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

Baza danych do obsługi restauracji

Szymon Gołębiowski
Dominika Bocheńczyk
Michał Gniadek

23 stycznia 2022

Spis treści

1 Projekt

Prezentowany projekt jest systemem wspomagającym działalność firmy z branży gastronomicznej, która świadczy usługi dla klientów indywidualnych oraz firm. Szczegółowe zasady działania oparte są na dostarczonym poleceniu.

Funkcjonalność systemu obejmuje następujące obszary działania firmy:

- **Klienci** - ewidencja klientów, którzy korzystają z usług firmy. Przechowywane są ich dane osobowe.
- **Menu** - baza danych przechowuje informacje o oferowanych posiłkach i cenach. Menu może zmieniać się w różnych okresach czasu.
- **Zamówienia** - system przetwarza wszystkie zamówienia obsługiwane przez firmę. Rabaty naliczane są automatycznie według ustalonych zasad.
- **Rezerwacje** - system kontroluje rezerwacje stolików w restauracji. Umożliwia przyjmowanie rezerwacji o różnym czasie trwania i gwarantuje, że nie pokrywają się one wzajemnie.
- **Faktury** - możliwe jest wystawianie faktur (pojedynczych lub zbiorczych) na podstawie zrealizowanych zamówień.
- **Raporty** - system umożliwia automatyczne generowanie raportów dotyczących różnych obszarów bieżącej działalności firmy.

2 Użytkownicy

W bazie danych przewidziana została współpraca z trzema rodzajami użytkowników. Każdy użytkownik korzysta z bazy poprzez warstwę aplikacji, która może wykonywać określone funkcje w bazie danych.

- **manager** - Aplikacja managera restauracji może wykonywać dowolne operacje SELECT w celu pobierania danych. Posiada również dostęp do procedur, funkcji i widoków, które związane są z zarządzaniem restauracją.Umożliwiają one bezpieczne i niepowodujące błędów tworzenie menu oraz zmianę zasad rabatów przyznawanych w restauracji. Pozwalają także na generowanie raportów związanych z działaniem restauracji oraz na wystawianie faktur.
- **obsługa** - Aplikacja przeznaczona dla obsługi może wykonywać dowolne operacje SELECT w celu pobierania potrzebnych danych. Ponadto ma dostęp do odpowiednio przygotowanych procedur, które w bezpieczny sposób przeprowadzają operacje związane z bieżącą obsługą restauracji, czyli składaniem i wydawaniem zamówień oraz rezerwacjami stolików.
- **klienci** - Aplikacja klienta ma dostęp do wybranych procedur umożliwiających składanie zamówień i rezerwacji online, generowanie faktur oraz raportów na temat własnych zamówień. Nie ma natomiast możliwości wykonywania własnych operacji SELECT. Wszystkie dane odczytywane z bazy muszą zostać wyciągnięte poprzez odpowiednio przygotowane funkcje i widoki.

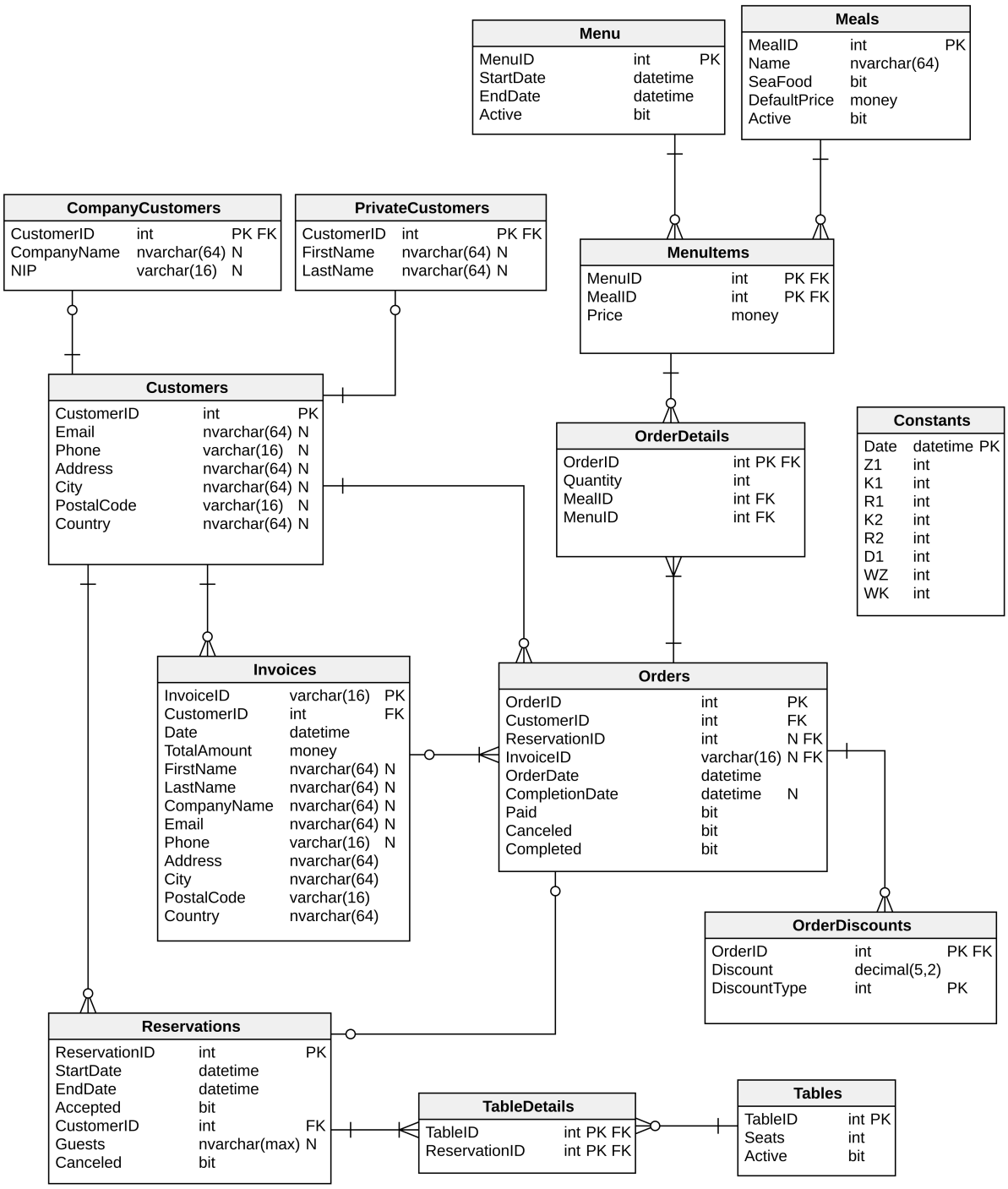
3 Lista funkcjonalności

W poniższej tabeli znajduje się lista funkcjonalności jakie udostępnia baza danych. Do każdej z funkcjonalności przypisany jest element systemu (procedura, funkcja lub widok), który realizuje daną funkcjonalność. Ponadto w ostatniej kolumnie znajduje się także informacja które typy użytkowników mają możliwość korzystania z danej funkcjonalności.

realizacja w systemie	funkcjonalność	uprawnienia
procedura CreateNewMeal	Dodanie nowego posiłku do listy dostępnych.	manager
procedura SetMealActive	Zmiana statusu posiłku na aktywny lub nieaktywny. - od tego zależy czy może być użyty w nowotworzonym menu	manager
procedura UpdateMealDefaultPrice	Zmiana domyślnej ceny produktu.	manager
procedura NewMenuInProgress	Rozpoczęcie tworzenia nowego menu na dany okres.	manager
procedura ChangeMenuDates	Zmiana czasu obowiązywania nieaktywnego menu.	manager
procedura SetMenuItem	Dodanie produktu z określoną ceną do nieaktywnego menu.	manager
procedura RemoveMenuItem	Usunięcie produktu z nieaktywnego menu.	manager
procedura ActivateMenu	Aktywacja utworzonego menu na dany okres. - sprawdzenie poprawności menu (czy wymieniono określoną liczbę produktów) - menu staje się widoczne dla klientów i mogą składać zamówienia - sprawdzenie czy na dany okres istnieje tylko jedno menu	manager
procedura DeactivateMenu	Deaktywacja menu na dany okres.	manager
widok GetMenuOrders	Sprawdzenie wszystkich zamówień złożonych na dane menu oraz klientów, którzy je złożyli.	manager
widok CurrentMenu	Podgląd menu obowiązującego w danej chwili.	manager obsługa klienci
funkcja GetMenuForDay	Podgląd menu zaplanowanego na dany dzień w przyszłości. - dania z owocami morza wyświetlane są tylko wtedy, gdy jeszcze możliwe jest ich zamówienie	manager obsługa klienci
procedura CreateOrder	Złożenie zamówienia z przyszłą datą realizacji. - musi być zachowany warunek o składaniu zamówień na dania z owocami morza z odpowiednim wyprzedzeniem	obsługa klienci
procedura CreateInstantOrder	Złożenie zamówienia z natychmiastowym czasem realizacji (sprzedaż towaru na miejscu). - zamówienie otrzymuje domyślnie status zrealizowanego	obsługa
procedura CancelOrder	Anulowanie zamówienia, które nie zostało jeszcze zrealizowane.	obsługa klienci
procedura PayForOrder	Opłacenie zamówienia. - jednocześnie następuje sprawdzenie warunków i jeśli zostaną spełnione - przyznanie rabatów	obsługa klienci
procedura CompleteOrder	Realizacja zamówienia.	obsługa
procedura CreateOrderInvoice	Wygenerowanie faktury na pojedyncze zamówienie. - następuje sprawdzenie czy dane zamówienie nie ma już wygenerowanej faktury	manager obsługa klienci
procedura CreateMonthlyInvoice	Wygenerowanie faktury zbiorczej za podany miesiąc. - uwzględniane są tylko zamówienia, na które nie została jeszcze wygenerowana żadna faktura	manager obsługa klienci
funkcja CustomerOrders	Podgląd wszystkich zamówień danego klienta.	obsługa klienci
funkcja GetOrderDetails	Pokazanie szczegółow konkretnego zamówienia.	obgługa klienci
widok OrdersToCompleteToday	Podgląd zamówień do realizacji w aktualnym dniu.	obsługa
widok CurrentWeekSeaFoodOrders	Podgląd zamówień z owocami morza do realizacji w aktualnym tygodniu.	obsługa
widok CalculatedOrders	Podgląd wszystkich zamówień wraz z obliczonymi kwotami.	obsługa
procedura AddTable	Dodanie nowego stolika.	manager
procedura EnableTable	Aktywacja stolika - możliwe są rezerwacje tego stolika.	manager
procedura DisableTable	Deaktywacja stolika - rezerwowanie tego stolika przestaje być możliwe.	manager
procedura AddReservation	Dodanie nowej rezerwacji na dowolny termin. - możliwa jest rezerwacja wielu stolików - istnieje możliwość zapisania listy gości	obsługa
procedura AddInstantReservation	Zarezerwowanie stolika na teraz (rezerwacja rozpoczyna się natychmiastowo).	obsługa

procedura FinishCurrentReservation	Skrócenie czasu rezerwacji (gdy klient opuścił restaurację przed końcem rezerwacji). - może być wykonana tylko w trakcie trwania danej rezerwacji	obsługa
procedura ExtendCurrentReservation	Wydłużenie czasu aktualnie trwającej rezerwacji.	obsługa
procedura PrivateOnlineReservation	Rezerwacja pojedynczego stolika dla klienta indywidualnego wraz ze złożeniem zamówienia. - sprawdzenie czy składane zamówienie jest na kwotę co najmniej WZ - sprawdzenie czy klient dokonał wcześniej co najmniej WK zamówień - domyślnie rezerwacja jest niezaakceptowana	klienci
procedura CompanyOnlineReservation	Rezerwacja dowolnej liczby stolików przez klienta firmowego (opcjonalnie ze złożeniem zamówienia). - możliwe jest podanie listy gości (pracowników firmy) - domyślnie rezerwacja jest niezaakceptowana	klienci
procedura CancelReservation	Anulowanie rezerwacji. - jeśli rezerwacja jest połączona z zamówieniem, to również ulega ono anulowaniu.	obsługa klienci
procedura AcceptReservation	Akceptacja rezerwacji złożonej przez formularz internetowy.	obsługa
funkcja TablesAvailableToReserve	Pokazanie stolików dostępnych o danej godzinie przez co najmniej określoną ilość czasu.	obsługa klienci
widok CurrentTables	Podgląd aktualnego stanu stolików.	obsługa
widok TodayReservations	Podgląd rezerwacji na dzisiejszy dzień.	obsługa
widok ReservationsToAccept	Podgląd rezerwacji oczekujących na akceptację.	obsługa
funkcja GetCustomersReservations	Podgląd wszystkich rezerwacji danego klienta wraz z informacją o ich statusie.	obsługa klienci
funkcja SingleReservationDetails	Podgląd szczegółów konkretnej rezerwacji.	obsługa klienci
widok ReservationsDetails	Podgląd wszstkich rezerwacji wraz z ich statusem oraz liczbą zajętych stolików i miejsc.	obsługa
procedura AddPrivateCustomer	Dodanie nowego klienta indywidualnego.	obsługa klienci
procedura AddCompanyCustomer	Dodanie nowego klienta firmowego.	obsługa klienci
procedura UpdatePrivateCustomer	Uzupełnienie lub edycja danych klienta indywidualnego.	obsługa klienci
procedura UpdateCompanyCustomer	Uzupełnienie lub edycja danych klienta firmowego.	obsługa klienci
procedura ForgetCustomer	Usunięcie klienta. - funkcja nie usuwa rekordu klienta z bazy, ale usuwa wszystkie dane osobowe i adresowe - jest to realizacja prawa klienta do usunięcia danych osobowych bez utraty spójności bazy danych	obsługa
widok CustomersFullNames	Pokazanie danych adresowych wszystkich klientów, którzy nie zostali usunięci.	manager obsługa
funkcja MealsStatistics	Raport pokazujący statystyki sprzedaży poszczególnych produktów (tygodniowy lub miesięczny)	manager
funkcja CustomerStatistics	Raport pokazujący zamówienia danego klienta w danym okresie czasu (tygodniowy lub miesięczny)	manager klienci
funkcja OrdersStatistics	Raport dotyczący zamówień zrealizowanych w danym okresie czasu (tygodniowy lub miesięczny)	manager
funkcja TablesStatistics	Raport dotyczący wykorzystania stolików w danym okresie czasu (tygodniowy lub miesięczny)	manager
funkcja DiscountsStatistics	Raport dotyczący przydzielonych rabatów w danym okresie czasu (tygodniowy lub miesięczny)	manager
procedura UpdateConstants	Ustawienie nowych wartości stałych rabatowych	manager
widok CurrentConstants	Podgląd wartości stałych rabatowych obowiązujących aktualnie	manager obsługa klienci

4 Schemat bazy danych



5 Uprawnienia

5.1 Manager

Może wykonywać dowolne operacje SELECT oraz korzystać z przygotowanych procedur, funkcji i widoków.

```
CREATE ROLE manager;

GRANT SELECT ON SCHEMA::dbo TO manager;

-- procedures
GRANT EXEC ON dbo.CreateNewMeal TO manager;
GRANT EXEC ON dbo.SetMealActive TO manager;
GRANT EXEC ON dbo.UpdateMealDefaultPrice TO manager;
GRANT EXEC ON dbo.NewMenuInProgress TO manager;
GRANT EXEC ON dbo.ChangeMenuDates TO manager;
GRANT EXEC ON dbo.SetMenuItem TO manager;
GRANT EXEC ON dbo.RemoveMenuItem TO manager;
GRANT EXEC ON dbo.ActivateMenu TO manager;
GRANT EXEC ON dbo.DeactivateMenu TO manager;
GRANT EXEC ON dbo.CreateNewMeal TO manager;
GRANT EXEC ON dbo.CreateOrderInvoice TO manager;
GRANT EXEC ON dbo.CreateMonthlyInvoice TO manager;
GRANT EXEC ON dbo.AddTable TO manager;
GRANT EXEC ON dbo.EnableTable TO manager;
GRANT EXEC ON dbo.DisableTable TO manager;
GRANT EXEC ON dbo.UpdateConstants TO manager;

-- functions
GRANT SELECT ON dbo.GetMenuForDay TO manager;
GRANT SELECT ON dbo.MealsStatistics TO manager;
GRANT SELECT ON dbo.CustomerStatistics TO manager;
GRANT SELECT ON dbo.OrdersStatistics TO manager;
GRANT SELECT ON dbo.TablesStatistics TO manager;
GRANT SELECT ON dbo.DiscountsStatistics TO manager;

-- views
GRANT SELECT ON dbo.CurrentMenu TO manager;
GRANT SELECT ON dbo.GetMenuOrders TO manager;
GRANT SELECT ON dbo.CurrentConstants TO manager;
GRANT SELECT ON dbo.CustomersFullNames TO manager;
```

5.2 Staff

Może wykonywać dowolne operacje SELECT oraz korzystać z przygotowanych procedur, funkcji i widoków.

```
CREATE ROLE staff;

GRANT SELECT ON SCHEMA::dbo TO staff;

-- procedures
GRANT EXEC ON dbo.CreateOrder TO staff;
GRANT EXEC ON dbo.CreateInstantOrder TO staff;
GRANT EXEC ON dbo.CancelOrder TO staff;
GRANT EXEC ON dbo.PayForOrder TO staff;
GRANT EXEC ON dbo.CompleteOrder TO staff;
```

```
GRANT EXEC ON dbo.CreateOrderInvoice TO staff;
GRANT EXEC ON dbo.CreateMonthlyInvoice TO staff;
GRANT EXEC ON dbo.AddReservation TO staff;
GRANT EXEC ON dbo.AddInstantReservation TO staff;
GRANT EXEC ON dbo.FinishCurrentReservation TO staff;
GRANT EXEC ON dbo.ExtendCurrentReservation TO staff;
GRANT EXEC ON dbo.CancelReservation TO staff;
GRANT EXEC ON dbo.AcceptReservation TO staff;
GRANT EXEC ON dbo.AddPrivateCustomer TO staff;
GRANT EXEC ON dbo.AddCompanyCustomer TO staff;
GRANT EXEC ON dbo.UpdatePrivateCustomer TO staff;
GRANT EXEC ON dbo.UpdateCompanyCustomer TO staff;
GRANT EXEC ON dbo.ForgetCustomer TO staff;

-- functions
GRANT SELECT ON dbo.GetMenuForDay TO staff;
GRANT SELECT ON dbo.CustomerOrders TO staff;
GRANT SELECT ON dbo.GetOrderDetails TO staff;
GRANT SELECT ON dbo.GetCustomersReservations TO staff;
GRANT SELECT ON dbo.SingleReservationDetails TO staff;
GRANT SELECT ON dbo.TablesAvailableToReserve TO staff;

-- views
GRANT SELECT ON dbo.CurrentMenu TO staff;
GRANT SELECT ON dbo.OrdersToCompleteToday TO staff;
GRANT SELECT ON dbo.CurrentWeekSeaFoodOrders TO staff;
GRANT SELECT ON dbo.CalculatedOrders TO staff;
GRANT SELECT ON dbo.CurrentTables TO staff;
GRANT SELECT ON dbo.TodayReservations TO staff;
GRANT SELECT ON dbo.ReservationsToAccept TO staff;
GRANT SELECT ON dbo.ReservationsDetails TO staff;
GRANT SELECT ON dbo.CustomersFullNames TO staff;
GRANT SELECT ON dbo.CurrentConstants TO staff;
```

5.3 Customer

Może korzystać wyłącznie z przygotowanych procedur, funkcji i widoków.

```
CREATE ROLE customer;

-- procedures
GRANT EXEC ON dbo.CreateOrder TO customer;
GRANT EXEC ON dbo.CancelOrder TO customer;
GRANT EXEC ON dbo.PayForOrder TO customer;
GRANT EXEC ON dbo.CreateOrderInvoice TO customer;
GRANT EXEC ON dbo.CreateMonthlyInvoice TO customer;
GRANT EXEC ON dbo.PrivateOnlineReservation TO customer;
GRANT EXEC ON dbo.CompanyOnlineReservation TO customer;
GRANT EXEC ON dbo.CancelReservation TO customer;
GRANT EXEC ON dbo.AddPrivateCustomer TO customer;
GRANT EXEC ON dbo.AddCompanyCustomer TO customer;
GRANT EXEC ON dbo.UpdatePrivateCustomer TO customer;
GRANT EXEC ON dbo.UpdateCompanyCustomer TO customer;

-- functions
GRANT SELECT ON dbo.GetMenuForDay TO customer;
GRANT SELECT ON dbo.CustomerOrders TO customer;
```

```
GRANT SELECT ON dbo.GetOrderDetails TO customer;
GRANT SELECT ON dbo.GetCustomersReservations TO customer;
GRANT SELECT ON dbo.SingleReservationDetails TO customer;
GRANT SELECT ON dbo.TablesAvailableToReserve TO customer;
GRANT SELECT ON dbo.CustomerStatistics TO customer;

-- views
GRANT SELECT ON dbo.CurrentMenu TO customer;
GRANT SELECT ON dbo.CurrentConstants TO customer;
```

6 Tabele

6.1 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)
);
```

6.2 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)
);
```

6.3 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)
);
```

6.4 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);
```

6.5 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)
);
```

6.6 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)
);
```

6.7 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);
```

6.8 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);
```

6.9 OrderDiscounts

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

```sql
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);
```

6.10 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,
    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);
```

6.11 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)
);
```

6.12 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);
```

6.13 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);
```

6.14 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);
```

7 Indeksy

7.1 MenuIndex

Indeks wykorzystujący zakres dostępności menu.

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

7.2 CompanyCustomersIndex

Indeks wykorzystujący nazwę firmy.

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

7.3 PrivateCustomersIndex

Indeks wykorzystujący imię i nazwisko klienta.

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

8 Widoki

8.1 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
```

8.2 ReservationsToAccept

Pokazuje rezerwacje, które nie zostały zaakceptowane.

```
CREATE OR ALTER VIEW ReservationsToAccept
AS
    SELECT *
    FROM
        ReservationsDetails
    WHERE
        [Status] = 'do akceptacji'
GO
```

8.3 TodayReservations

Pokazuje rezerwacje na aktualny dzień.

```
CREATE OR ALTER VIEW TodayReservations
AS
    SELECT *
    FROM
        ReservationsDetails
    WHERE
        DATEDIFF(day, StartDate, GETDATE()) = 0
GO
```

8.4 ReservationsDetails

Pokazuje szczegóły wszystkich rezerwacji.

```
CREATE OR ALTER VIEW ReservationsDetails
AS
    SELECT
        ReservationID,
        CustomerID,
        (SELECT c.FullName FROM CustomersFullNames c WHERE c.CustomerID =
        ↪  r.CustomerID) FullName,
        StartDate,
        EndDate,
        Guests,
        (SELECT count(1) FROM TableDetails td WHERE td.ReservationID =
        ↪  r.ReservationID) TablesNum,
        (CASE
            WHEN (Accepted = 0 AND Canceled = 0) THEN 'do akceptacji'
            WHEN Canceled = 1 THEN 'anulowana'
            WHEN (Canceled = 0 AND EndDate < GETDATE()) THEN 'zakończona'
            WHEN (Accepted = 1 AND GETDATE() < StartDate) THEN 'zaakceptowana'
            ELSE 'trwajaca'
        END
        ) Status
    FROM
        Reservations r
GO
```

8.5 CurrentTables

Podgląd aktualnego stanu stolików.

```
CREATE OR ALTER VIEW CurrentTables
AS
    SELECT
        TableID,
        Seats,
        (CASE
            WHEN dbo.TableAvailableAtTime(TableID, GETDATE(), GETDATE()) = 1 THEN
            ↪  'dostepny'
            ELSE 'zajety'
        END
        ) Status,
        dbo.CurrentTableReservation(TableID) ReservationID,
```

```
        dbo.EndOfTableOccupationTime(TableID) EndOfReservation
    FROM Tables
GO
```

8.6 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
```

8.7 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
```

8.8 CurrentWeekSeaFoodOrders

Pokazuje zamówienia z owocami morza do realizacji w aktualnym tygodniu

```
CREATE OR ALTER VIEW CurrentWeekSeaFoodOrders
AS SELECT
        o.CustomerID,
        o.OrderID,
        o.CompletionDate,
```

```
        dbo.TotalOrderAmount(o.OrderID) TotalAmount
    FROM
        Orders o
        INNER JOIN OrderDetails od ON od.OrderID = o.OrderID
        INNER JOIN Meals m ON m.MealID = od.MealID
    WHERE
        m.SeaFood = 1
        AND o.CompletionDate > GETDATE()
        AND DATEDIFF(day, o.CompletionDate, GETDATE()) < 7
        AND Canceled = 0
GO
```

8.9 MenusInProgress

Pokazuje nieaktywne menu.

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

8.10 CurrentMenu

Pokazuje aktualne menu dla zamówień na ten sam dzień

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

8.11 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) FullName,
        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
```

9 Procedury

9.1 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
```

9.2 AddTable(Seats)

Dodaje nowy stolik.

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

9.3 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
```

9.4 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
```

9.5 AddReservation(...)

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
    SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
    BEGIN TRANSACTION

        IF dbo.AreTablesAvailable(@StartDate, @EndDate, @Tables) = 0 BEGIN
            ;THROW 52000, 'At least one of the tables is not available', 1
            RETURN
        END

        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

    COMMIT
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK;
        THROW;
    END CATCH
END
GO
```

9.6 AddInstantReservation(CustomerID, EndDate, Tables, ReservationID OUTPUT)

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

```sql
CREATE OR ALTER PROCEDURE AddInstantReservation (
    @CustomerID int,
    @EndDate datetime,
    @Tables ReservationTablesListT READONLY,
    @ReservationID int = NULL OUTPUT
)
AS BEGIN

    DECLARE @StartDate datetime = GETDATE();

    EXEC AddReservation
        @CustomerID = @CustomerID,
        @StartDate = @StartDate,
        @EndDate = @EndDate,
```

```
        @Tables = @Tables,
        @ReservationID = @ReservationID OUTPUT;
END
GO
```

9.7 PrivateOnlineReservation(...)

Tworzy rezerwację online dla klienta indywidualnego wraz ze złożeniem zamówienia. Sprawdzane jest czy klient ma możliwość złożenia rezerwacji online.

```
CREATE OR ALTER PROCEDURE PrivateOnlineReservation (
    @OrderDate datetime = NULL,
    @StartDate datetime,
    @EndDate datetime,
    @CustomerID int,
    @Guests nvarchar(max) = NULL,
    @Tables ReservationTablesListT READONLY,
    @OrderedItems OrderedItemsListT READONLY
)
AS BEGIN
    BEGIN TRY
    SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
    BEGIN TRANSACTION

        IF NOT EXISTS (SELECT * FROM PrivateCustomers WHERE CustomerID = @CustomerID)
        ↪ BEGIN
            ;THROW 52000, 'It is not a private customer', 1
            RETURN
        END

        IF 0 = dbo.CanReserveOnline(@CustomerID, @StartDate, @OrderedItems) BEGIN
            ;THROW 52000, 'The customer is not allowed to make an online reservation',
            ↪ 1
            RETURN
        END

        DECLARE @ReservationID int;
        EXEC AddReservation
            @StartDate = @StartDate,
            @EndDate = @EndDate,
            @Accepted = 0,
            @CustomerID = @CustomerID,
            @Tables = @Tables,
            @ReservationID = @ReservationID OUTPUT;

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

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


```

```
        UPDATE Orders
        SET ReservationID = @ReservationID
        WHERE OrderID = @OrderID

    COMMIT
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK;
        THROW;
    END CATCH
END
GO
```

9.8 CompanyOnlineReservation(...)

Tworzy rezerwację online dla klienta firmowego. Rezerwacja może być połączona ze złożeniem zamówienia.

```
CREATE OR ALTER PROCEDURE CompanyOnlineReservation  (
    @OrderDate datetime = NULL,
    @StartDate datetime,
    @EndDate datetime,
    @CustomerID int,
    @Guests nvarchar(max) = NULL,
    @Tables ReservationTablesListT READONLY,
    @OrderedItems OrderedItemsListT READONLY
)
AS BEGIN
    BEGIN TRY
    SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
    BEGIN TRANSACTION

        IF NOT EXISTS (SELECT * FROM CompanyCustomers WHERE CustomerID = @CustomerID)
        ↪ BEGIN
            ;THROW 52000, 'It is not a company customer', 1
            RETURN
        END

        DECLARE @ReservationID int;
        EXEC AddReservation
            @StartDate = @StartDate,
            @EndDate = @EndDate,
            @Accepted = 0,
            @CustomerID = @CustomerID,
            @Tables = @Tables,
            @ReservationID = @ReservationID OUTPUT;

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

        DECLARE @OrderID int;
        EXEC CreateOrder
            @CustomerID = @CustomerID,
            @OrderDate = @OrderDate,
            @CompletionDate = @StartDate,
            @OrderedItems = @OrderedItems,
```

```
            @OrderID = @OrderID OUTPUT;


        UPDATE Orders
        SET ReservationID = @ReservationID
        WHERE OrderID = @OrderID

    COMMIT
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK;
        THROW;
    END CATCH
END
GO
```

9.9 AcceptReservation(ReservationID)

Akceptuje rezerwację złożoną przez formularz internetowy

```
CREATE OR ALTER PROCEDURE AcceptReservation (@ReservationID int)
AS BEGIN
    BEGIN TRY
    SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
    BEGIN TRANSACTION

        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

    COMMIT
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK;
        THROW;
    END CATCH
END
GO
```

9.10 CancelReservation(ReservationID)

Anuluje wybraną rezerwację

```
CREATE OR ALTER PROCEDURE CancelReservation (@ReservationID int)
AS BEGIN
    BEGIN TRY
    SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
    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 canceled', 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
        IF @@TRANCOUNT > 0
            ROLLBACK;
        THROW;
    END CATCH
END
GO
```

9.11 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

    UPDATE Reservations SET EndDate = GETDATE()
    WHERE ReservationID = @ReservationID
END
GO
```

9.12 ExtendCurrrentReservation(ReservationID, NewEndDate)

Wydłużenie czasu rezerwacji do preferowanej godziny jeśli to możliwe

```
CREATE OR ALTER PROCEDURE ExtendCurrentReservation(@ReservationID int, @NewEndDate
↪  datetime)
AS BEGIN

    IF @NewEndDate < (SELECT EndDate FROM Reservations WHERE ReservationID =
    ↪  @ReservationID) BEGIN
        ;THROW 52000, 'New end time is before the current one', 1
        RETURN
    END

    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


    IF EXISTS (
        SELECT *
        FROM
            TableDetails td
        WHERE
            td.ReservationID = @ReservationID
            AND td.TableID IN  (
                SELECT
                    TD.TableID
                FROM
                    TableDetails TD
                    INNER JOIN Reservations R ON TD.ReservationID = R.ReservationID
                WHERE
                    R.StartDate >= (SELECT EndDate FROM Reservations WHERE
                    ↪  ReservationID = @ReservationID)
                    AND R.StartDate < @NewEndDate
            )
    )
    BEGIN
        ;THROW 52000, 'Extension is not possible for this amount of time', 1
        RETURN
    END

    UPDATE Reservations SET EndDate = @NewEndDate
    WHERE ReservationID = @ReservationID
END
GO
```

9.13 CreateOrderInvoice(OrderID)

Generuje fakturę w tabeli Invoices dla danego zamówienia.

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

    DECLARE @CustomerID int = (SELECT CustomerID FROM Orders WHERE OrderID = @OrderID)

    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 52000, 'The customer has not provided indispensable data to create an
        ↪  invoice', 1
        RETURN
    END

    IF (SELECT InvoiceID FROM Orders WHERE OrderID = @OrderID) IS NOT NULL
    BEGIN
        ;THROW 52000, 'Order already has an invoice', 1
        RETURN
    END

    IF (SELECT Paid FROM Orders WHERE OrderID = @OrderID) = 0
    BEGIN
        ;THROW 52000, 'Order has not been paid yet', 1
        RETURN
    END

    BEGIN TRY;
    BEGIN TRANSACTION;

        DECLARE  @InvoiceID varchar(16) = dbo.CreateInvoiceID()

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


        UPDATE Orders SET InvoiceID = @InvoiceID
```

```
50          WHERE OrderID = @OrderID
51
52      COMMIT;
53      END TRY
54      BEGIN CATCH;
55          ROLLBACK;
56          THROW;
57      END CATCH
58  END
59  GO
```

9.14 CreateMonthlyInvoice(CustomerID, Month, Year)

Generuje fakturę dla danego klienta, dla danego miesiąca.

```
1   CREATE OR ALTER PROCEDURE CreateMonthlyInvoice(@CustomerID Int, @Month int, @Year int)
2   AS BEGIN
3
4       IF NOT EXISTS (SELECT * FROM Customers WHERE CustomerID = @CustomerID) BEGIN
5           ;THROW 52000, 'The customer does not exist', 1
6           RETURN
7       END
8
9       IF 0 = dbo.CanCreateInvoice(@CustomerID)
10      BEGIN
11          ;THROW 52000, 'The customer has not provided indispensable data to create an
            ↪  invoice', 1
12          RETURN
13      END
14
15      IF DATEFROMPARTS(@Year, @Month, DAY(EOMONTH(DATEFROMPARTS(@Year, @Month, 1)))) >=
        ↪  GETDATE()
16      BEGIN
17          ;THROW 52000, 'The month has not passed yet', 1
18          RETURN
19      END
20
21      BEGIN TRY;
22      BEGIN TRANSACTION;
23
24          DECLARE  @InvoiceID varchar(16) = dbo.CreateInvoiceID()
25
26          INSERT INTO Invoices(
27               InvoiceID, Date, CustomerID, TotalAmount, FirstName, LastName,
                 ↪  CompanyName, Email, Phone, Address, City, PostalCode, Country
28          )
29          SELECT
30              @InvoiceID, GETDATE(), Customers.CustomerID,
                ↪  SUM(dbo.TotalOrderAmount(Orders.OrderID)), MAX(FirstName),
                ↪  MAX(LastName),
31                  MAX(CompanyName), MAX(Email), MAX(Phone), MAX(Address), MAX(City),
                    ↪  MAX(PostalCode), MAX(Country)
32          FROM
33              Customers
34              LEFT JOIN Orders ON Orders.CustomerID = Customers.CustomerID AND
                ↪  Orders.InvoiceID IS NULL
35                                  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 = @InvoiceID
        WHERE
            Orders.CustomerID = @CustomerID
            AND Orders.InvoiceID IS NULL
            AND MONTH(Orders.CompletionDate) = @Month
            AND YEAR(Orders.CompletionDate) = @Year

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

9.15 CreateOrder(...)

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

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

```
25          END
26
27          DECLARE @MenuID int = dbo.GetMenuIDForDay(@CompletionDate)
28          IF @MenuID IS NULL
29          BEGIN
30              ;THROW 52000, 'The menu does not exist', 1
31              RETURN
32          END
33
34          -- check if all items belong to the proper menu
35          IF (SELECT count(1) FROM @OrderedItems) !=  (SELECT count(1) FROM MenuItems
            ↪  WHERE MenuID = @MenuID AND MealID IN (SELECT MealID FROM @OrderedItems))
            ↪  BEGIN
36              ;THROW 52000, 'The ordered items list is incorrect', 1
37              RETURN
38          END
39
40          -- check if order incluing seafood is placed in enough advance
41          IF EXISTS (SELECT * FROM @OrderedItems oi INNER JOIN Meals m ON m.MealID =
            ↪  oi.MealID WHERE m.SeaFood = 1)
42          AND 0 = dbo.CanOrderSeafood(@OrderDate, @CompletionDate) BEGIN
43              ;THROW 52000, 'The order including seafood must be placed in advance', 1
44          END
45
46
47          INSERT INTO Orders(CustomerID, OrderDate, CompletionDate, Paid, Canceled,
            ↪  Completed)
48          VALUES (@CustomerID, @OrderDate, @CompletionDate, 0, 0, 0)
49
50          SET @OrderID = @@IDENTITY
51
52          INSERT INTO OrderDetails(OrderID, Quantity, MealID, MenuID)
53          SELECT @OrderID, Quantity, MealID, @MenuID FROM @OrderedItems
54
55          COMMIT;
56      END TRY
57      BEGIN CATCH
58          IF @@TRANCOUNT > 0
59              ROLLBACK;
60          THROW;
61      END CATCH
62  END
63  GO
```

9.16 CreateInstantOrder(CustomerID, CompletionDate, OrderedItems, OrderID OUTPUT)

Tworzy nowe zamówienie w systemie i ustawia je jako zrealizowane oraz opłacone. Funkcja jes wykorzystywana, gdy klient kupuje towar na miejscu.

```
1   CREATE OR ALTER PROCEDURE CreateInstantOrder(
2       @CustomerID int,
3       @CompletionDate datetime,
4       @OrderedItems OrderedItemsListT READONLY,
5       @OrderID int = NULL OUTPUT
6   )
7   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
        IF @@TRANCOUNT > 0
            ROLLBACK;
        THROW;
    END CATCH
END
GO
```

9.17 CancelOrder(OrderID)

Anuluje zamówienie, które nie zostało jeszcze zrealizowane.

```
CREATE OR ALTER PROCEDURE CancelOrder (@OrderID int)
AS BEGIN
    BEGIN TRY
    SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
    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 has a reservation
        IF NULL != (SELECT ReservationID FROM Orders WHERE OrderID = @OrderID) BEGIN
            ;THROW 52000, 'The order cannot be canceled because it has a reservation',
            ↪  1
            RETURN
        END

        -- check if the order was completed
        IF 1 = (SELECT Completed FROM Orders WHERE OrderID = @OrderID) BEGIN
            ;THROW 52000, 'The order has been already completed', 1
            RETURN;
        END

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

```
30
31          -- set order as canceled
32          UPDATE Orders SET Canceled = 1 WHERE OrderID = @OrderID
33
34      COMMIT
35      END TRY
36      BEGIN CATCH
37          IF @@TRANCOUNT > 0
38              ROLLBACK;
39          THROW;
40      END CATCH
41  END
42  GO
```

9.18 PayForOrder(OrderID)

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       SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
5       BEGIN TRANSACTION
6
7           -- check if the order exists
8           IF NOT EXISTS (SELECT * FROM Orders WHERE OrderID = @OrderID) BEGIN
9               ;THROW 52000, 'The order does not exist', 1
10              RETURN
11          END
12
13          -- check if the order was paid
14          IF 1 = (SELECT Paid FROM Orders WHERE OrderID = @OrderID) BEGIN
15              ;THROW 52000, 'The order has been already paid', 1
16              RETURN;
17          END
18
19          -- check if the order was canceled
20          IF 1 = (SELECT Canceled FROM Orders WHERE OrderID = @OrderID) BEGIN
21              ;THROW 52000, 'The order was canceled', 1
22              RETURN;
23          END
24
25          -- update status
26          UPDATE Orders SET Paid = 1 WHERE OrderID = @OrderID
27
28          DECLARE @CustomerID int = (SELECT CustomerID FROM Orders WHERE OrderID =
            ↪  @OrderID)
29          DECLARE @CompletionDate datetime;
30
31          SELECT
32              @CustomerID = CustomerID,
33              @CompletionDate = CompletionDate
34          FROM Orders
35          WHERE OrderID = @OrderID
36
37          -- give discount type 1
38          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
        IF @@TRANCOUNT > 0
            ROLLBACK;
        THROW;
    END CATCH
END
GO
```

9.19 CompleteOrder(OrderID, CompletionDate)

Zapisuje informację, że zamówienie zostało wydane klientowi.

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

        -- check if the order was completed
        IF 1 = (SELECT Completed FROM Orders WHERE OrderID = @OrderID) BEGIN
            ;THROW 52000, 'The order has been already completed', 1
            RETURN;
        END

        UPDATE Orders
        SET Completed = 1,
            CompletionDate = ISNULL(@CompletionDate, GETDATE())
        WHERE OrderID = @OrderID

    COMMIT
    END TRY
    BEGIN CATCH
```

```
        IF @@TRANCOUNT > 0
            ROLLBACK;
        THROW;
    END CATCH
END
GO
```

9.20 CreateNewMeal(Name, IsSeaFood, DefaultPrice, Active, MealID OUTPUT)

Tworzy nowy posiłek.

```
CREATE OR ALTER PROCEDURE CreateNewMeal(@Name nvarchar(64), @IsSeaFood bit,
↪  @DefaultPrice money, @Active bit = 1, @MealID int OUTPUT)
AS BEGIN
    INSERT INTO Meals([Name], SeaFood, DefaultPrice, Active)
    VALUES(@Name, @IsSeaFood, @DefaultPrice, @Active)

    SET @MealID = @@IDENTITY
END
GO
```

9.21 SetMealActive(MealID, Active)

Aktywuje lub dezaktywuje posiłek.

```
CREATE OR ALTER PROCEDURE SetMealActive(@MealID int, @Active bit)
AS BEGIN
    UPDATE Meals
    SET Active = @Active
    WHERE MealID = @MealID
END
GO
```

9.22 UpdateMealDefaultPrice(MealID, DefaultPrice)

Zmienia domyślną cenę posiłku.

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

9.23 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)
```

```
5
6       SET @MenuID = @@IDENTITY
7   END
8   GO
```

9.24 ChangeMenuDates(MenuID, StartDate, EndDate)

Zmienia datę obowiązywania nieaktywnego menu

```
1   CREATE OR ALTER PROCEDURE ChangeMenuDates(@MenuID int, @StartDate datetime = NULL,
    ↪  @EndDate datetime = NULL)
2   AS BEGIN
3       IF (SELECT Active FROM Menu WHERE MenuID = @MenuID) = 1 BEGIN
4           ;THROW 52000, 'Menu is active', 1
5           RETURN
6       END
7
8       DECLARE @PrevStartDate datetime
9       DECLARE @PrevEndDate datetime
10
11      SELECT @PrevStartDate = StartDate, @PrevEndDate = EndDate
12      FROM Menu WHERE MenuID = @MenuID
13
14      UPDATE Menu
15      SET StartDate = ISNULL(@StartDate, @PrevStartDate),
16          EndDate = ISNULL(@EndDate, @PrevEndDate)
17      WHERE MenuID = @MenuID
18  END
19  GO
```

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

9.26 RemoveMenuItem(MenuID, MealID)

Usuwa posiłek z nieaktynego menu.

```sql
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
```

9.27 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
```

9.28 DeactivateMenu(MenuID)

Dezaktywuje menu.

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

9.29 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;
        IF @@TRANCOUNT > 0
            ROLLBACK;
        THROW;
    END CATCH
END
GO
```

9.30 AddPrivateCustomer(...)

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

```sql
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;
        IF @@TRANCOUNT > 0
            ROLLBACK;
        THROW;
    END CATCH
END
GO
```

9.31 UpdateCustomer(...)

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

```sql
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);
```

```
25          DECLARE @PREV_Country nvarchar(64);
26
27          -- get previous values from Customers
28          SELECT
29              @PREV_Email = Email,
30              @PREV_Phone = Phone,
31              @PREV_Address = Address,
32              @PREV_City = City,
33              @PREV_PostalCode = PostalCode,
34              @PREV_Country = Country
35          FROM Customers
36          WHERE CustomerID = @CustomerID
37
38          -- set new values in Customers
39          UPDATE Customers
40          SET Email = ISNULL(@Email, @PREV_Email),
41              Phone = ISNULL(@Phone, @PREV_Phone),
42              Address = ISNULL(@Address, @PREV_Address),
43              City = ISNULL(@City, @PREV_City),
44              PostalCode = ISNULL(@PostalCode, @PREV_PostalCode),
45              Country = ISNULL(@Country, @PREV_Country)
46          WHERE CustomerID = @CustomerID
47
48          COMMIT;
49      END TRY
50      BEGIN CATCH;
51          IF @@TRANCOUNT > 0
52              ROLLBACK;
53          THROW;
54      END CATCH
55  END
56  GO
```

9.32 UpdateCompanyCustomer(...)

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

```
1   CREATE OR ALTER PROCEDURE UpdateCompanyCustomer(
2       @CustomerID int,
3
4       @Email nvarchar(64) = NULL,
5       @Phone nvarchar(16) = NULL,
6       @Address nvarchar(64) = NULL,
7       @City nvarchar(64) = NULL,
8       @PostalCode varchar(16) = NULL,
9       @Country nvarchar(64) = NULL,
10      @CompanyName nvarchar(64) = NULL,
11      @NIP varchar(16) = NULL
12  )
13  AS BEGIN
14      BEGIN TRY;
15      BEGIN TRANSACTION;
16
17          IF NOT EXISTS (SELECT * FROM Customers WHERE CustomerID = @CustomerID) BEGIN
18              ;THROW 52000, 'The customer does not exist', 1
19              RETURN
20          END
```

```
21
22          -- update values in Customers (common part)
23          EXEC UpdateCustomer
24              @CustomerID = @CustomerID,
25              @Email = @Email,
26              @Phone = @Phone,
27              @Address = @Address,
28              @City = @City,
29              @PostalCode = @PostalCode,
30              @Country = @Country;
31
32          -- update values in CompanyCustomers
33          DECLARE @PREV_CompanyName nvarchar(64);
34          DECLARE @PREV_NIP varchar(16);
35
36          SELECT
37              @PREV_CompanyName = CompanyName,
38              @PREV_NIP = NIP
39          FROM CompanyCustomers
40          WHERE CustomerID = @CustomerID
41
42          UPDATE CompanyCustomers
43          SET CompanyName = ISNULL(@CompanyName, @PREV_CompanyName),
44              NIP = ISNULL(@NIP, @PREV_NIP)
45          WHERE CustomerID = @CustomerID
46
47      COMMIT;
48      END TRY
49      BEGIN CATCH;
50          IF @@TRANCOUNT > 0
51              ROLLBACK;
52          THROW;
53      END CATCH
54  END
55  GO
```

9.33 UpdatePrivateCustomer(...)

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

```
1   CREATE OR ALTER PROCEDURE UpdatePrivateCustomer(
2       @CustomerID int,
3
4       @Email nvarchar(64) = NULL,
5       @Phone nvarchar(16) = NULL,
6       @Address nvarchar(64) = NULL,
7       @City nvarchar(64) = NULL,
8       @PostalCode varchar(16) = NULL,
9       @Country nvarchar(64) = NULL,
10      @FirstName nvarchar(64) = NULL,
11      @LastName nvarchar(64) = NULL
12  )
13  AS BEGIN
14      BEGIN TRY;
15      BEGIN TRANSACTION;
16
17          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);

        SELECT
            @PREV_FirstName = FirstName,
            @PREV_LastName = LastName
        FROM PrivateCustomers
        WHERE CustomerID = @CustomerID

        UPDATE PrivateCustomers
        SET FirstName = ISNULL(@FirstName, @PREV_FirstName),
            LastName = ISNULL(@LastName, @PREV_LastName)
        WHERE CustomerID = @CustomerID

    COMMIT;
    END TRY
    BEGIN CATCH;
        IF @@TRANCOUNT > 0
            ROLLBACK;
        THROW;
    END CATCH
END
GO
```

9.34 ForgetCustomer(...)

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

```
CREATE OR ALTER PROCEDURE ForgetCustomer(@CustomerID INT)
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 Customers
    SET
        Email = NULL,
        Phone = NULL,
```

```
        Address = NULL,
        City = NULL,
        PostalCode = NULL,
        Country = NULL
    WHERE CustomerID = @CustomerID

    UPDATE PrivateCustomers
    SET
        FirstName = NULL,
        LastName = NULL
    WHERE CustomerID = @CustomerID

    UPDATE CompanyCustomers
    SET
        CompanyName = NULL,
        NIP = NULL
    WHERE CustomerID = @CustomerID

    COMMIT;
    END TRY
    BEGIN CATCH;
        IF @@TRANCOUNT > 0
            ROLLBACK;
        THROW;
    END CATCH
END
GO
```

10 Funkcje

10.1 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
    RETURN CASE
        WHEN EXISTS (SELECT * FROM @Tables WHERE dbo.TableAvailableAtTime(TableID,
        ↪  @StartDate, @EndDate) = 0)
            THEN 0
        ELSE 1
    END
END
GO
```

10.2 SingleReservationDetails(ReservationID)

Zwraca szczegóły na temat danej rezerwacji.

```
CREATE OR ALTER FUNCTION SingleReservationDetails(@ReservationID int)
RETURNS TABLE
AS RETURN
    SELECT *
    FROM
```

```
        ReservationsDetails
    WHERE
        ReservationID = @ReservationID
GO
```

10.3 GetCustomersReservations(CustomerID)

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

```
CREATE OR ALTER FUNCTION GetCustomersReservations(@CustomerID int)
RETURNS TABLE
AS RETURN
    SELECT *
    FROM
        ReservationsDetails
    WHERE
        CustomerID = @CustomerID
GO
```

10.4 CanReserveOnline(CustomerID, CompletionDate, OrderedItems)

Zwraca informację czy dany klient indywidualny spełnia warunki do złożenia rezerwacji online.

```
CREATE OR ALTER FUNCTION CanReserveOnline(@CustomerID int, @CompletionDate datetime,
↪  @OrderedItems OrderedItemsListT READONLY)
RETURNS bit
BEGIN

    DECLARE @MenuID int = dbo.GetMenuIDForDay(@CompletionDate)

    DECLARE @MinAmount money = (SELECT WZ FROM CurrentConstants)
    DECLARE @MinOrdersNum int = (SELECT WK FROM CurrentConstants)

    DECLARE @TotalAmount money = (
        SELECT
            sum(oi.Quantity * mi.Price)
        FROM
            @OrderedItems oi
            INNER JOIN MenuItems mi ON mi.MealID = oi.MealID AND mi.MenuID = @MenuID
        )

    DECLARE @OrdersNum int = (
        SELECT
            count(1)
        FROM
            Orders
        WHERE
            CustomerID = @CustomerID
            AND Completed = 1
        )

    RETURN CASE WHEN @TotalAmount >= @MinAmount AND @OrdersNum >= @MinOrdersNum THEN 1
    ↪  ELSE 0 END
END
GO
```

10.5 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
    RETURN CASE
        WHEN @TableID NOT IN
            (SELECT
                TableID
            FROM
                TableDetails TD
                INNER JOIN Reservations R ON TD.ReservationID = R.ReservationID
            WHERE
                NOT EndDate <= @StartDate
                AND NOT StartDate >= @EndDate
                AND Canceled = 0
            )
        THEN 1
        ELSE 0
    END
END
GO
```

10.6 EndOfTableOccupationTime(TableID)

Zwraca czas zakończenia rezerwacji jeśli stolik jest aktualnie zarezerwowany.

```
CREATE OR ALTER FUNCTION EndOfTableOccupationTime(@TableID int)
RETURNS datetime
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
GO
```

10.7 CurrentTableReservation(TableID)

Zwraca numer rezerwacji jeśli stolik jest aktualnie zarezerwowany.

```
CREATE OR ALTER FUNCTION CurrentTableReservation(@TableID int)
RETURNS int
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
GO
```

10.8 TablesAvailableToReserve(StartDate, EndDate)

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

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

10.9 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.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
```

10.10 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
```

10.11 OrdersStatistics(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 OrdersStatistics(
    @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
```

10.12 TablesStatistics(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 TablesStatistics (
    @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
```

10.13 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
```

10.14 CanCreateInvoice(CustomerID)

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 IS NOT NULL AND pc.LastName IS NOT NULL) OR cc.CompanyName
            ↪  IS NOT NULL) AND
            c.Address IS NOT NULL AND
            c.City IS NOT NULL AND
            c.PostalCode IS NOT NULL AND
            c.Country IS NOT NULL AND
            c.CustomerID = @CustomerID
        ) THEN 1 ELSE 0 END
END
GO
```

10.15 CountInvoicesForDay(Day)

Zwraca liczbę faktur wystawionych danego dnia.

```sql
CREATE OR ALTER FUNCTION CountInvoicesForDay(@Day datetime) RETURNS int
BEGIN
RETURN (
    SELECT
        count(1)
    FROM
        Invoices
    WHERE
        DATEDIFF(day, @Day, Date) = 0
    )
END
GO
```

10.16 CreateInvoiceID

Zwraca proponowany numer faktury na obecny dzień. Numer faktury jest połączeniem daty wystawienia i numeru kolejnej faktury z tego dnia.

```sql
CREATE OR ALTER FUNCTION CreateInvoiceID() RETURNS varchar(16)
BEGIN
RETURN CONCAT_WS('/',
                DATEPART(year, GETDATE()),
                DATEPART(month, GETDATE()),
                DATEPART(day, GETDATE()),
                dbo.CountInvoicesForDay(GETDATE())
                )
END
GO
```

10.17 TotalDiscountForOrder(OrderID)

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
```

10.18 TotalOrderAmount(OrderID)

Zwraca całkowitą cenę zamówienia biorąc pod uwagę rabaty.

```sql
CREATE OR ALTER FUNCTION TotalOrderAmount(@OrderID int) RETURNS money
BEGIN
    DECLARE @Amount money = (
        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
    )

    RETURN CASE WHEN @Amount IS NOT NULL THEN @Amount ELSE 0 END
END
GO
```

10.19 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
```

10.20 IsDiscountType1(CustomerID, CheckDate)

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
```

10.21 IsDiscountType2(CustomerID, CheckDate)

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

```
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(OrderID))
        FROM
            Orders
        WHERE
            CustomerID = @CustomerID
            AND DATEDIFF(DAY, OrderDate, GETDATE()) <= @LastDays
            AND Completed = 1
            AND CompletionDate < @CheckDate
    )

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

10.22 CustomerOrders(CustomerID)

Pokazuje wszystkie zamówienia danego klienta

```
CREATE OR ALTER FUNCTION CustomerOrders(@CustomerID int)
RETURNS TABLE
AS RETURN
    SELECT
        ReservationID,
        InvoiceID,
        OrderDate,
        CompletionDate,
        Status,
        TotalAmount
    FROM
        CalculatedOrders
    WHERE
        CustomerID = @CustomerID
GO
```

10.23 GetOrderDetails(OrderID)

Pokazuje szczegóły konkretnego zamówienia.

```
CREATE OR ALTER FUNCTION GetOrderDetails(@OrderID int)
RETURNS TABLE
AS RETURN
    SELECT
        m.MealID,
        m.Name,
        mi.Price,
        od.Quantity,
        (od.Quantity * Price) TotalPrice
    FROM
        OrderDetails od
        INNER JOIN Meals m ON m.MealID = od.MealID
        INNER JOIN MenuItems mi ON mi.MealID = od.MealID AND mi.MenuID = od.MenuID
    WHERE
        od.OrderID = @OrderID
GO
```

10.24 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
```

10.25 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,
            Customers.Phone,
            Customers.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
```

10.26 GetMenuForDay(Date)

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,
```

```
16              m.Name,
17              (CASE WHEN m.SeaFood = 1 THEN 'TAK' ELSE 'NIE' END) SeaFood,
18              Price
19          FROM Meals m
20              INNER JOIN MenuItems mi ON mi.MealID = m.MealID
21          WHERE
22              mi.MenuID = @MenuID
23              AND (m.SeaFood = 0 OR @ShowSeaFood = 1)
24      RETURN
25  END
26  GO
```