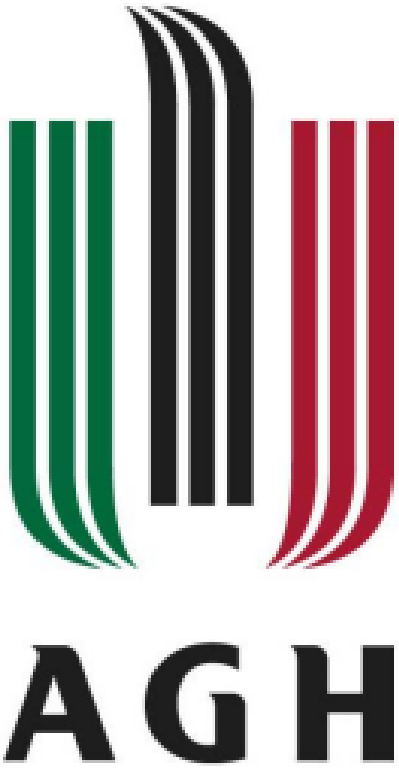


AGH

Podstawy baz danych
Dokumentacja projektu

Hubert Kabziński, Mateusz Waga, Jakub Woś

26 marca 2023

Spis treści

Rozdział 1

Użytkownicy

- Klient zarejestrowany
- Klient niezarejestrowany
- Klient firma
- Pracownik
- Administrator
- Właściciel
- System

Rozdział 2

Funkcje użytkowników

2.1 Pracownik

- modyfikacja menu (poprzez dodanie nowego menu lub zmianę już obecnych w bazie)
- dostęp do menu danego okresu
- zmienianie ilości produktu na stanie
- dodanie, usunięcie, modyfikacja zamówienia
- pokazanie zamówień na wynos które nie zostały jeszcze wykonane
- zobaczenie historii zamówień danego klienta
- wystawienie faktury (dla danego zamówienia lub zbiorczej)
- akceptacja rezerwacji przez pracownika (poprzez ustawienie flagi Accepted w tabeli reservations)

2.2 Klient

- zarejestrowany:
  - złożenie rezerwacji
  - opłacenie zamówienia
- niezarejestrowany:
  - złożenie zamówienia (przez internet)
- firma:
  - złożenie zamówienia
  - rezerwacji stolików na firmę
  - rezerwację stolików dla konkretnych pracowników firmy (imiennie)

2.3 System

- generowanie raportów dotyczących zamówień oraz rabatów dla klienta indywidualnego oraz firm.
- generowanie faktur
- przypominanie o zmianie menu (poprzez analizę pod kątem zmian połowy produktów w menu które pojawiły się w przeciągu ostatnich)
- w śr/czw/pt dodanie do menu owoców morza
- przyznawanie rabatów satysfakcji warunków przez danego klienta

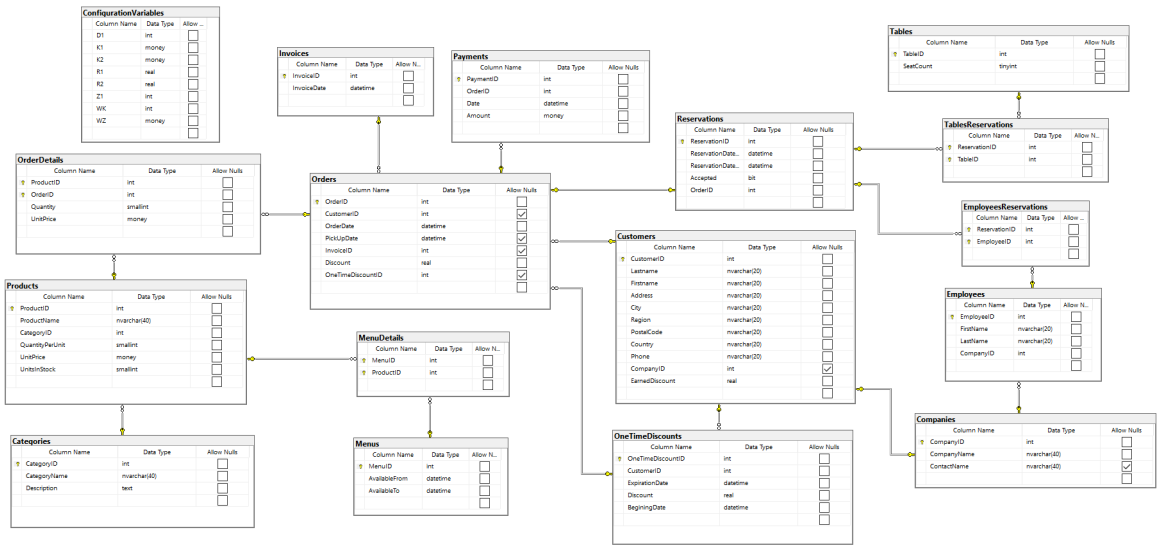
- generowanie raportów miesięcznych i tygodniowych, dotyczących rezerwacji stolików, rabatów, menu, a także statystyk zamówienia dla klientów indywidualnych oraz firm dotyczących kwot oraz czasu składania zamówień.

2.4 Właściciel

2.5 Administrator

Rozdział 3

Diagram Bazy Danych



Rozdział 4

Tabele

4.1 Tabela Categories

Klucz główny: CategoryID
ID kategorii: CategoryID
Nazwa kategorii: CategoryName
Opis kategorii: Description

```
CREATE TABLE [dbo].[Categories](
[CategoryID] [int] IDENTITY(1,1) NOT NULL,
[CategoryName] [nvarchar](40) NOT NULL,
[Description] [text] NOT NULL,
PRIMARY KEY CLUSTERED ([CategoryID] ASC))
```

4.2 Tabela Customers

Klucz główny: CustomerID
ID klienta: CustomerID
Nazwisko klienta: LastName
Imię klienta: FirstName
Adres zamieszkania: Address
Miejscowość zamieszkania: City
Województwo zamieszkania: Region
Kod pocztowy: PostalCode
Państwo zamieszkania: Country
Numer telefonu: Phone
Id firmy: CompanyID
Uzyskany rabat: EarnedDiscount

```
CREATE TABLE [dbo].[Customers](
[CustomerID] [int] IDENTITY(1,1) NOT NULL,
[Lastname] [nvarchar](20) NOT NULL,
[Firstname] [nvarchar](20) NOT NULL,
[Address] [nvarchar](20) NOT NULL,
[City] [nvarchar](20) NOT NULL,
[Region] [nvarchar](20) NOT NULL,
[PostalCode] [nvarchar](20) NOT NULL,
[Country] [nvarchar](20) NOT NULL,
[Phone] [nvarchar](20) NOT NULL,
[CompanyID] [int] NULL,
[EarnedDiscount] [real] NOT NULL,
CONSTRAINT [PK_Customers]
PRIMARY KEY CLUSTERED ([CustomerID] ASC))
```

Wartości domyślne:

EarnedDsicount przyjmuje domyslnie wartosć 0

```
ALTER table Customers
ADD CONSTRAINT DF_Customers_EarnedDiscount default 0 for EarnedDiscount
```

CompanyID przyjmuje domyslnie wartosć null

```
ALTER table Customers
ADD constraint DF_Customers_CompanyID default null for CompanyID
```

Warunki integralnościowe:

Uzyskana zniżka musi być >= 0

```
ALTER TABLE [dbo].[Customers]
ADD CONSTRAINT [CHK_EarnedDiscount] CHECK
(([EarnedDiscount]>=(0)))
```

Kolumna CompanyID musi mieć unikalne wartości

```
ALTER TABLE [dbo].[Customers]
ADD CONSTRAINT [UQ_CompanyID] UNIQUE ([CompanyID])
Kod pocztowy musi być w formacie XX-XXX gdzie X to liczba z przedziału [0, 9]
ALTER TABLE [dbo].[Customers]
ADD CONSTRAINT [CHK_Customers_PostalCode] CHECK
(([PostalCode] LIKE '[0-9][0-9]-[0-9][0-9][0-9]'))
```

Numer telefonu musi być w formacie XXXXXXXXX gdzie X to liczba z przedziału [0, 9]

```
ALTER TABLE [dbo].[Customers]
ADD CONSTRAINT [CHK_Customers_Phone] CHECK (([Phone]
LIKE '+[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'))
```

4.3 Tabela Menus

Klucz główny: MenuID
ID menu: MenuID
Menu dostępne od daty: AvailableFrom
Menu dostępne do daty: AvailableTo

```
CREATE TABLE [dbo].[Menus](
[MenuID] [int] IDENTITY(1,1) NOT NULL,
[AvailableFrom] [datetime] NOT NULL,
[AvailableTo] [datetime] NOT NULL,
CONSTRAINT [PK_Menus]
PRIMARY KEY CLUSTERED ([MenuID] ASC))
```

Warunki integralnościowe:

Data rozpoczęcia dostępności musi być większa lub równa od daty zakończenia dostępności

```
ALTER TABLE [dbo].[Menus]
ADD CONSTRAINT [CHK_Menus_Date] CHECK
(([AvailableFrom]=[AvailableTo]))
```

4.4 Tabela MenuDetails

Klucz główny: MenuID, ProductID
ID menu w którym jest dany produkt: MenuID
ID produktu do który należy do danego menu: ProductID

```
CREATE TABLE [dbo].[MenuDetails](
[MenuID] [int] NOT NULL,
[ProductID] [int] NOT NULL,
CONSTRAINT [PK_MenuDetails]
PRIMARY KEY CLUSTERED ([ProductID] ASC, [ProductID] ASC))
```

4.5 Tabela OneTimeDiscounts

Klucz główny: OneTimeDiscountID
ID jednorazowego rabatu: OneTimeDiscountID
ID klienta posiadającego rabat: CustomerID
Data rozpoczęcia rabatu: BeginingDate
Data wygaśnięcia rabatu: ExpirationDate
Wysokość rabatu: Discount

```
CREATE TABLE [dbo].[OneTimeDiscounts](
[OneTimeDiscountID] [int] IDENTITY(1,1) NOT NULL,
[CutomerID] [int] NOT NULL,
[Discount] [real] NOT NULL,
[BeginingDate] [datetime] NOT NULL,
[ExpirationDate] [datetime] NOT NULL,
CONSTRAINT [PK_OneTimeDiscounts]
PRIMARY KEY CLUSTERED ([OneTimeDiscountID] ASC))
```

Warunki integralnościowe:

Wysokość rabatu musi być z przedziału [0, 1]

```
ALTER TABLE [dbo].[OneTimeDiscounts]
ADD CONSTRAINT [CK_OneTimeDiscounts_Discount] CHECK
(([Discount]>=(0)) AND ([Discount]<=(1)))
```

Data rozpoczecia rabatu musi być wcześniej niż zakończenia

```
ALTER TABLE [dbo].[OneTimeDiscounts]
ADD CONSTRAINT [CK_OneTimeDiscounts_Date] CHECK
((BeginingDate < ExpirationDate))
```

4.6 Tabela OrderDetails

Klucz główny: ProductID, OrderID
ID kupionego produktu: ProductID
ID zamówienia: OrderID
Ilość kupionego produktu : Quantity
Jednostkowa cena sztuki produktu: UnitPrice

```
CREATE TABLE [dbo].[OrderDetails](
[ProductID] [int] NOT NULL,
[OrderID] [int] NOT NULL,
[Quantity] [smallint] NOT NULL,
[UnitPrice] [money] NOT NULL,
CONSTRAINT [PK_OrderDetails]
PRIMARY KEY CLUSTERED ([OrderID] ASC,[ProductID] ASC))
```

Warunki integralnościowe:

Ilość zakupionych sztuk produktu musi być >= 0

```
ALTER TABLE [dbo].[OrderDetails]
ADD CONSTRAINT [CHK_OrderDetails_Quantity] CHECK (([Quantity]>=(0)))
```

Cena jednostkowa za sztukę produktu musi być >= 0

```
ALTER TABLE [dbo].[OrderDetails]
ADD CONSTRAINT [CHK_OrderDetails_UnitPrice] CHECK
(([UnitPrice]>=(0)))
```

4.7 Tabela Orders

Klucz główny: OrderID
ID zamówienia: OrderID
ID klienta który złożył zamówienie : CustomerID
Data złożenia zamówienia: OrderDate
Data odbioru zamówienia: PickUpDate
ID rachunku: InvoiceID
Przyznany rabat na zamówienie: Discount
ID jednorazowego rabatu użytego na zamówienie: OneTimeDiscountID

```
CREATE TABLE [dbo].[Orders](
[OrderID] [int] IDENTITY(1,1) NOT NULL,
[CustomerID] [int] NOT NULL,
[OrderDate] [datetime] NOT NULL,
[PickUpDate] [datetime] NULL,
[InvoiceID] [int] NULL,
[Discount] [real] NOT NULL,
[OneTimeDiscountID] [int] NULL
CONSTRAINT [PK_Orders]
PRIMARY KEY CLUSTERED ([OrderID] ASC))
```

Wartości domyślne:

Domyślnie wysokość rabatu to 0

```
ALTER TABLE Orders
ADD DEFAULT (0) FOR Discount
```

Domyślnie ID jednorazowej zniżki ma wartość null

```
ALTER TABLE Orders
ADD DEFAULT (null) FOR OneTimeDiscountID
```

Warunki integralnościowe:

Data odbioru musi być >= od daty złożenia zamówienia

```
ALTER TABLE [dbo].[Orders]
ADD CONSTRAINT [CHK_Orders_Dates] CHECK (([OrderDate]=[PickUpDate]))
```

Uzyskana zniżka musi być z przedziału [0, 1]

```
ALTER TABLE [dbo].[Orders]
ADD CONSTRAINT [CHK_Discount] CHECK (([Discount]>=(0)) AND ([Discount]<=(1)))
```

4.8 Tabela Products

Klucz główny: ProductID
ID produktu: ProductID
Nazwa produktu: ProductName
ID kategorii do której należy produkt: CategoryID
Ilość sztuk produktu w opakowaniu: QuantityPerUnit
Koszt jednej sztuki produktu: UnitPrice
Ilość sztuk w magazynie : UnitsInStock

```
CREATE TABLE [dbo].[Products](
[ProductID] [int] IDENTITY(1,1) NOT NULL,
[ProductName] [nvarchar](40) NOT NULL,
[CategoryID] [int] NOT NULL,
[QuantityPerUnit] [smallint] NOT NULL,
[UnitPrice] [money] NOT NULL,
[UnitsInStock] [smallint] NOT NULL,
PRIMARY KEY CLUSTERED ([ProductID] ASC))
```

Warunki integralnościowe:

Ilość sztuk produktu w opakowaniu musi być >= 0

```
ALTER TABLE [dbo].[Products]
ADD CONSTRAINT [CHK_Products_QuantityPerUnit] CHECK (([QuantityPerUnit]>=(0)))
```

Koszt jednej sztuki produktu musi być >= 0

```
ALTER TABLE [dbo].[Products]
ADD CONSTRAINT [CHK_Products_UnitPrice] CHECK (([UnitPrice]>=(0)))
```

Ilość sztuk w magazynie musi być >= 0

```
ALTER TABLE [dbo].[Products]
ADD CONSTRAINT [CHK_Products_UnitsInStock] CHECK (([UnitsInStock]>=(0)))
```

4.9 Tabela TablesReservations

Klucz główny: ReservationID, TableID
ID rezerwacji: ReservationID
ID stolika na który jest rezerwacja: TableID

```
CREATE TABLE [dbo].[TablesReservations](
[ReservationID] [int] NOT NULL,
[TableID] [int] NOT NULL,
PRIMARY KEY CLUSTERED ([ReservationID] ASC, [TableID] ASC))
```

4.10 Tabela EmployeesReservations

Klucz główny: ReservationID, EmployeeID
ID rezerwacji: ReservationID
ID pracownika firmy na którego złożona jest rezerwacja: EmployeeID

```
CREATE TABLE [dbo].[EmployeesReservations](
[ReservationID] [int] NOT NULL,
[EmployeeID] [int] NULL,
PRIMARY KEY CLUSTERED ([ReservationID] ASC, [EmployeeID] ASC))
```

4.11 Tabela Reservations

Klucz główny: ReservationID
ID rezerwacji: ReservationID
Początek rezerwacji: ReservationDateStart
Koniec rezerwacji: ReservationDateStart
ID zamówienia związanego z rezerwacją: OrderID
Flaga czy rezerwacja została już zaakceptowana przez pracownika: Accepted

```
CREATE TABLE [dbo].[Reservations](
[ReservationID] [int] IDENTITY(1,1) NOT NULL,
[ReservationDateStart] [datetime] NOT NULL,
[ReservationDateEnd] [datetime] NOT NULL,
[Accepted] [bit] NOT NULL,
[OrderID] [int] NOT NULL
CONSTRAINT [PK_Reservations]
PRIMARY KEY CLUSTERED ([ReservationID] ASC))
```

Warunki integralnościowe:

Początek rezerwacji musi być wcześniej od jej końca

```
ALTER TABLE [dbo].[Reservations]
ADD CONSTRAINT [CHK_Reservations_Date] CHECK
(([ReservationDateStart]<[ReservationDateEnd]))
```

4.12 Tabela Tables

Klucz główny: TableID
ID stolika: TableID
Ilość miejsc siedzących przy stole: SeatCount

```
CREATE TABLE [dbo].[Tables](
[TableID] [int] IDENTITY(1,1) NOT NULL,
[SeatCount] [tinyint] NOT NULL,
PRIMARY KEY CLUSTERED ([TableID] ASC))
```

Warunki integralnościowe:

Ilość miejsc siedzących przy stole musi być >= 1

```
ALTER TABLE [dbo].[Tables]
ADD CONSTRAINT [CHK_Tables_SeatCount] CHECK
(([SeatCount]>=(1)))
```

4.13 Tabela Employees

Klucz główny: EmployeeID
ID pracownika: EmployeeID
Nazwisko klienta: LastName
Imię klienta: FirstName
ID firmy zatrudniającej: CompanyID

```
CREATE TABLE [dbo].[Employees](
[EmployeeID] [int] IDENTITY(1,1) NOT NULL,
[FirstName] [nvarchar](40) NOT NULL,
[LastName] [nvarchar](40) NOT NULL,
[CompanyID] [int] NOT NULL,
PRIMARY KEY CLUSTERED ([EmployeeID] ASC))
```

4.14 Tabela Invoices

Klucz główny: InvoiceID
ID rachunku: InvoiceID
Data wystawienia rachunku: InvoiceDate

```
CREATE TABLE [dbo].[Invoices](
[InvoiceID] [int] IDENTITY(1,1) NOT NULL,
[InvoiceDate] [datetime] NOT NULL,
PRIMARY KEY CLUSTERED ([InvoiceID] ASC))
```

4.15 Tabela Payments

Klucz główny: PaymentID
ID wpłaty: PaymentID
ID zamówienia, na które wpłyneła wpłata: OrderID
Data zaksięgowania wpłaty: Date
Wysokość wpłaty: Amount

```
CREATE TABLE [dbo].[Payments](
[PaymentID] [int] IDENTITY(1,1) NOT NULL,
[OrderID] [int] NOT NULL,
[Date] [datetime] NOT NULL,
[Amount] [money] NOT NULL,
PRIMARY KEY CLUSTERED ([PaymentID] ASC))
```

Warunki integralnościowe:

Wysokość wpłaty musi być > 0

```
ALTER TABLE [dbo].[Payments]
ADD CONSTRAINT [CHK_Payments_Amount] CHECK (([Amount]>(0)))
```

4.16 Tabela ConfigurationVariables

Liczba dni po opłaceniu zamówienia przez, które ważna jest jednorazowa zniżka: D1
Minimalna łączna wartość zamówienia dzięki której można uzyskać stałą zniżkę: K1
Łączna kwota zamówień po której przyznawana jest jednorazowa zniżka: K2
Wysokość stałej zniżki dla klienta: R1
Wysokość jednorazowej zniżki dla klienta: R2
Liczba zamówień po których przyznawana jest stała zniżka dla klienta: Z1
Liczba złożonych i opłaconych zamówień po których można złożyć rezerwacje: WK
Minimalna wartość zamówienia do złożenia rezerwacji: WZ

```
CREATE TABLE [dbo].[ConfigurationVariables](
[D1] [int] NOT NULL,
[R1] [money] NOT NULL,
[R2] [money] NOT NULL,
[K1] [real] NOT NULL,
[K2] [real] NOT NULL,
[Z1] [int] NOT NULL,
[WK] [int] NOT NULL,
[WZ] [money] NOT NULL
```

Warunki integralnościowe:

Liczba dni po opłaceniu zamówienia przez, które ważna jest jednorazowa zniżka musi być > 0

```
ALTER TABLE [dbo].[ConfigurationVariables]
ADD CONSTRAINT [CHK_D1] CHECK ([D1]>(0))
```

Minimalna łączna wartość zamówienia dzięki której można uzyskać stałą zniżkę musi być > 0

```
ALTER TABLE [dbo].[ConfigurationVariables]
ADD CONSTRAINT [CHK_K1] CHECK ([K1]>(0))
```

Łączna kwota zamówień po której przyznawana jest jednorazowa zniżka musi być > 0

```
ALTER TABLE [dbo].[ConfigurationVariables]
ADD CONSTRAINT [CHK_K2] CHECK ([K2]>(0))
```

Wysokość stałej zniżki dla klienta musi być z przedziału [0,1]

```
ALTER TABLE [dbo].[ConfigurationVariables]
ADD CONSTRAINT [CHK_R1] CHECK ((0)<[R1] AND [R1]<(1))
```

Wysokość jednorazowej zniżki dla klienta musi być z przedziału [0,1]

```
ALTER TABLE [dbo].[ConfigurationVariables]
ADD CONSTRAINT [CHK_R2] CHECK ((0)<[R2] AND [R2]<(1))
```

Liczba zamówień po których przyznawana jest stała zniżka dla klienta musi być > 0

```
ALTER TABLE [dbo].[ConfigurationVariables]
ADD CONSTRAINT [CHK_Z1] CHECK ([Z1]>(0))
```

Liczba złożonych i opłaconych zamówień po których można złożyć rezerwacje > 0

```
ALTER TABLE [dbo].[ConfigurationVariables]
ADD CONSTRAINT [CHK_WK] CHECK ([WK]>(0))
```

Minimalna wartość zamówienia do złożenia rezerwacji > 0

```
ALTER TABLE [dbo].[ConfigurationVariables]
ADD CONSTRAINT [CHK_WZ] CHECK ([WZ]>(0))
```

Rozdział 5

Widoki

5.1 Widok MenuStatistics

Wyświetla ilość sprzedaży produktów z menu od daty pojawienia się tego menu

```
create view vwMenuStatistics as
    select M.MenuID, M.AvailableFrom, M.AvailableTo, MD.ProductID,
            (
            select sum(Quantity)
                    from OrderDetails OD
                    join Orders O on OD.OrderID = O.OrderID
                    where OD.ProductID = MD.ProductID and OrderDate between
                    M.AvailableFrom and M.AvailableTo
            ) as SellCount
    from MenuDetails MD
    join Menus M on MD.MenuID = M.MenuID
```

5.2 Widok OrderData

Wyświetla informacje o zamówieniach

```
CREATE view vwOrderData as
    select O.OrderID, dbo.udfGetOrderValue(OrderID) as OrderValue,
    CustomerID,
    OrderDate, ISNULL((
            select sum(Amount)
            from Payments P
            where P.OrderID = O.OrderID
    ), 0) as Paid,
    PickUpDate,
    InvoiceID,
    Discount,
    OneTimeDiscountID
    from Orders O
go
```

5.3 Widok UnPaidOrders

Wyświetla informacje o niezapłaconych zamówieniach

```
CREATE view vwUnPaidOrders as
    select OrderID, OrderValue, CustomerID, OrderDate,
        PickUpDate, InvoiceID, Discount, OneTimeDiscountID
    from vwOrderData OD
    where Paid < OD.OrderValue
go
```

5.4 Widok OverPaidOrders

```
create view vwOverPaidOrders as
    select OrderID, OrderValue, CustomerID, OrderDate,
        PickUpDate, InvoiceID, Discount, OneTimeDiscountID
    from vwOrderData OD
    where OD.OrderValue - OD.Paid < 0
```

5.5 Widok OrderStatisticsMonthly

Wyświetla wartość i ilość zamówień z podziałem na miesiące

```
create view vwOrderStatisticsMonthly as
    select
            year(O.OrderDate) as 'year',
            month(O.OrderDate) as 'month',
            isNull(count(O.OrderID), 0) as 'OrdersCount',
            isNull(sum(OD.Quantity * OD.UnitPrice * (1 -
            O.Discount)), 0) as 'OrdersValue'
    from Orders O
    join OrderDetails OD on O.OrderID = OD.OrderID
    group by year(O.OrderDate), month(O.OrderDate)
```

5.6 Widok OrderStatisticsWeekly

Wyświetla wartość i ilość zamówień z podziałem na tygodnie

```
create view vwOrderStatisticsWeekly as
    select
        year(O.OrderDate) as 'year',
        DATEPART(week, O.OrderDate) as 'week',
        isNull(count(O.OrderID), 0) as 'OrdersCount',
        isNull(sum(OD.Quantity * OD.UnitPrice * (1 -
        O.Discount)), 0) as 'OrdersValue'
    from Orders O
    join OrderDetails OD on O.OrderID = OD.OrderID
    group by year(O.OrderDate), DATEPART(week,
    O.OrderDate)
```

5.7 Widok ProductsOverallSales

Wyświetla dla każdego produktu sumaryczną ilość zamówionych sztuk.

```
create view vwProductsOverallSales as
    select P.ProductID,
            P.ProductName,
            (
                    select(count(OD.OrderID))
                    from OrderDetails OD
                    where OD.ProductID = P.ProductID
            ) as SalesCount,
            CategoryID,
            QuantityPerUnit,
            UnitPrice,
            UnitsInStock
    from Products P
```

5.8 Widok OrdersToPickUp

Wyświetla zamówienia, które jeszcze nie zostały dostarczone

```
create view vwOrdersToPickUp as
select OrderID, CustomerID from dbo.Orders
where  PickUpDate>GETDATE()
```

5.9 Widok CustomerStatistics

Wyświetla ilość złożonych zamówień oraz ich łączny koszt

```
create view vwCustomerStatistics as
    select C.CustomerID, count(O.OrderID) number_of_orders, sum(OD.OrderValue) as value_of_orders
        from dbo.Customers C
        inner join dbo.Orders O on C.CustomerID = O.CustomerID
        left join  dbo.vwOrderData OD on O.OrderID=OD.OrderID
group by C.CustomerID
```

5.10 Widok OwingCustomers

Wyświetla informacje o klientach, którzy nie opłacili zamówień wraz z sumaryczną wartością

```
create view  vwOwingCustomers as
select UPO.CustomerID customer_id, sum(OD.OrderValue) orders_value
from dbo.vwUnPaidOrders UPO
    inner join dbo.vwOrderData OD on UPO.CustomerID=OD.CustomerID
group by UPO.CustomerID
```

5.11 Widok CustomerEarnedDiscount

Wyświetla stałą zniżkę przyznaną danemu klientowi

```
create view vwCustomerEarnedDiscount as
select C.CustomerID, C.EarnedDiscount from dbo.Customers C
inner join dbo.Orders O on C.CustomerID = O.CustomerID
```

5.12 Widok CustomerAvailableDiscounts

Wyświetla zniżki, które jeszcze nie zostały wykorzystane przez danego klienta

```
create view vwCustomerAvailableDiscount as
    select C.CustomerID, O.Discount, O.ExpirationDate from dbo.OneTimeDiscounts O
    inner join dbo.Customers C on C.CustomerID = O.CustomerID
    where O.ExpirationDate>GETDATE()
```

5.13 Widok InvoiceData

Wyświetla informacje o fakturach

```
CREATE view vwInvoicesData as
select distinct I.InvoiceID,
       I.InvoiceDate,
       O.CustomerID,
       C.Firstname +' '+C.Lastname as 'Full_name',
       C.Address+' '+ C.City+' '+C.Country+' '+C.Region+' '+C.PostalCode as 'Address',
       C.Phone,
       sum (OD.OrderValue) as SummaryOrderValue,
       CO.CompanyID,
       CO.CompanyName from dbo.Invoices I
inner join dbo.Orders O on I.InvoiceID = O.InvoiceID
inner join dbo.Customers C on C.CustomerID = O.CustomerID
inner join dbo.vwOrderData OD on OD.OrderID=O.OrderID
left join dbo.Companies CO on CO.CompanyID=C.CompanyID
group by I.InvoiceID, I.InvoiceDate, O.CustomerID, C.Firstname +' '+C.Lastname,
         C.Address+' '+ C.City+' '+C.Country+' '+C.Region+' '+C.PostalCode, C.Phone,
         CO.CompanyID, CO.CompanyName
go

grant select on vwInvoicesData to employee
go

grant select on vwInvoicesData to owner
go
```

5.14 Widok ProductsData

Wyświetla informacje do jakiej kategorii należą dane produkty wraz z opisem tej kategorii

```
create view vwProductsData as
select P.ProductName, C.CategoryName, C.Description
from dbo.Products as P
    inner join dbo.Categories C on C.CategoryID=P.CategoryID
```

5.15 Widok MonthlySalesStatistics

Wyświetla sumaryczną ilość sprzedaży danego produktu z podziałem na konkretne miesiące

```
create view vwMonthlySalesStatistics as
select Year(OrderDate) as year,
       month(OrderDate) as month,
       P.ProductName,
       isNull(count(OD.ProductID),0) as NumberOfSales from Products P
left join OrderDetails OD on OD.ProductID=P.ProductID
left join Orders O on O.OrderID=OD.OrderID
group by P.ProductName,P.ProductID,Year(OrderDate),month(OrderDate)
```

5.16 Widok WeeklySalesStatistics

Wyświetla sumaryczną ilość sprzedaży danego produktu z podziałem na konkretne tygodnie

```
create view vwWeekSalesStatistics as
select Year(OrderDate) as year,
       datepart(week,OrderDate) as week,
       P.ProductName,
       isNull(count(OD.ProductID),0) as NumberOfSales from Products P
left join OrderDetails OD on OD.ProductID=P.ProductID
left join Orders O on O.OrderID=OD.OrderID
group by P.ProductName,P.ProductID,Year(OrderDate),datepart(week,OrderDate)
```

5.17 Widok ReservationsData

Wyświetla informacje o każdej rezerwacji, która została zaakceptowana

```
create view vwReservationsData as
select R.ReservationID,
       ReservationDateStart,
       ReservationDateEnd,
       T.TableID,
       SeatCount from Reservations R
join TablesReservations RD on R.ReservationID = RD.ReservationID and R.Accepted=1
left join Tables T on RD.TableID = T.TableID
```

5.18 Widok PendingReservationsRequests

Wyświetla informacje o każdej rezerwacji, która jeszcze nie została zaakceptowana

```
create view vwPendingReservationsRequests as
select R.ReservationID,
       ReservationDateStart,
       ReservationDateEnd,
       T.TableID,
       SeatCount from Reservations R
join TablesReservations RD on R.ReservationID = RD.ReservationID and R.Accepted=0
left join Tables T on RD.TableID = T.TableID
```

5.19 Widok FutureReservations

Wyświetla informacje o rezerwacjach które jeszcze się nie rozpoczęły

```
create view vwFutureReservations as
select R.ReservationID,
       ReservationDateStart,
       ReservationDateEnd,
       T.TableID,
       SeatCount from Reservations R
join TablesReservations RD on R.ReservationID = RD.ReservationID and R.ReservationDateStart>GETDATE()
left join Tables T on RD.TableID = T.TableID
```

5.20 Widok TablesReservationsMonthly

Wyświetla dla każdego stolika ilość jego rezerwacji z podziałem na miesiące

```
create view vwTablesReservationsMontly as
select year(ReservationDateStart) as year,
       month(ReservationDateStart) as month,
       T.TableID,
       T.SeatCount,
       isNull(count(RD.TableID),0) as NumberOfReservation from Tables T
left join TablesReservations RD on T.TableID = RD.TableID
left join Reservations R2 on RD.ReservationID = R2.ReservationID
group by year(ReservationDateStart),month(ReservationDateStart),T.TableID,T.SeatCount
```

5.21 Widok TablesReservationsWeekly

Wyświetla dla każdego stolika ilość jego rezerwacji z podziałem na tygodnie

```
create view vwTablesReservationsWeekly as
select year(ReservationDateStart) as year,
       datepart(week,ReservationDateStart) as month,
       T.TableID,
       T.SeatCount,
       count(RD.TableID) as NumberOfReservation from Tables T
left join TablesReservations RD on T.TableID = RD.TableID
left join Reservations R2 on RD.ReservationID = R2.ReservationID
group by year(ReservationDateStart),datepart(week,ReservationDateStart),T.TableID,T.SeatCount
```

Rozdział 6

Procedury

6.1 Procedura AddCategory

Przyjmuje nazwę oraz opis kategorii jako argument, w rezultacie dodaje nową kategorię do tabeli Categories.

```
CREATE PROCEDURE uspAddCategory
@CategoryName varchar(40),
@CategoryDescription text = ''
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF EXISTS(
           SELECT *
           FROM Categories
           WHERE @CategoryName = CategoryName
       )
       BEGIN
           THROW 52000, N'Kategoria o podanej nazwie istnieje już w bazie danych', 1
       END
       INSERT INTO Categories(CategoryName, Description)
       VALUES(@CategoryName, @CategoryDescription);
   END TRY
   BEGIN CATCH
       DECLARE @msg nvarchar(2048) =
       N'Błąd podaczas dodawania kategorii: ' + ERROR_MESSAGE();
       THROW 52000, @msg, 1;
   END CATCH
END
go
```

6.2 Procedura AddProduct

Procedura przyjmuje nazwę, id kategorii, ilość sztuk w porcji, cenę jednostkową produktu oraz ilość produktu w magazynie, w rezultacie dodaje produkt do tabeli Products.

```
CREATE PROCEDURE uspAddProduct
@ProductName nvarchar(40),
@CategoryID int,
@QuantityPerUnit smallint,
@UnitPrice money,
@UnitsInStock int
AS
BEGIN
   BEGIN TRY
       IF EXISTS(
           SELECT *
           FROM Products
           WHERE @ProductName = ProductName
       )
       BEGIN
           THROW 52000, N'Produkt o podanej nazwie istnieje już w bazie danych', 1;
       END
       IF NOT EXISTS(
           SELECT *
           FROM Categories
           WHERE @CategoryID = @CategoryID
       )
       BEGIN
           THROW 52000, N'Kategoria o podanym ID nie istnieje', 1
       END
       INSERT INTO Products(ProductName, CategoryID, QuantityPerUnit, UnitPrice, UnitsInStock)
           VALUES (@ProductName, @CategoryID, @QuantityPerUnit, @UnitPrice, @UnitsInStock)
    END TRY
    BEGIN CATCH
       DECLARE @msg nvarchar(2048) =
       N'Błąd podaczas dodawania produktu: ' + ERROR_MESSAGE();
       THROW 52000, @msg, 1;
    END CATCH
END
go
```

6.3 Procedura AddEmployee

Procedura przyjmuje jako argumenty imię i nazwisko pracownika oraz nazwę firmy, w rezultacie dodaje go do tabeli Employees

```
CREATE PROCEDURE uspAddEmployee
@FirstName nvarchar(20),
@LastName nvarchar(20),
@CompanyID int
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF NOT EXISTS(
           SELECT *
           FROM Companies
           WHERE @CompanyID = CompanyID
       )
       BEGIN
           THROW 52000, 'Firma o podanym ID nazwie nie istnieje', 1
       END
       INSERT INTO Employees(firstname, lastname, companyid)
       values (@FirstName, @LastName, @CompanyID)
   END TRY
   BEGIN CATCH
       DECLARE @msg nvarchar(2048) =
            N'Błąd podaczas dodawania pracownika: ' + ERROR_MESSAGE();
       THROW 52000, @msg, 1
   END CATCH
END
go
```

6.4 Procedura AddProductToMenu

Procedura przyjmuje ID produktu oraz ID Menu do którego ma zostać dodany i w rezultacie dodaje ten produkt do odpowiedniego menu w tabeli Menu.

```
CREATE PROCEDURE uspAddProductToMenu
@ProductID int,
@MenuID int
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS(
            SELECT *
            FROM Products
            WHERE ProductID = @ProductID
        )
        BEGIN
            THROW 52000, 'Produkt o podanym ID nie istnieje', 1
        END
        IF NOT EXISTS(
            SELECT *
            FROM Menus
            WHERE MenuID = @MenuID
        )
        BEGIN
            THROW 52000, 'Menu o podanym ID nie istnieje', 1
        END
        INSERT INTO MenuDetails(MenuID, ProductID)
        VALUES (@MenuID, @ProductID);
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)
        =N'Błąd dodania produktu do menu: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
END
go
```

6.5 Procedura uspAddPayment

Procedura przyjmuje ID zamówienia oraz wysokość wpłaty i w rezultacie dodaje płatność do tablicy Payments z aktualną datą.

```
CREATE PROCEDURE uspAddPayment
@OrderID int,
@Amount money
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS(
            SELECT *
            FROM Orders
            WHERE @OrderID = OrderID
        )
        BEGIN
            THROW 52000, 'Zamówienie o podanym ID nie istnieje', 1
        END
        INSERT INTO Payments(OrderID, Amount, Date)
        values (@OrderID, @Amount, GETDATE())
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048) =
            N'Błąd podaczas dodawania płatności: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
END
go
```

6.6 Procedura uspRemoveProductFromOrder

Procedura przyjmuje ID zamówienia oraz ID produktu i w rezultacie usuwa produkt z zamówienia .

```
create PROCEDURE uspRemoveProductFromOrder
@ProductID int,
@OrderID int
AS
begin
    begin try
        if not exists(select * from dbo.Products where ProductID=@ProductID)
            throw 52000,N'Podane produkt nie istnieje',1

        if not exists(select * from dbo.Orders where OrderID=@OrderID)
        begin;
            throw 52000,N'Podane zamówienie nie istnieje',1
            end

        if not exists(select * from dbo.Orders where OrderID=@OrderID and InvoiceID != null)
            begin;
                throw 52000,
                    N'Na zamówienie wygenerowana została faktura, modyfikacja jest niemożliwa',1
                end

        if not exists(
            select *
            from dbo.OrderDetails
            where OrderID=@OrderID and ProductID = @ProductID
        )
        begin;
            throw 52000,N'W zamówieniu o podanym ID nie ma produktu o podanym ID',1
            end

        delete from dbo.OrderDetails
            where OrderID=@OrderID and ProductID = @ProductID;
    end try
    begin catch
        declare @message nvarchar(2048)=
            N'Wystąpił błąd podczas usuwania z zamówienia: '+error_message();
        throw 52000,@message,1
    end catch
end
go
```

6.7 Procedura uspChangeStock

Procedura przyjmuje ID produktu i nową ilość aktualnie dostępnych produktów, w rezultacie uaktualnia ilość produktów na stanie .

```
create PROCEDURE uspChangeStock
@ProductID int,
@NewStockValue smallInt
AS
begin
    begin try
        if not exists(select * from dbo.Products where ProductID=@ProductID)
            throw 52000,N'Podany produkt nie istnieje',1

            UPDATE Products
            SET
                UnitsInStock = @NewStockValue
            WHERE
                ProductID = @ProductID;
    end try
    begin catch
        declare @message nvarchar(2048)=
            N'Wystąpił błąd podczas zmieniania dostępnej ilości produktów: '+error_message();
        throw 52000,@message,1
    end catch
end
go
```

6.8 Procedura uspAddInvoice

Procedura przyjmuje ID zamówienia, w rezultacie dodaje dane do faktury do bazy danych.

```
CREATE PROCEDURE uspAddInvoice
@OrderID int
AS
begin
    begin try
        if not exists(select * from dbo.Orders where OrderID=@OrderID)
            throw 52000,N'Podane zamówienie nie istnieje',1
        if not exists(select * from dbo.Orders where OrderID=@OrderID and InvoiceID is null)
            throw 52000,N'Na podane zamówienie została już utworzona faktura',1
        insert into Invoices (InvoiceDate) values(GETDATE())
        declare @InvoiceID int = SCOPE_IDENTITY();
        update Orders
            set InvoiceID = @InvoiceID
            where OrderID = @OrderID
    end try
    begin catch
        declare @message nvarchar(2048)=
            N'Wystąpił błąd podczas generowania faktury: '+error_message();
        throw 52000,@message,1
    end catch
end
go
```

6.9 Procedura uspMonthlyInvoice

Procedura przyjmuje ID klienta, w rezultacie dodaje tworzy fakturę zbiorcza (miesięczną).

```
CREATE PROCEDURE uspMonthlyInvoice
@CusomerID int
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF EXISTS(
            SELECT I.InvoiceID
            FROM Invoices I
            JOIN Orders O ON O.InvoiceID = I.InvoiceID
            WHERE O.CustomerID = @CusomerID
            GROUP BY I.InvoiceID
            HAVING COUNT(OrderID) > 1
        )
        BEGIN
            THROW 52000,
                N'Podany klient utworzył już fakture zbiorczą w przeciągu ostatniego miesiąca', 1;
        END

        INSERT Invoices (InvoiceDate) VALUES (GETDATE());
        UPDATE Orders
            SET InvoiceID = SCOPE_IDENTITY()
            WHERE InvoiceID is NULL and
                CustomerID = @CusomerID and
                DATEADD(MONTH, -1, GETDATE()) < OrderDate

    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048) =
        N'Błąd podaczas tworzenia faktury zbiorczej: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1;
    END CATCH
END
go
```

6.10 Procedura uspAddMenu

Procedura przyjmuje date początku i końca dostępności menu, w rezultacie dodaje nowe menu.

```
create PROCEDURE uspAddMenu
@AvailableFrom datetime,
@AvailableTo datetime
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF EXISTS(
            SELECT *
            FROM Menus
            WHERE (not ( @AvailableTo <= AvailableFrom or AvailableTo <= @AvailableFrom )) or
                ( @AvailableTo = AvailableFrom and AvailableTo = @AvailableFrom )
        )
        BEGIN
            THROW 52000, N'Podane daty kolidują z obecnymi już w bazie', 1
        END

        IF (@AvailableFrom <= DATEADD(day, -1, GETDATE()))
        BEGIN
            THROW 52000, N'Menu musi być ustalne z conajmniej dziennym wyprzedzeniem', 1
        END


        INSERT INTO Menus(AvailableFrom, AvailableTo)
            VALUES(@AvailableFrom, @AvailableTo);
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048) =
        N'Błąd podaczas dodawania menu: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1;
    END CATCH
END
go
```

6.11 Procedura AddCustomer

Procedura przyjmuje dane adresowe klienta oraz jego rodzaj. W zależności od wybranego typu klienta, procedura, powinna przyjąć również nazwę firmy lub nazwę klienta indywidualnego. W rezultacie klient dodawany jest do odpowiednich tabel.

```
create procedure uspAddCustomer
@FirstName nvarchar(20),
@LastName nvarchar(20),
@City nvarchar(20),
@Region nvarchar(20),
@PostalCode nvarchar(20),
@Phone nvarchar(20),
@Address nvarchar(20),
@Country nvarchar(20),
@ContactName nvarchar(40),
@CustomerType varchar(1),
@CompanyName varchar(20)
as begin
    set nocount on
        begin try
            if exists(
                select * from dbo.Customers where Phone=@Phone
                )
            begin;
            throw 52000, N'Numer telefonu znajduje się już w bazie',1
                end
            if exists(
                select * from dbo.Companies where CompanyName=@CompanyName
```

```
                select * from dbo.Companies where CompanyName=@CompanyName
                )
            begin;
            throw 52000, N'Nazwa firmy znajduje się juz w bazie',1
                end
            declare @ScopeIdentity int
            if @CustomerType='I'
                insert into dbo.Customers(Lastname, Firstname, Address, City, Region, PostalCode, Country, Phone, CompanyID, EarnedDiscount)
                values (@LastName,@FirstName,@Address,@City,@Region,@PostalCode,@Country,@Phone,null,0)
            if @CustomerType='C'
                insert into dbo.Companies( CompanyName, ContactName)
                values (@CompanyName,@ContactName)
                select @ScopeIdentity=scope_identity()
                insert into dbo.Customers( Lastname, Firstname, Address, City, Region, PostalCode, Country, Phone,CompanyID, EarnedDiscount)
                values (@LastName,@FirstName,@Address,@City,@Region,@PostalCode,@Country,@Phone,@ScopeIdentity,0)
            end try
        begin catch
            declare @message nvarchar(2048)=N'Wystąpił błąd przy dodawaniu klienta: '+error_message();
            throw 52000, @message, 1
        end catch
end
```

6.12 Procedura AddOrder

Procedura przyjmuje ID klienta, informację czy jest na wynos, ewentualną datę odbioru zamówienia oraz zniżkę. Następnie proceura weryfikuje poprawność wprowadzonych danych, jeżeli stwierdzi ich poprawność to dodaje zamówienie. W przeciwnym wypadku zwraca błąd.

```
create procedure uspAddOrder
@CustomerID int,
@Takeaway bit,
@PickUpDate datetime,
@Discount real
as begin
    set nocount on
    begin try
        declare @OneTimeDiscountID int
        set @OneTimeDiscountID = null
        declare @OrderDate datetime
        set @OrderDate = getdate()
        if isnull(@PickUpDate,'9999-01-01')<getdate()
            begin;
                throw 52000,N'Wprowadzono niepoprawną datę odbioru zamówienia',1
            end
        if not exists(
            select EarnedDiscount from dbo.Customers where EarnedDiscount=@Discount and CustomerID=@CustomerID
            )
        and not exists(
            select Discount from dbo.OneTimeDiscounts where @Discount=Discount and @CustomerID=CustomerID and ExpirationDate<getdate()
            )
            begin;
                throw 52000, N'Podana wartość zniżki nie istnieje',1
```

```
        if not exists(
            select EarnedDiscount from dbo.Customers where EarnedDiscount=@Discount and CustomerID=@CustomerID
            ) and exists (
                select Discount from dbo.OneTimeDiscounts where @Discount=Discount and @CustomerID=CustomerID and ExpirationDate<getdate()
            )
            begin
                set @OneTimeDiscountID=(select OneTimeDiscountID from dbo.OneTimeDiscounts where CustomerID=@CustomerID
                                and @Discount=Discount and ExpirationDate<getdate())
            end
        if(@Takeaway=1)
            begin
                insert into dbo.Orders(CustomerID, OrderDate, PickUpDate, InvoiceID, Discount, OneTimeDiscountID)
                values (@CustomerID,@OrderDate,@PickUpDate,null,0, @OneTimeDiscountID)
            end
        if(@Takeaway=0)
            begin
                insert into dbo.Orders(CustomerID, OrderDate, PickUpDate, InvoiceID, Discount,OneTimeDiscountID)
                values (@CustomerID,@OrderDate,null,null,@Discount,@OneTimeDiscountID)
            end
        end try
    begin catch
        declare @message nvarchar(2048)=N'Wystąpił błąd podczas tworzenia zamówienia: '+error_message();
        throw 52000,@message,1
    end catch
end
```

6.13 Procedura AddProductToOrder

Procedura przyjmuje ilość, nazwę produktu oraz ID zamówienia. W rezultacie informacje te dodawane są do zamówienia.

```
CREATE procedure dbo.uspAddProductToOrder
@OrderID int,
@Quantity int,
@ProductID int
as begin
    set nocount on
    begin try
        declare @QuantityOfProduct int
        set @QuantityOfProduct = (select UnitsInStock from dbo.Products where @ProductID=ProductID)
        if not exists(
            select * from dbo.Orders where OrderID=@OrderID
            )
        begin;
            throw 52000,N'Podane zamówienie nie istnieje',1
            end
        if not exists(
            select * from dbo.Products where ProductID=@ProductID
            )
        begin;
            throw 52000,N'Podany produkt nie instnieje',1
            end
        if not exists(
            select *
            from dbo.Orders
            where OrderID=@OrderID and InvoiceID is null
            )
                throw 52000,N'Na podane zamówienie została już utworzona faktura',1
        if not exists(
                select * from dbo.Products P
                join dbo.MenuDetails MD on P.ProductID=MD.ProductID
                join dbo.Menus M on MD.MenuID = M.MenuID
                    where M.AvailableFrom<getdate() and
                        M.AvailableTo>getdate() and P.ProductID=@ProductID
            )
        begin;
            throw 52000,N'Podany produkt nie znajduje się obecnie w menu',1
            end
        if (
            @QuantityOfProduct-@Quantity<0
            )
        begin
            throw  52000,N'Podany produkt skończył się lub nie ma wystarczającej ilości,
            aby pokryć zamówienie',1
        end
        declare @QuantityDiff int
            set @QuantityDiff=((
                select UnitsInStock
                from dbo.Products
                where ProductID=@ProductID
                )-@Quantity)
            exec dbo.uspChangeStock @ProductID=@ProductID,@NewStockValue=@QuantityDiff
        if exists(
            select *
            from OrderDetails
            where OrderID = @OrderID and ProductID = @ProductID
            )
```

```
            begin
                update OrderDetails
                    set Quantity=(
                        select max(Quantity)
                        from OrderDetails
                        where OrderID=@OrderID and ProductID=@ProductID
                    ) + @Quantity
                    where OrderID=@OrderID and ProductID=@ProductID
            end
        else
            begin
                insert into dbo.OrderDetails(ProductID, OrderID, Quantity, UnitPrice)
                values (@ProductID,@OrderID,@Quantity,(
                    select UnitPrice
                    from dbo.Products
                    where @ProductID=ProductID
                ))
            end

    end try
    begin catch
        declare @message nvarchar(2048)=
        N'Wystąpił błąd podczas dodawania do zamówienia: '+error_message();
        throw 52000,@message,1
    end catch
end
go
```

6.14 Procedura AddTableToReservation

Procedura przyjmuje ID rezerwacji oraz ID stolika. W rezultacie do danej rezerwacji przypisuje ID stolika.

```
create procedure uspAddTableToReservation
@ReservationID int,
@TableID int
as begin
    set nocount on
    begin try
        declare @ReservationDateStart datetime
        declare @ReservationDateEnd datetime
        set @ReservationDateStart =(select ReservationDateStart from dbo.Reservations where ReservationID=@ReservationID)
        set @ReservationDateEnd=(select ReservationDateEnd from dbo.Reservations where ReservationID=@ReservationID)
        if exists(select TableID from dbo.TablesReservations
                join dbo.Reservations R on TablesReservations.ReservationID = R.ReservationID
                where (R.ReservationDateStart between @ReservationDateStart and @ReservationDateEnd)
                   or (R.ReservationDateEnd between @ReservationDateStart and @ReservationDateEnd))
            begin;
                throw 52000,N'Wybrany stolik jest już zarezerwowany w danych godzinach',1
            end
        if not exists(
            select * from dbo.Tables where TableID=@TableID
            )
        begin;
            throw 52000,'Podany stolik nie istnieje',1
        end
        if not exists(
            select * from dbo.Reservations where ReservationID=@ReservationID
            )
        begin;
            throw 52000,'Podana rezerwacja nie istnieje',1
        end
        insert into dbo.TablesReservations(ReservationID,TableID)
        values (@ReservationID,@TableID)
    end try
    begin catch
        declare @message nvarchar(2048)=N'Wystąpił błąd przy dodawaniu stolika do rezerwacji: '+error_message();
        throw 52000, @message, 1
    end catch
end
```

6.15 Procedura AddTable

Procedura dodaje stolik do tabeli Tables wraz z jego wielkością.

```
create procedure uspAddTable
@SeatCount int
as begin
    set nocount on
    begin try
        insert into dbo.Tables(SeatCount)
        values (@SeatCount)
    end try
    begin catch
        declare @message nvarchar(2048)=N'Wystąpił błąd przy dodawaniu stolika: '+error_message();
        throw 52000, @message, 1
    end catch
end
```

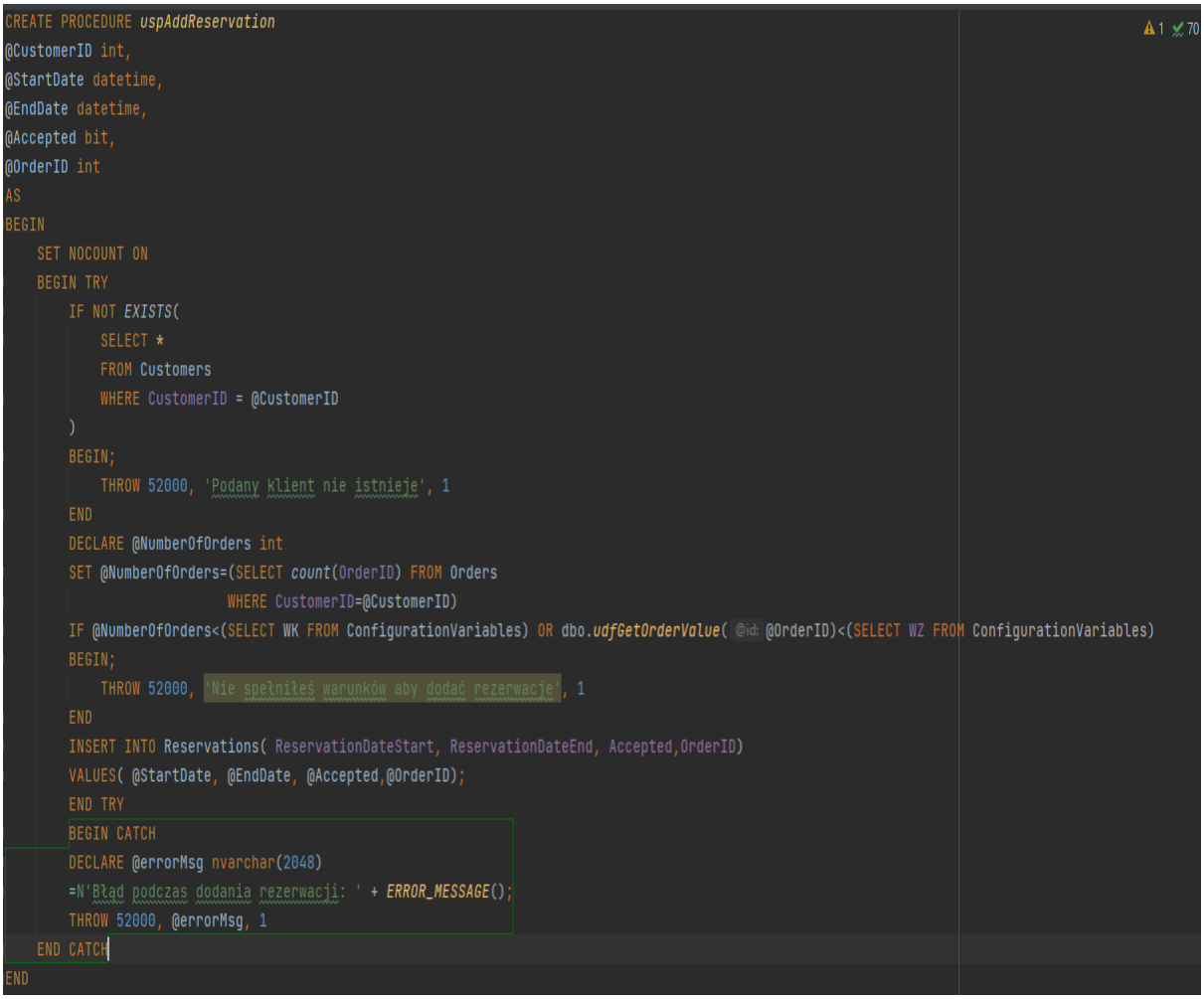
6.16 Procedura ModifyTable

Procedura w argumencie przyjmuje id stolika oraz liczbę miejsc siedzących, następnie wyszukuje stolik o danym ID i zmienia jego liczbę miejsc siedzących na ten podany w argumencie.

```
CREATE PROCEDURE uspModifyTable
@TableID int,
@SeatCount int
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS(
            SELECT * FROM Tables
            WHERE TableID=@TableID
        )
        BEGIN;
            THROW 52000, 'Podany stolik nie istnieje.', 1
        END
        IF @SeatCount < 1
        BEGIN;
            THROW 52000, N'Stolik musi mieć przynajmniej 1 miejsce siedzące.', 1
        END
        IF @SeatCount IS NOT NULL
        BEGIN
            UPDATE Tables
                SET SeatCount = @SeatCount
                WHERE Tables.TableID=@TableID
        END
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)=N'Błąd podczas edytowania stolika: ' +ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
END
```

6.17 Procedura AddReservation

Procedura przyjmuje w argumencie id klienta, czas startu oraz czas zakończenia i status rezerwacji, następnie dodaje nową rezerwację.



```
CREATE PROCEDURE uspAddReservation
@CustomerID int,
@StartDate datetime,
@EndDate datetime,
@Accepted bit,
@OrderID int
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS(
            SELECT *
            FROM Customers
            WHERE CustomerID = @CustomerID
        )
        BEGIN;
            THROW 52000, 'Podany klient nie istnieje', 1
        END
        DECLARE @NumberOfOrders int
        SET @NumberOfOrders=(SELECT count(OrderID) FROM Orders
                             WHERE CustomerID=@CustomerID)
        IF @NumberOfOrders<(SELECT WK FROM ConfigurationVariables) OR dbo.udfGetOrderValue( @id: @OrderID)<(SELECT WZ FROM ConfigurationVariables)
        BEGIN;
            THROW 52000, 'Nie spełniłeś warunków aby dodać rezerwacje', 1
        END
        INSERT INTO Reservations( ReservationDateStart, ReservationDateEnd, Accepted,OrderID)
        VALUES( @StartDate, @EndDate, @Accepted,@OrderID);
        END TRY
        BEGIN CATCH
        DECLARE @errorMsg nvarchar(2048)
        =N'Błąd podczas dodania rezerwacji: ' + ERROR_MESSAGE();
        THROW 52000, @errorMsg, 1
    END CATCH
END
```

6.18 Procedura ChangeReservationStatus

Procedura przyjmuje w argumencie id rezerwacji i zmienia jej Status na ten podany w argumencie

```
CREATE PROCEDURE uspChangeReservationStatus
@ReservationID int,
@Accepted bit
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        BEGIN
            UPDATE Reservations
                SET Accepted = @Accepted
                WHERE Reservations.ReservationID=@ReservationID
        END
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)=N'Błąd podczas edytowania rezerwacji: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
END
```

6.19 Procedura RemoveCategory

Procedura przyjmuje w argumencie id kategorii i ją usuwa

```
CREATE PROCEDURE uspRemoveCategory
@CategoryID int
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
    IF NOT EXISTS(
        SELECT *
        FROM Categories
        WHERE CategoryID = @CategoryID
    )
    BEGIN;
        THROW 52000,'Kategori o podanym ID nie istnieje',1
    end
    DELETE FROM Categories
        WHERE CategoryID = @CategoryID
    END TRY
    BEGIN CATCH
    DECLARE @msg nvarchar(2048) = N'Błąd podczas usuwania kategorii: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1;
    END CATCH
END
```

6.20 Procedura AddEmployeeToReservation

Procedura przyjmuje w argumencie id pracownika i id rezerwacji a następnie dodaje pracownika o podanym id do rezerwacji

```
CREATE PROCEDURE uspAddEmployeeToReservation
@ReservationID int,
@EmployeeID int
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS(
            SELECT *
            FROM Reservations
            WHERE ReservationID = @ReservationID
        )
        BEGIN;
            THROW 52000, 'Podana rezerwacja nie istnieje', 1
        end
        IF NOT EXISTS(
            SELECT *
            FROM Employees
            WHERE EmployeeID=@EmployeeID
            )
        BEGIN;
            THROW 52000, 'Podany pracownik nie istnieje nie istnieje', 1
        END
        INSERT INTO EmployeesReservations(reservationid, EmployeeID)
        VALUES(@ReservationID,@EmployeeID);
        END TRY
        BEGIN CATCH
        DECLARE @errorMsg nvarchar(2048)
        =N'Błąd podczas dodania pracownika do rezerwacji: ' + ERROR_MESSAGE();
        THROW 52000, @errorMsg, 1
    END CATCH
END
```

Rozdział 7

Funkcje

7.1 Funkcja udfGetOrdersMoreExpensiveThan

Zwraca tabelę z zamówieniami powyżej o wartości przekraczającej wartość otrzymaną jako argument.

```
CREATE FUNCTION udfGetOrdersMoreExpensiveThan(@value int)
RETURNS table AS
RETURN
    SELECT OD.OrderID, OD.CustomerID, OD.OrderValue
    FROM vwOrderData OD
    WHERE OD.OrderValue > @value
go
```

7.2 Funkcja udfGetMenuItemsByID

Zwraca tabelę z pozycjami danego menu którego ID przyjmuje jako argument

```
CREATE FUNCTION udfGetMenuItemsByID(@menuId int)
RETURNS TABLE AS
RETURN
    SELECT M.MenuID, M.AvailableFrom, M.AvailableTo, P.ProductName, P.UnitPrice
    FROM Products P
        JOIN MenuDetails MD on MD.ProductID = P.ProductID
        JOIN Menus M on M.MenuID = MD.MenuID
    WHERE (M.MenuID = @menuId)
go
```

7.3 Funkcja udfGetMenuItemsByDate

Zwraca tabelę z pozycjami menu aktualnego dla daty podanej jako argument.

```
CREATE FUNCTION udfGetMenuItemsByDate(@date date)
RETURNS TABLE AS
RETURN
    SELECT M.MenuID, M.AvailableFrom, M.AvailableTo, P.ProductName, P.UnitPrice
    FROM Products P
        JOIN MenuDetails MD on MD.ProductID = P.ProductID
        JOIN Menus M on M.MenuID = MD.MenuID
    WHERE @date BETWEEN M.AvailableFrom AND M.AvailableTo
go
```

7.4 Funkcja udfGetMealsSoldAtLeastXTimes

Zwraca pozycje, które sprzedały się więcej niż przyjętą jako argument liczbę razy.

```
CREATE FUNCTION udfGetMealsSoldAtLeastXTimes(@timesSold int)
    RETURNS table AS
        RETURN
        SELECT ProductName,SUM(Quantity) as [Ilość sprzedanych sztuk] FROM Products P
        JOIN OrderDetails OD on P.ProductID = OD.ProductID
        GROUP BY ProductName
        HAVING SUM(Quantity)>@timesSold
go
```

7.5 Funkcja udfGetValueOfOrdersOnDay

Zwraca wartość zamówień złożonych w danym jako argument dniu.

```
CREATE FUNCTION udfGetValueOfOrdersOnDay(@date date)
    RETURNS money
AS
BEGIN
    RETURN ISNULL((
        SELECT SUM(OD.OrderValue)
        FROM vwOrderData OD
            JOIN Orders O on OD.OrderID = O.OrderID
        WHERE YEAR(@date) = YEAR(OD.OrderDate)
            AND MONTH(@date) = MONTH(OD.OrderDate)
            AND DAY(@date) = DAY(OD.OrderDate)
    ), 0)
END
go
```

7.6 Funkcja udfGetValueOfOrdersInMonth

Zwraca wartość zamówień złożonych w danym jako argument miesiącu roku.

```
CREATE FUNCTION udfGetValueOfOrdersInMonth(@year int, @month int)
RETURNS money
AS
BEGIN
    RETURN ISNULL((
        SELECT SUM(OD.OrderValue)
        FROM vwOrderData OD
        INNER JOIN Orders O on OD.OrderID = O.OrderID
        WHERE @year = YEAR(OD.OrderDate)
        AND @month = MONTH(OD.OrderDate)
    ), 0)
END
go
```

7.7 Funkcja udfGetAvgPriceOfMenu

Zwraca średnią cenę produktów z menu którego ID przyjmuje jako argument.

```
CREATE FUNCTION udfGetAvgPriceOfMenu(@MenuID int)
RETURNS money
AS
BEGIN
    RETURN (
        SELECT AVG(P.UnitPrice)
        FROM MenuDetails M
        JOIN Products P on M.ProductID = P.ProductID
        WHERE MenuID = @MenuID
    )
END
go
```

7.8 Funkcja udfGetBestProducts

Zwraca daną liczbę najczęściej kupowanych produktów.

```
create function udfGetBestProducts(@val int)
    returns table as
        return select distinct top (@val) P.ProductName, POS.SalesCount from dbo.Products P
        join dbo.vwProductsOverallSales POS on P.ProductID=POS.ProductID
        order by POS.SalesCount desc
```

7.9 Funkcja udfGetCustomersOrderedMoreThanXTimes

Zwraca klientów, którzy zamówili co najmniej daną ilość razy.

```
create function udfGetCustomersOrderedMoreThanXTimes(@amount int)
    returns table as
        return select CS.CustomerID, C.Firstname, C.Lastname, C.Address, C.Region, C.Phone, C.PostalCode from  dbo.vwCustomerStatistics CS
            join dbo.Customers C on C.CustomerID=CS.CustomerID
            where number_of_orders>@amount
```

7.10 Funkcja udfGetCustomersWhoOweMoreThanX

Zwraca klientów, którzy są dłużni na kwotę większą lub równą od wartości przyjętej jako argument.

```
create function udfGetCustomersWhoOweMoreThanX(@val int)
    returns table as
    return select customer_id, orders_value from dbo.vwOwingCustomers
        where orders_value>@val
```

7.11 Funkcja udfGetBestOneTimeDiscount

Zwraca wartość zniżek jednorazowych dla klienta.

```
create function udfGetBestOneTimeDiscount(@CustomerID int)
returns real
as begin
    return (select top 1 max(Discount) from dbo.vwCustomerAvailableDiscount
            where CustomerID=@CustomerID)
end
```

7.12 Funkcja udfGetBestEarnedDiscount

Zwraca wartość zniżki stałej dla klienta.

```
create function udfGetBestEarnedDiscount(@CustomerID int)
returns real
as begin
    return (select top 1 max(EarnedDiscount) from dbo.vwCustomerEarnedDiscount
        where CustomerID=@CustomerID)
end
```

7.13 Funkcja udfGetEmployeesOfCompany

Zwraca pracowników firmy zadanej jako argument. Jeżeli firma nie istnieje zwraca null.

```
create function udfGetEmployeesOfCompany(@CompanyName nvarchar(40))
    returns table as
    return
    select E.Firstname,E.Lastname from dbo.Employees E
        join dbo.Companies C on E.CompanyID = C.CompanyID
        where @CompanyName=CompanyName
```

7.14 Funkcja udfGetMaxPriceOfMenu

Zwraca największą cenę produktu dla przyjętego jako argument ID Menu.

```
create function udfGetMaxPriceOfMenu(@MenuID int)
    returns money
as
begin
    return (select top 1 max(P.UnitPrice) from dbo.Products P
            join dbo.MenuDetails MD on MD.ProductID=P.ProductID
            join dbo.Menus M on MD.MenuID = M.MenuID
               where M.MenuID=@MenuID)
end
```

7.15 Funkcja udfGetMealsSoldAtLeastXTimes

W argumencie przyjmuje liczbę, następnie zwraca posiłki sprzedane co najmniej daną liczbę razy.

```
CREATE FUNCTION udfGetMealsSoldAtLeastXTimes(@input int)
    RETURNS table AS
        RETURN
        SELECT ProductName,SUM(Quantity) as [Ilość sprzedanych sztuk] FROM Products P
        JOIN OrderDetails OD on P.ProductID = OD.ProductID
        GROUP BY ProductName
        HAVING SUM(Quantity)>@input
go
```

7.16 Funkcja udfGetMinPriceOfMenu

Przyjmuje jako argument ID menu, następnie zwraca najtańszą cenę produktu z tego menu.

```
CREATE FUNCTION udfGetMinPriceOfMenu(@MenuID int)
    RETURNS money
AS
BEGIN
    RETURN
    (SELECT MIN(P.UnitPrice)
    FROM MenuDetails M
    JOIN Products P on M.ProductID = P.ProductID and MenuID = @MenuID)
END
go
```

7.17 Funkcja udfMenuIsCorrect

Sprawdza czy aktualne Menu jest poprawne(czyli zmiana połowy pozycji w dwa tygodnie wstecz)

```
CREATE FUNCTION udfMenuIsCorrect()
    RETURNS bit
AS
BEGIN
    DECLARE @PreviousMenuID int
    DECLARE @CurrentMenu int
    SET @CurrentMenu=(SELECT MenuID FROM Menus
    WHERE GETDATE() BETWEEN AvailableFrom AND AvailableTo)
    SET @PreviousMenuID=(SELECT MenuID FROM Menus
    WHERE DATEADD(day,-14,GETDATE()) BETWEEN AvailableFrom AND AvailableTo)
    DECLARE @sameIteams int
    SET @sameIteams=(SELECT count(*)FROM
                    (SELECT ProductID FROM MenuDetails MD
                    JOIN Menus M ON MD.MenuID = M.MenuID AND M.MenuID=@CurrentMenu
                    INTERSECT
                    SELECT ProductID FROM MenuDetails MD
                    JOIN Menus M ON MD.MenuID = M.MenuID AND M.MenuID=@PreviousMenuID) AS POD)
    DECLARE @minChange int
    SET @minChange=(SELECT COUNT(*) FROM MenuDetails WHERE MenuID=@PreviousMenuID)/2
    IF @sameIteams<=@minChange
    BEGIN
        RETURN 1
    end
    RETURN 0
END
go
```

7.18 Funkcja udfGetOrderValue

Przyjmuje jako argument id zamówienia, następnie zwraca jego wartość

```
CREATE FUNCTION udfGetOrderValue(@id int)
    RETURNS money
AS
BEGIN
    RETURN
    ISNULL((SELECT SUM((Quantity*UnitPrice))*(1-MIN(O.Discount))
    FROM OrderDetails
    join Orders O on OrderDetails.OrderID = O.OrderID
    WHERE O.OrderID=@id
    GROUP BY O.OrderID), 0)
END
go
```

7.19 Funkcja udfGetOrderDiscountValue

Przyjmuje jako argument id zamówienia, następnie zwraca rabat zamówienia

```
CREATE FUNCTION udfGetOrderDiscountValue(@OrderID int)
    RETURNS real AS
BEGIN
    RETURN
    (SELECT DISTINCT Discount FROM Orders
    WHERE @OrderID=OrderID)
END
go
```

7.20 Funkcja udfGetBestDiscount

Przyjmuje jako argument id klienta, następnie zwraca najbardziej korzystny rabat, który aktualnie jest dostępny dla tego klienta

```
CREATE FUNCTION udfGetBestDiscount(@OrderID int)
    RETURNS real AS
BEGIN
    RETURN
    (SELECT MAX(Discount)
     FROM (VALUES (dbo.udfGetBestEarnedDiscount ( @CustomerID: @OrderID)),(dbo.udfGetBestOneTimeDiscount( @CustomerID: @OrderID)))
        AS AllDiscount(Discount))
END
go
```

Rozdział 8

Triggery

8.1 Trigger MondaySeaFoodCheck

Trigger ten za zadanie ma blokować zamówienia, w których znajdują się owoce morza, a nie zostały złożone do poniedziałku poprzedzającego zamówienie.

```
create trigger TR_OrderDetails_MondaySeaFoodCheck
on dbo.OrderDetails
after insert
as
begin
    set nocount on;
        if exists(
            select * from inserted as i
            inner join dbo.Orders as O on i.OrderID=O.OrderID
            inner join dbo.Products as P on i.ProductID=P.ProductID
            inner join dbo.Reservations as R on O.OrderID=R.OrderID
            where(datename(weekday, O.PickUpDate) like 'Thursday'
                and datediff(day,O.OrderDate, O.PickUpDate)<=2
                and P.CategoryID=5)
                    or (datename(weekday, O.PickUpDate) like 'Friday'
                and datediff(day,O.OrderDate, O.PickUpDate)<=3
                and P.CategoryID=5)
                    or (datename(weekday, O.PickUpDate) like 'Saturday'
                and datediff(day,O.OrderDate, O.PickUpDate)<=4
                and P.CategoryID=5)
                    or (datename(weekday, R.ReservationDateStart) like 'Thursday'
                and datediff(day,O.OrderDate, R.ReservationDateStart)<=2
                and P.CategoryID=5)
```

```
                    or (datename(weekday, R.ReservationDateStart) like 'Friday'
                and datediff(day,O.OrderDate, R.ReservationDateStart)<=3
                and P.CategoryID=5)
                    or (datename(weekday, R.ReservationDateStart) like 'Saturday'
                and datediff(day,O.OrderDate, R.ReservationDateStart)<=4
                and P.CategoryID=5)
            )
        begin;
        throw 50001, N'Zamównienie zawierajce owoce morza powinno być złożone maksymalnie do poniedziałku poprzedzającego zamówienie. ',1
    end
end
```

8.2 Trigger UpdateEarnedDiscount

Trigger ten ma blokować zniżki, które zostały wprowadzone ze złymi danymi

```
create trigger TR_OneTimeDiscount_UpdateEarnedDiscount
on OneTimeDiscounts
for insert
as
begin
    if(select OneTimeDiscountID from inserted)not in(select OneTimeDiscountID from OneTimeDiscounts)
    begin
        RAISERROR('Podany OneTimeDiscountID nie istnieje', 16, 1)
        rollback transaction
    end
    if(select ExpirationDate from inserted)<GETDATE()
    begin
        RAISERROR('Podaj date większą niż dzisiaj! ', 16, 1)
        rollback transaction
    end
    if(select CustomerID from inserted)not in(select CustomerID from Customers)
    begin
        RAISERROR('Podany użytkownik nie istnieje', 16, 1)
        rollback transaction
    end
end
```

8.3 Trigger GrantOneTimeDiscount

Trigger ma dodawać zniżkę jednorazową, jeżeli zostały spełnione odpowiednie warunki

```
create trigger TR_OneTimeDiscount_GrantOneTimeDiscount
on Payments
after insert
as
begin
    set NOCOUNT on
    begin
        declare @CustomerID int
        set @CustomerID=(select CustomerID from Orders O
        join Payments P
        on O.OrderID=P.OrderID and P.PaymentID=(select PaymentID from inserted))
        declare @StartingDay date
        set @StartingDay=(select max(BeginingDate) from OneTimeDiscounts
        where CustomerID=@CustomerID)
        if(select count(Amount) from Payments P
        join Orders O on P.OrderID = O.OrderID and CustomerID=@CustomerID
        where P.Date>@StartingDay)>=(select K2 from ConfigurationVariables)
        begin
            insert into OneTimeDiscounts (CustomerID,BeginingDate, ExpirationDate, Discount)
            values (@CustomerID,getdate(),DATEADD(day,(select D1 from ConfigurationVariables),getdate()),(select R2 from ConfigurationVariables))
        end
    end
end
go
```

Rozdział 9

Indeksy

Indeks PK_Categories

```
ALTER TABLE [dbo].[Categories]
ADD  CONSTRAINT [PK_Categories] PRIMARY KEY CLUSTERED ([CategoryID] ASC)
```

Indeks PK_Customers

```
ALTER TABLE [dbo].[Customers]
ADD  CONSTRAINT [PK_Customers] PRIMARY KEY CLUSTERED ([CustomerID] ASC)
```

Indeks IX_Customers_FirstName

```
CREATE NONCLUSTERED INDEX IX_Customers_FirstName
ON Customers (FirstName);
```

Indeks IX_Customers_LastName

```
CREATE NONCLUSTERED INDEX IX_Customers_LastName
ON Customers (LastName);
```

Indeks UQ_Customers_CompanyID

```
CREATE UNIQUE NONCLUSTERED INDEX UQ_Customers_CompanyID
ON Customers(CompanyID)
WHERE CompanyID IS NOT NULL;
```

Indeks PK_Menus

```
ALTER TABLE [dbo].[Menus]
ADD  CONSTRAINT [PK_Menus] PRIMARY KEY CLUSTERED ([MenuID] ASC)
```

Indeks PK_MenuDetails

```
ALTER TABLE [dbo].[MenuDetails]
ADD  CONSTRAINT [PK_MenuDetails] PRIMARY KEY CLUSTERED ([MenuID] ASC, [ProductID] ASC)
```

Indeks PK_OrderDetails

```
ALTER TABLE [dbo].[OrderDetails]
ADD  CONSTRAINT [PK_OrderDetails] PRIMARY KEY CLUSTERED ([ProductID] ASC, [OrderID] ASC)
```

Indeks PK_Orders

```
ALTER TABLE [dbo].[Orders]
ADD  CONSTRAINT [PK_Orders] PRIMARY KEY CLUSTERED ([OrderID] ASC)
```

Indeks PK_Products

```
ALTER TABLE [dbo].[Products]
ADD  CONSTRAINT [PK_Products] PRIMARY KEY CLUSTERED ([ProductID] ASC)
```

Indeks PK_TablesReservations

```
ALTER TABLE [dbo].[TablesReservations]
ADD  CONSTRAINT [PK_TablesReservations] PRIMARY KEY CLUSTERED ([ReservationID] ASC, [TableID] ASC)
```

Indeks PK_EmployeesReservations

```
ALTER TABLE [dbo].[EmployeesReservations]
ADD  CONSTRAINT [PK_EmployeesReservations] PRIMARY KEY CLUSTERED ([ReservationID] ASC, [EmployeeID]
```

Indeks PK_Reservations

```
ALTER TABLE [dbo].[Reservations]
ADD  CONSTRAINT [PK_Reservations] PRIMARY KEY CLUSTERED ([ReservationID] ASC)
```

Indeks UQ_Reservations_OrderID

```
Create UNIQUE NONCLUSTERED INDEX UQ_Reservations_OrderID
ON Reservations(OrderID)
WHERE OrderID IS NOT NULL;
```

Indeks PK_Tables

```
ALTER TABLE [dbo].[Tables]
ADD  CONSTRAINT [PK_Tables] PRIMARY KEY CLUSTERED ([TableID] ASC)
```

Indeks PK_Employees

```
ALTER TABLE [dbo].[Employees]
ADD  CONSTRAINT [PK_Employees] PRIMARY KEY CLUSTERED ([EmployeeID] ASC)
```

Indeks IX_Employees_FirstName

```
CREATE NONCLUSTERED INDEX IX_Employees_FirstName
ON Employees (FirstName);
```

Indeks IX_Employees_LastName

```
CREATE NONCLUSTERED INDEX IX_Employees_LastName
ON Employees (LastName);
```

Indeks PK_Invoices

```
ALTER TABLE [dbo].[Invoices]
ADD  CONSTRAINT [PK_Invoices] PRIMARY KEY CLUSTERED ([InvoiceID] ASC)
```

Indeks PK_Payments

```
ALTER TABLE [dbo].[Payments]
ADD  CONSTRAINT [PK_Payments] PRIMARY KEY CLUSTERED ([PaymentID] ASC)
```

Indeks PK_OneTimeDiscounts

```
    ALTER TABLE [dbo].[OneTimeDiscounts]
    ADD  CONSTRAINT [PK_OneTimeDiscounts] PRIMARY KEY CLUSTERED ([OneTimeDiscountID] ASC)
```

Rozdział 10

Uprawnienia

10.1 Klient zarejestrowany

Klient zarejestrowany ma dostęp do składania zamówień, ale także, w przypadku złożenia rezerwacji, do płatności online za nadchodzące zamówienia.

```
create role registeredCustomer
grant execute on uspAddOrder to registeredCustomer
grant execute on uspAddPayment to registeredCustomer
```

10.2 Klient niezarejestrowany

Klient niezarejestrowany ma dostęp jedynie do składania zamówień.

```
create role notRegisteredCustomer
grant execute on uspAddOrder to notRegisteredCustomer
```

10.3 Klient firma

Klient firma ma dostęp do takich samych funkcjonalności co klient zarejestrowany, a ponadto może dodawać wiele stolików do zamówienia oraz powiększać rezerwacje o kolejnych pracowników.

```
create role companyCustomer
grant execute on uspAddOrder to companyCustomer
grant execute on uspAddPayment to companyCustomer
grant execute on uspAddTableToReservation to companyCustomer
grant execute on uspAddEmployeeToReservation to companyCustomer
```

10.4 Pracownik

Pracownik ma dostęp do większości funkcjonalności bazy danych. Może dodawać zamówienia, modyfikować je, dodawać produkty, czy kategorie, dodawać rezerwacje, płatności, produkty do menu, czy też modyfikować stoliki.

```
create role employee
grant select on dbo.vwCustomerAvailableDiscount to employee
grant select on dbo.vwCustomerEarnedDiscount to employee
grant select on dbo.vwCustomerStatistics to employee
grant select on dbo.vwFutureReservations to employee
grant select on dbo.vwInvoicesData to employee
grant select on dbo.vwMenuStatistics to employee
grant select on dbo.vwMonthlySalesStatistics to employee
grant select on dbo.vwOrderData to employee
grant select on dbo.vwOrderStatisticsMonthly to employee
grant select on dbo.vwOrderStatisticsWeekly to employee
grant select on dbo.vwOrdersToPickUp to employee
grant select on dbo.vwOverpaidOrders to employee
grant select on dbo.vwOwingCustomers to employee
grant select on dbo.vwPendingReservationsRequests to employee
grant select on dbo.vwProductsData to employee
grant select on dbo.vwProductsOverallSales to employee
grant select on dbo.vwReservationsData to employee
grant select on dbo.vwTablesReservationsMontly to employee
grant select on dbo.vwTablesReservationsWeekly to employee
grant select on dbo.vwUnPaidOrders to employee
grant select on dbo.vwWeeklySalesStatistics to employee
```

```
grant execute on dbo.uspChangeStock to employee
grant execute on dbo.uspAddCategory to employee
grant execute on dbo.uspAddReservation to employee
grant execute on dbo.uspAddOrder to employee
grant execute on dbo.uspAddProductToMenu to employee
grant execute on dbo.uspAddProductToOrder to employee
grant execute on dbo.uspAddPayment to employee
grant execute on dbo.uspAddTabelToReservation to employee
grant execute on dbo.uspRemoveProductFromOrder to employee
grant execute on dbo.uspRemoveCategory to employee
grant execute on dbo.uspChangeReservationStatus to employee
grant execute on dbo.uspAddTable to employee
grant execute on dbo.uspAddTabelToReservation to employee
grant execute on dbo.uspAddEmployeeToReservation to employee
```

10.5 Administrator

Administrator ma dostęp do dowolnej funkcji bazy danych.

```
create role administarator
grant all privileges on u_jwos.dbo to administrator
```

10.6 Właściciel

Właściciel, jako osoba nadzorująca, ma dostęp do wszystkich widoków i tabel. Nie wykonuje procedur, gdyż takie zadanie leży w kompetencjach personelu.

```
create role owner
grant select on dbo.Categories to owner
grant select on dbo.Companies to owner
grant select on dbo.ConfigurationVariables to owner
grant select on dbo.Customers to owner
grant select on dbo.Employees to owner
grant select on dbo.EmployeesReservations to owner
grant select on dbo.Invoices to owner
grant select on dbo.MenuDetails to owner
grant select on dbo.Menus to owner
grant select on dbo.OneTimeDiscounts to owner
grant select on dbo.OrderDetails to owner
grant select on dbo.Orders to owner
grant select on dbo.Payments to owner
grant select on dbo.Products to owner
grant select on dbo.Reservations to owner
grant select on dbo.Tables to owner
grant select on dbo.TablesReservations to owner

grant select on dbo.vwCustomerAvailableDiscount to owner
grant select on dbo.vwCustomerEarnedDiscount     to owner
grant select on dbo.vwCustomerStatistics to owner
grant select on dbo.vwFutureReservations to owner
```

```
grant select on dbo.vwFutureReservations to owner
grant select on dbo.vwInvoicesData to owner
grant select on dbo.vwMenuStatistics to owner
grant select on dbo.vwMonthlySalesStatistics to owner
grant select on dbo.vwOrderData to owner
grant select on dbo.vwOrderStatisticsMonthly to owner
grant select on dbo.vwOrderStatisticsWeekly to owner
grant select on dbo.vwOrdersToPickUp to owner
grant select on dbo.vwOverpaidOrders to owner
grant select on dbo.vwOwingCustomers to owner
grant select on dbo.vwPendingReservationsRequests to owner
grant select on dbo.vwProductsData to owner
grant select on dbo.vwProductsOverallSales to owner
grant select on dbo.vwReservationsData to owner
grant select on dbo.vwTablesReservationsMontly to owner
grant select on dbo.vwTablesReservationsWeekly to owner
grant select on dbo.vwUnPaidOrders to owner
grant select on dbo.vwWeeklySalesStatistics to owner
```