nvarchar(16),
    Email nvarchar(64))
BEGIN
    DECLARE @StartDate datetime;
    DECLARE @EndDate datetime;

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

    INSERT @MenuOrders
    SELECT OrderID, CompletionDate, Customers.CustomerID, [Name], Phone, Email
    From Orders
    JOIN CustomerNames ON CustomerNames.CustomerID = Orders.CustomerID
    JOIN Customers ON Customers.CustomerID = Orders.CustomerID
    WHERE CompletionDate BETWEEN @StartDate AND @EndDate
    RETURN
END
GO
```

8.21 GetMenuForDay

Zwraca menu dostępne w danym dniu w przyszłości.

```sql
CREATE OR ALTER FUNCTION GetMenuForDay(@Date datetime)
RETURNS @DayMenu TABLE(
    MealID int,
    Name nvarchar(64),
    SeaFood varchar(4),
    Price money
)
BEGIN

    DECLARE @MenuID int = dbo.GetMenuIDForDay(@Date);
    DECLARE @ShowSeaFood bit = dbo.CanOrderSeafood(GETDATE(), @Date)

    INSERT @DayMenu
        SELECT
            m.MealID,
            m.Name,
            (CASE WHEN m.SeaFood = 1 THEN 'TAK' ELSE 'NIE' END) SeaFood,
            Price
        FROM Meals m
            INNER JOIN MenuItems mi ON mi.MealID = m.MealID
        WHERE
            mi.MenuID = @MenuID
            AND (m.SeaFood = 0 OR @ShowSeaFood = 1)
    RETURN
END
GO
```