Projekt Bazy Danych

Podstawy Baz Danych 2022/23

Miron Czech, Sebastian Piaskowy, Maksymilian Zawiślak

Spis treści

Typy użytkowników

Rodzaje użytkowników systemu

1. Klient indywidualny
2. Klient grupowy (firmowy)
3. Klient no-name
4. Administrator
5. Pracownik (kelner)
6. Manager restauracji

Opis funkcji systemu dla danego użytkownika

1. Klient indywidualny
    - Składanie zamówienia
        - rezerwacja stolika (wraz z zamówieniem)
        - zamówienie na wynos (zamówienie opłacane z góry)
    - Generowanie raportów tygodniowych i miesięcznych o swoich zamówieniach
    - Informacje o swoich rabatach
    - Odczyt aktualnego menu
2. Klient grupowy (firmowy)
    - Składanie zamówień
        - Rezerwacja stolika
            - Na firmę
            - Imiennie (na pracowników)
        - zamówienia na wynos (zamówienia opłacane z góry)
    - Generowanie raportów tygodniowych i miesięcznych o swoich zamówieniach
    - Odczyt aktualnego menu
3. Klient no-name
    - Składanie zamówień na miejscu
4. Administrator
    - Pełna kontrola nad systemem, czyli:
        - Wykonywanie instrukcji DROP DATABASE, START DATABASE i ROLLFORWARD DATABASE.
        - Modyfikacja, dodawanie i usuwanie dowolnych obiektów w bazie, niezależnie od tego, kto jest ich właścicielem.
        - Tworzenie tabel, widoków i indeksów, które będą własnością innych użytkowników.
        - Nadawanie uprawnień do bazy danych, w tym uprawnień administracyjnych, innemu użytkownikowi.
5. Pracownik (kelner)
    - Odczyt aktualnego menu
    - Rezerwacja stolika dla klienta
        - Akceptowanie rezerwacji
        - Zarządzanie stolikami
    - Wystawianie rachunków
6. Manager restauracji
    - To co pracownik
    - Ustalanie menu
        - Dodawanie nowych produktów
        - Usuwanie produktów
    - Generowanie raportów miesięcznych i tygodniowych dotyczących
        - Menu
        - Rezerwacji
        - Stolików
        - Rabatów

    - Statystyk zamówienia
- Ustalanie wartości
    - Z1, K1, R1%, K2, R2%, D1, WZ, WK

Ogólne funkcjonalności systemu

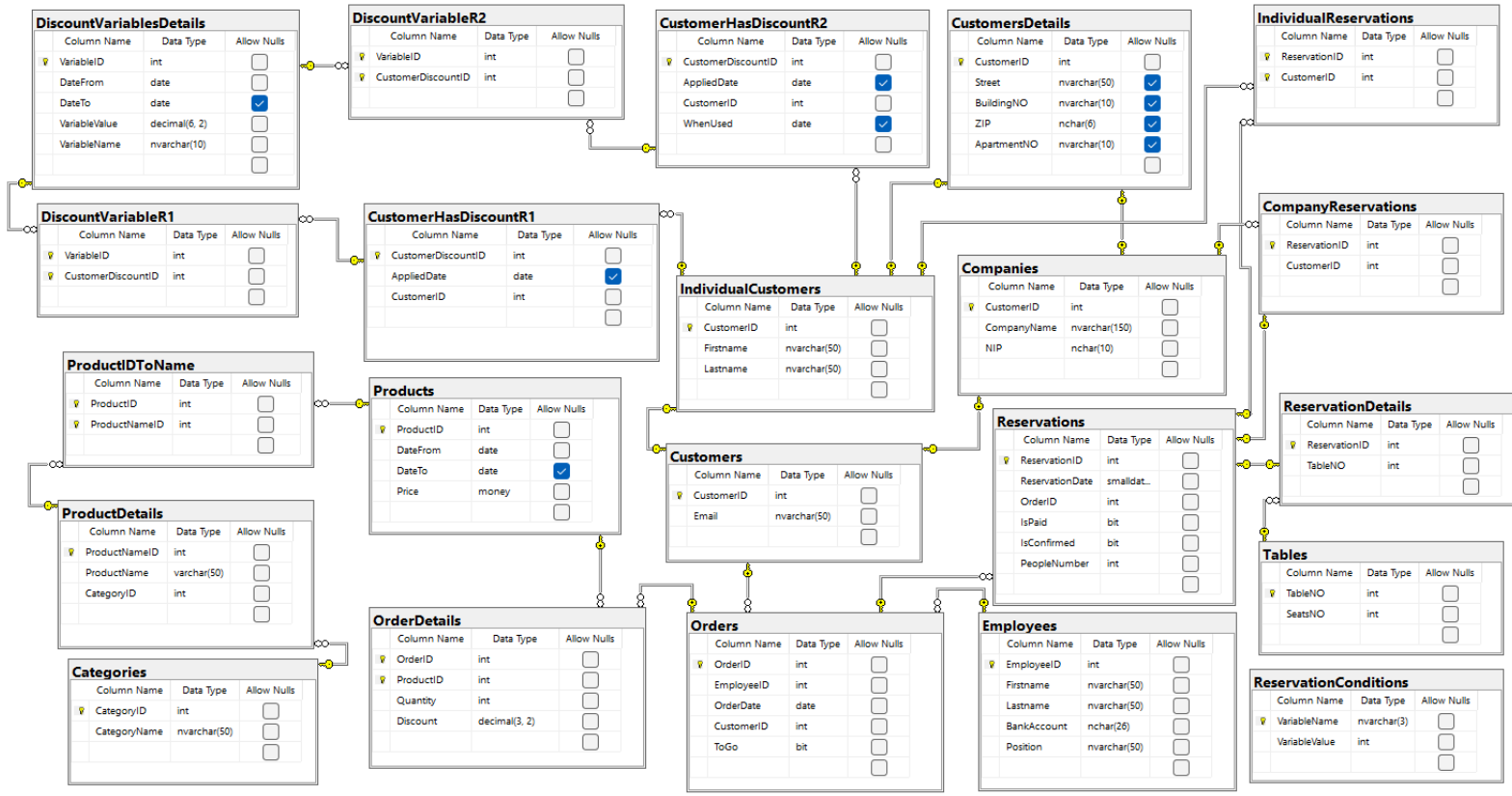
Automatyczne

1. System powiadamia managerów o tym, że nie została zmieniona połowa pozycji w menu względem 2 tygodni wcześniej, jeśli taka sytuacja zaistnieje
2. System sprawdza przy rezerwacji dla klienta indywidualnego czy spełnia wymagane warunki
3. Wyliczanie łącznego kosztu zamówienia
4. Wyliczanie i przydzielanie rabatów
5. System nie pozwala na zamawianie pozycji z menu zawierających owoce morza później niż do poniedziałku poprzedzającego zamówienie i na dni inne niż czwartek, piątek i sobota
6. System powiadamia managerów, jeśli nie ma menu na następny dzień

Funkcjonalności użytkownika

1. Funkcjonalność składania zamówień
2. Akceptacja rezerwacji

Schemat bazy danych



Opisy Tabel

Categories

Numer kategorii: CategoryID
Nazwa kategorii: CategoryName

Warunki integralnościowe:

- CategoryName - unikalna

```
create table Categories
(
    CategoryID   int          not null
        constraint PK_Categories
            primary key,
    CategoryName nvarchar(50) not null
)
go

create unique index IX_Categories
    on Categories (CategoryID)
go
```

Companies

Numer klienta: ClientID
Nazwa firmy: CompanyName
Numer NIP: NIP

Warunki integralnościowe:

- CompanyName - unikalna
- NIP - XXXXXXXXXX, gdzie X to cyfra od 0 do 9
- NIP - unikalny

```
create table Companies
(
    CustomerID  int           not null
        constraint PK_Companies
            primary key
        constraint FK_Companies_Customers
            references Customers
        constraint FK_Companies_CustomersDetails
            references CustomersDetails,
    CompanyName nvarchar(150) not null,
    NIP         nchar(10)     not null
        constraint CK_Companies
            check ([NIP] like
             '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]')
)
go

create unique index IX_Companies
    on Companies (NIP)
go

create unique index IX_Companies_1
    on Companies (CompanyName)
go
```

CompanyReservation

Numer rezerwacji: ReservationID
Numer klienta: CutomerID

```
create table CompanyReservations
(
    ReservationID int not null
        constraint PK_CompanyReservations
            primary key
        constraint FK_CompanyReservations_Reservations
            references Reservations,
    CustomerID    int not null
        constraint FK_CompanyReservations_Companies
            references Companies
)
go
```

Customers

Numer kilenta: CutomerID
Adres Email: Email

Warunki integralnościowe:

- Email - zawiera “@” - (LIKE ‘%@%’)
- Email - unikalny

```
create table Customers
(
    CustomerID int          not null
        constraint PK_Customers
            primary key,
    Email      nvarchar(50) not null
        constraint CK_Customers
            check ([Email] like '%@%')
)
go

create unique index IX_Customers
    on Customers (Email)
go
```

CustomersDetails

Numer klienta: CustomerID
Ulica: Street
Numer budynku: BuildingNO
Kod pocztowy: ZIP
Numer mieszkania: ApartmentNO

Warunki integralnościowe:

- ZIP - XX-XXX, gdzie X to cyfra od 0 do 9

```
create table CustomersDetails
(
    CustomerID  int not null
        constraint PK_CustomersDetails
            primary key,
    Street      nvarchar(50),
    BuildingNO  nvarchar(10),
    ZIP         nchar(6)
        constraint CK_CustomersDetails
            check ([ZIP] like '[0-9][0-9]-[0-9][0-9][0-9]'),
    ApartmentNO nvarchar(10)
)
go
```

CustomerHasDiscountR1

Numer zniżki klienta: CutomerDiscountID
Data przyznania zniżki: AppliedDate
Numer klienta: CustomerID

```
create table CustomerHasDiscountR1
(
    CustomerDiscountID int not null
        constraint PK_CustomerHasDiscountR1
            primary key,
    AppliedDate        date,
    CustomerID         int not null
        constraint FK_CustomerHasDiscountR1_IndividualCustomers
            references IndividualCustomers
)
go
```

CustomerHasDiscountR2

Numer zniżki klienta: CutomerDiscountID
Data przyznania zniżki: AppliedDate
Numer klienta: CustomerID
Data użycia zniżki: WhenUsed

```
create table CustomerHasDiscountR2
(
    CustomerDiscountID int not null
        constraint PK_CustomerHasDiscountR2
            primary key,
    AppliedDate        date,
    CustomerID         int not null
        constraint FK_CustomerHasDiscountR2_IndividualCustomers
            references IndividualCustomers,
    WhenUsed           date
)
go
```

DiscountVariablesDetails

Numer zmienej: VariableID
Data początku obowiązywania: DateFrom
Data końca obowiązywania: DateTo
Wartości zmiennej: VariableValue
Nazwa zmiennej: VariableName

Warunki integralnościowe:

- VariableName - zawiera się w (‘K1’, ‘R1’, ‘K2’, ‘R2’, ‘D1’, ‘Z1’)
- VariableValue >=0
- DateFrom <= DateTo OR DateTo IS NULL

```
create table DiscountVariablesDetails
(
    VariableID    int           not null
        constraint PK_DiscountVariableNames
            primary key,
    DateFrom      date          not null,
    DateTo        date,
    VariableValue decimal(6, 2) not null
        constraint CK_DiscountVariablesDetails_1
            check ([VariableValue] >= 0),
    VariableName  nvarchar(10)  not null
        constraint CK_DiscountVariablesNames
            check ([VariableName] = 'D1' OR [VariableName] = 'Z1'
              OR [VariableName] = 'R2' OR [VariableName] = 'K2' OR
                   [VariableName] = 'R1' OR [VariableName] = 'K1'),
    constraint CK_DiscountVariablesDetails
        check ([DateFrom] <= [DateTo] OR [DateTo] IS NULL)
)
go
```

DiscountVariableR1

Numer zmiennej: VariableID
Numer zniżki klienta: CutomerDiscountID

```
create table DiscountVariableR1
(
    VariableID         int not null
        constraint FK_DiscountVariableR1_DiscountVariablesDetails
            references DiscountVariablesDetails,
    CustomerDiscountID int not null
        constraint FK_DiscountVariableR1_CustomerHasDiscountR1
            references CustomerHasDiscountR1,
    constraint PK_DiscountVariableR1
        primary key (VariableID, CustomerDiscountID)
)
go
```

DiscountVariableR2

Numer zmiennej: VariableID
Numer zniżki klienta: CutomerDiscountID

```
create table DiscountVariableR2
(
    VariableID         int not null
        constraint FK_DiscountVariableR2_DiscountVariablesDetails
            references DiscountVariablesDetails,
    CustomerDiscountID int not null
        constraint FK_DiscountVariableR2_CustomerHasDiscountR2
            references CustomerHasDiscountR2,
    constraint PK_DiscountVariableR2
        primary key (VariableID, CustomerDiscountID)
)
go
```

Employees

Numer pracownika: EmployeeID
Imię pracownika: Firstname
Nazwisko pracownika: Lastname
Numer konta bankowego pracownika: BankAccount
Pozycja pracownika: Position

Warunki integralnościowe:

- BankAccount - XXXXXXXXXXXXXXXXXXXXXXXXXX, gdzie X to cyfra od 0 do 9
- Position - zawiera się w (‘Waiter’, ‘Cook’ , ‘Manager’, ‘Administrator’)

```
create table Employees
(
    EmployeeID  int          not null
        constraint PK_Employees
            primary key,
    Firstname   nvarchar(50) not null,
    Lastname    nvarchar(50) not null,
    BankAccount nchar(26)    not null
        constraint CK_Employees
            check ([BankAccount] like
                   '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9
                    ][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9
                    ][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
    Position    nvarchar(50) not null
        constraint CK_Employees_1
            check ([Position] = 'Cook' OR [Position] =
             'Administrator' OR [Position] = 'Waiter' OR
                  [Position] = 'Manager')
)
go
```

IndividualCustomers

Numer klienta: CutomerID
Imię klienta: Firstname
Nazwisko klienta: Lastname

```
create table IndividualCustomers
(
    CustomerID int          not null
        constraint PK_IndividualCustomers
            primary key
        constraint FK_IndividualCustomers_Customers
            references Customers
        constraint FK_IndividualCustomers_CustomersDetails
            references CustomersDetails,
    Firstname  nvarchar(50) not null,
    Lastname   nvarchar(50) not null
)
go
```

IndividualReservation

Numer rezerwacji: ReservationID
Numer klienta: CutomerID

```
create table IndividualReservations
(
    ReservationID int not null
        constraint FK_IndividualReservations_Reservations
            references Reservations,
    CustomerID    int not null
        constraint FK_IndividualReservations_IndividualCustomers
            references IndividualCustomers,
    constraint PK_IndividualReservations
        primary key (ReservationID, CustomerID)
)
go
```

OrderDetails

Numer zamówienia: OrderID
Numer produktu: ProductID
Ilość: Quantity
Zniżka: Discount

Warunki integralnościowe:

- Quantity - większe od 0
- DiscountID - może być NULL

```
create table OrderDetails
(
    OrderID   int                                          not null
        constraint [FK_Order Details_Orders]
            references Orders,
    ProductID int                                          not null
        constraint FK_OrderDetails_Products
            references Products,
    Quantity  int                                          not null
        constraint CK_OrderDetails
            check ([Quantity] > 0),
    Discount  decimal(3, 2)
        constraint DF_OrderDetails_Discount default 0 not null,
    constraint [PK_Order Details]
        primary key (OrderID, ProductID)
)
go
```

Orders

Numer zamówienia: OrderID
Numer pracownika obsługującego: EmployeeID
Data zamówienia: OrderDate
Numer klienta: CustomerID
Zamówienie na wynos: ToGo

```
create table Orders
(
    OrderID    int  not null
        constraint PK_Orders
            primary key,
    EmployeeID int  not null
        constraint FK_Orders_Employees
            references Employees,
    OrderDate  date not null,
    CustomerID int  not null
        constraint FK_Orders_Customers
            references Customers,
    ToGo       bit  not null
)
go
```

ProductDetails

Numer produktu: ProductID
Nazwa produktu: ProductName
Numer kategorii: CategoryID

```
create table ProductDetails
(
    ProductNameID int         not null
        constraint PK_ProductDetails_1
            primary key,
    ProductName   varchar(50) not null,
    CategoryID    int         not null
        constraint FK_ProductDetails_Categories
            references Categories
)
go
```

Products

Numer produktu: ProductID
Data początku obowiązywania: DateFrom
Data końca obowiązywania: DateTo
Cena: Price

Warunki integralnościowe:

- DataTo - musi być później niż DataFrom albo DataTo jest NULL
- Price - większe bądź równe 0

```
create table Products
(
    ProductID int   not null
        constraint PK_Products_1
            primary key,
    DateFrom  date  not null,
    DateTo    date,
    Price     money not null
        constraint CK_Products_1
            check ([Price] >= 0),
    constraint CK_Products
        check ([DateTo] >= [DateFrom] OR [DateTo] IS NULL)
)
go
```

ProductIDNames

Numer produktu: ProductID
Numer nazwy produktu: ProductNameID

```
create table ProductIDToName
(
    ProductID     int not null
        constraint FK_ProductIDToName_Products
            references Products,
    ProductNameID int not null
        constraint FK_ProductIDToName_ProductDetails
            references ProductDetails,
    constraint PK_ProductIDToName
        primary key (ProductID, ProductNameID)
)
go
```

ReservationConditions

Nazwa zmiennej: VariableName
Wartość zmiennej: VariableValue

Warunki integralnościowe:

- VariableName - zawiera się w (‘WZ’, ‘WK’)
- VariableValue - większe bądź równe 0

```
create table ReservationConditions
(
    VariableName  nvarchar(3) not null
        constraint PK_Variables
            primary key
        constraint CK_ReservationConditions
            check ([VariableName] = 'WK' OR [VariableName] = 'WZ'),
    VariableValue int         not null
        constraint CK_ReservationConditions_1
            check ([VariableValue] >= 0)
)
go
```

ReservationDetails

Numer rezerwacji: ReservationID
Numer stolika: TableNO

```
create table ReservationDetails
(
    ReservationID int not null
        constraint PK_ReservationDetails
            primary key,
    TableNO       int not null
        constraint FK_TableHist_Tables
            references Tables
)
go
```

Reservations

Numer rezerwacji: ReservationID
Data rezerwacji: ReservationDate
Numer zamówienia: OrderID
Opłacenie zamówienia: IsPaid
Status zatwierdzenia zamówienia: IsComfirmed
Liczba osób na które jest zrobione zamówienie: PeopleNumber

Warunki integralnościowe:

- IsConfirmed - wartość domyślna 0

```
create table Reservations
(
    ReservationID   int                                     not null
        constraint PK_Reservations
            primary key,
    ReservationDate smalldatetime                           not null,
    OrderID         int                                     not null
        constraint FK_Reservations_Orders
            references dbo.Orders,
    IsPaid          bit                                     not null,
    IsConfirmed     bit
        constraint DF_Reservations_IsConfirmed default 0 not null,
    PeopleNumber    int                                     not null
)
go
```

Tables

Numer stolika: TableNO
Ilość siedzeń: SeatsNO

Warunki integralnościowe:

- SeatsNO - większe bądź równe 0

```
create table Tables
(
    TableNO int not null
        constraint PK_Tables
            primary key,
    SeatsNO int not null
        constraint CK_Tables
            check ([SeatsNO] >= 0)
)
go
```

Widoki

Widok CustomerDiscounts

Pokazuje zniżki wszystkich klientów

```
CREATE VIEW CustomersDiscounts
AS
SELECT CustomerID, 'R1' AS 'DiscountType', CHD.CustomerDiscountID, DVD
  .VariableValue AS 'DiscountValue',
ISNULL(CONVERT(nvarchar(20), AppliedDate), 'not applied yet') AS 'AppliedDate', '-'
   AS 'WhenUsed',
DVD.DateFrom AS 'DiscountValidFrom', DVD.DateTo AS 'DiscountValidTo'
FROM     CustomerHasDiscountR1 AS CHD INNER JOIN
                  DiscountVariableR1 AS DV ON CHD.CustomerDiscountID = DV
                    .CustomerDiscountID INNER JOIN
                  DiscountVariablesDetails AS DVD ON DVD.VariableID = DV.VariableID
UNION
SELECT CustomerID, 'R2' AS 'DiscountType', CHD.CustomerDiscountID, DVD
  .VariableValue AS 'DiscountValue',
ISNULL(CONVERT(nvarchar(20), AppliedDate), 'not applied yet') AS 'AppliedDate',
  ISNULL(CONVERT(nvarchar(20), WhenUsed),
                  'not used') AS 'WhenUsed', DVD.DateFrom AS 'DiscountValidFrom',
                   DVD.DateTo AS 'DiscountValidTo'
FROM     CustomerHasDiscountR2 AS CHD INNER JOIN
                  DiscountVariableR2 AS DV ON CHD.CustomerDiscountID = DV
                    .CustomerDiscountID INNER JOIN
                  DiscountVariablesDetails AS DVD ON DVD.VariableID = DV.VariableID
go
```

Widok CustomerWithNotPaidReservations

Pokazuje klientów z nieopłaconymi rezerwacjami oraz ile wynosi ich dług

```
CREATE VIEW CustomerWithNotPaidReservations
AS
SELECT Customers.CustomerID, SUM(GetOrderValue( @OrderID: Orders.OrderID)) AS 'Suma
 nie opłaconych zamówień'
FROM     Customers INNER JOIN
                 Orders ON Customers.CustomerID = Orders.CustomerID INNER JOIN
                 Reservations ON Orders.OrderID = Reservations.OrderID
WHERE  (Reservations.IsPaid = 0)
GROUP BY Customers.CustomerID
go
```

Widok CutomersInfo

Pokazuje informacje na temat klientów, ilość ich zamówień oraz sumę kosztu zamówień

```
CREATE VIEW CutomersInfo
AS
SELECT
    c.CustomerID, c.Email, ic.Firstname AS 'Customer Info 1', ic.Lastname AS
     'Customer Info 2', isNull(SUM(GetOrderValue( @OrderID: o.OrderID)),0) AS
     'OrdersSum', COUNT(o.OrderID) AS 'OrdersQuantity'
FROM Customers AS c
LEFT JOIN Orders AS o ON c.CustomerID = o.CustomerID
INNER JOIN IndividualCustomers AS ic ON c.CustomerID = ic.CustomerID
GROUP BY c.CustomerID, c.Email, ic.Firstname, ic.Lastname
UNION
SELECT c.CustomerID, c.Email, co.CompanyName AS 'Customer Info 1', co.NIP AS
 'Customer Info 2', isNull(SUM(GetOrderValue( @OrderID: o.OrderID)),0) AS
 'OrdersSum', COUNT(o.OrderID) AS 'OrdersQuantity'
FROM Customers AS c
LEFT JOIN Orders AS o ON c.CustomerID = o.CustomerID
INNER JOIN Companies AS co ON c.CustomerID = co.CustomerID
GROUP BY c.CustomerID, c.Email, co.CompanyName, co.NIP
go
```

Widok MealInfo

Pokazuje nazwe produktu oraz do jakiej kategorii należy

```
CREATE VIEW MealInfo
AS
SELECT ProductDetails.ProductName, Categories.CategoryName
FROM     Categories INNER JOIN
                  ProductDetails ON Categories.CategoryID = ProductDetails
                    .CategoryID
go
```

Widok MenuToday

Pokazuje dzisiejsze menu

```
CREATE VIEW MenuToday
AS
SELECT ProductDetails.ProductName, Products.Price, Categories.CategoryName
FROM     Products INNER JOIN
                  ProductIDToName ON Products.ProductID = ProductIDToName.ProductID
                   INNER JOIN
                  ProductDetails ON ProductIDToName.ProductNameID = ProductDetails
                   .ProductNameID INNER JOIN
                  Categories ON ProductDetails.CategoryID = Categories.CategoryID
WHERE  (Products.DateFrom <= GETDATE()) AND (Products.DateTo IS NULL OR
                  Products.DateTo >= GETDATE())
go
```

Widok NotPaidReservations

Pokazuje jeszcze nieopłacone rezerwacje

```
CREATE VIEW NotPaidReservations
AS
SELECT ReservationID, ReservationDate, IsPaid
FROM     Reservations
WHERE  (IsPaid = 0)
go
```

Widok OcupiedTables

Wyświetla informacje o dokładnych datach, w których stolik był zajęty przez rezerwację

```
CREATE VIEW OcupiedTables
AS
SELECT ReservationDetails.TableNO, ReservationDetails.ReservationID, Reservations
  .ReservationDate, Reservations.IsConfirmed
FROM     ReservationDetails INNER JOIN
                  Reservations ON ReservationDetails.ReservationID = Reservations
                    .ReservationID
WHERE  (Reservations.IsConfirmed = 1)
go
```

Widok OrdersWithSeafoodAll

Pokazuje wszystkie zamówienia (przyszłe i przeszłe) z owocami morza

```
CREATE VIEW OrdersWithSeafoodAll
AS
SELECT OrderDetails.OrderID, Orders.CustomerID, OrderDetails.ProductID, Orders
 .OrderDate
FROM     OrderDetails INNER JOIN
                 Products ON OrderDetails.ProductID = Products.ProductID INNER JOIN
                 ProductIDToName ON Products.ProductID = ProductIDToName.ProductID
                   INNER JOIN
                 ProductDetails ON ProductIDToName.ProductNameID = ProductDetails
                  .ProductNameID INNER JOIN
                 Categories ON ProductDetails.CategoryID = Categories.CategoryID
                  INNER JOIN
                 Orders ON OrderDetails.OrderID = Orders.OrderID
WHERE  (Categories.CategoryName LIKE 'Sea Food')
go
```

Widok OrdersWithSeafoodSoon

Pokazuje przyszłe zamówienia z owocami morza

```
CREATE VIEW OrdersWithSeafoodSoon
AS
SELECT ProductDetails.ProductName, Orders.OrderDate, OrderDetails.Quantity
FROM     Orders INNER JOIN
                  OrderDetails ON Orders.OrderID = OrderDetails.OrderID INNER JOIN
                  Products ON OrderDetails.ProductID = Products.ProductID INNER JOIN
                  ProductIDToName ON Products.ProductID = ProductIDToName.ProductID
                    INNER JOIN
                  ProductDetails ON ProductIDToName.ProductNameID = ProductDetails
                   .ProductNameID INNER JOIN
                  Categories ON ProductDetails.CategoryID = Categories.CategoryID
WHERE  (Categories.CategoryName LIKE N'Sea Food') AND (Orders.OrderDate > GETDATE())
go
```

Widok ProductsByDaysInMenu

Pokazuje ile dni produkty są już w menu i ile sztuk udało się sprzedać

```
CREATE VIEW ProductsByDaysInMenu
AS
SELECT ProductDetails.ProductName, DATEDIFF(DAY, Products.DateFrom, GETDATE()) AS
 DaysInMenu, Products.Price, ProductIDToName.ProductID, ISNULL(SUM(OrderDetails
 .Quantity), 0) AS QuantitySold
FROM     Products INNER JOIN
                 ProductIDToName ON Products.ProductID = ProductIDToName.ProductID
                   INNER JOIN
                 ProductDetails ON ProductIDToName.ProductNameID = ProductDetails
                  .ProductNameID LEFT OUTER JOIN
                 OrderDetails ON Products.ProductID = OrderDetails.ProductID
WHERE  (Products.DateFrom < GETDATE()) AND (Products.DateTo IS NULL OR
                 Products.DateTo > GETDATE())
GROUP BY ProductDetails.ProductName, DATEDIFF(DAY, Products.DateFrom, GETDATE()),
 Products.Price, ProductIDToName.ProductID
go
```

Widok ProductsOrderHistory

Pokazuje wyniki sprzedaży produktów z ich cenami i zakresami obowiązywania w menu

```
CREATE VIEW ProductsOrderHistory
AS
SELECT Products.ProductID, ProductDetails.ProductName, Products.Price, ISNULL(SUM
 (OrderDetails.Quantity), 0) AS QuantitySold, Products.DateFrom, Products.DateTo
FROM     ProductDetails INNER JOIN
                  Categories ON ProductDetails.CategoryID = Categories.CategoryID
                   INNER JOIN
                  ProductIDToName ON ProductDetails.ProductNameID = ProductIDToName
                   .ProductNameID INNER JOIN
                  Products ON ProductIDToName.ProductID = Products.ProductID INNER
                   JOIN
                  Orders INNER JOIN
                  OrderDetails ON Orders.OrderID = OrderDetails.OrderID ON
                   Products.ProductID = OrderDetails.ProductID
GROUP BY Products.ProductID, ProductDetails.ProductName, Products.Price, Products
 .DateFrom, Products.DateTo
go
```

Widok RaportsByCustomer

Pokazuje informacje o zamówieniach wszystkich klientów

```
CREATE VIEW RaportsByCustomer
AS
SELECT Orders.CustomerID, Orders.OrderID, ProductDetails.ProductName, OrderDetails
  .Quantity, Products.Price, OrderDetails.Discount, Orders.OrderDate, Orders.ToGo
FROM     Orders INNER JOIN
                  OrderDetails ON Orders.OrderID = OrderDetails.OrderID INNER JOIN
                  Products ON OrderDetails.ProductID = Products.ProductID INNER JOIN
                  ProductIDToName ON Products.ProductID = ProductIDToName.ProductID
                   INNER JOIN
                  ProductDetails ON ProductIDToName.ProductNameID = ProductDetails
                   .ProductNameID
go
```

Widok RaportsByOrder

Pokazuje dokładne informacje o każdym zamówieniu

```
CREATE VIEW RaportsByOrder
AS
SELECT Orders.OrderID, Orders.CustomerID, ROUND(SUM((Products.Price * OrderDetails
 .Quantity) * (1 - OrderDetails.Discount)), 2) AS OrderValue, Orders.ToGo, Orders
 .EmployeeID, Orders.OrderDate
FROM     Orders INNER JOIN
                 OrderDetails ON Orders.OrderID = OrderDetails.OrderID INNER JOIN
                 Products ON OrderDetails.ProductID = Products.ProductID
GROUP BY Orders.OrderID, Orders.CustomerID, Orders.EmployeeID, Orders.OrderDate,
 Orders.ToGo
go
```

Widok RaportsByProduct

Pokazuje dokładne informacje o sprzedaży danego produktu

```
CREATE VIEW RaportsByProduct
AS
SELECT Products.ProductID, ProductDetails.ProductName, ROUND(SUM((OrderDetails
 .Quantity * Products.Price) * (1 - OrderDetails.Discount)), 2) AS ProductsSoldValue,
  Products.DateFrom, Products.DateTo,
                Products.Price, SUM(OrderDetails.Quantity) AS TotalQuantitySold,
                 Categories.CategoryName
FROM     Products INNER JOIN
                OrderDetails ON Products.ProductID = OrderDetails.ProductID INNER
                 JOIN
                Orders ON OrderDetails.OrderID = Orders.OrderID INNER JOIN
                ProductIDToName ON Products.ProductID = ProductIDToName.ProductID
                 INNER JOIN
                ProductDetails ON ProductIDToName.ProductNameID = ProductDetails
                 .ProductNameID INNER JOIN
                Categories ON ProductDetails.CategoryID = Categories.CategoryID
GROUP BY Products.ProductID, ProductDetails.ProductName, Products.DateFrom,
 Products.DateTo, Products.Price, Categories.CategoryName
go
```

Widok RaportsByReservation

Pokazuje rezerwacje informacje o rezerwacjach

```
CREATE VIEW RaportsByReservation
AS
SELECT Reservations.ReservationID, Reservations.ReservationDate, ReservationDetails
 .TableNO, ROUND(SUM((OrderDetails.Quantity * Products.Price) * (1 - OrderDetails
 .Discount)), 2) AS ReservationValue
FROM     Reservations INNER JOIN
                 ReservationDetails ON Reservations.ReservationID =
                  ReservationDetails.ReservationID INNER JOIN
                 Orders ON Reservations.OrderID = Orders.OrderID INNER JOIN
                 OrderDetails ON Orders.OrderID = OrderDetails.OrderID INNER JOIN
                 Products ON OrderDetails.ProductID = Products.ProductID
GROUP BY Reservations.ReservationID, Reservations.ReservationDate, ReservationDetails
 .TableNO
go
```

Widok ReportsByTable

Pokazuje wyniki stolików

```
CREATE VIEW ReportsByTable
AS
SELECT Tables.TableNO, COUNT(Orders.OrderID) AS 'Orders Quantity', SUM
 (GetOrderValue( @OrderID: Orders.OrderID)) AS 'Orders value'
FROM     ReservationDetails INNER JOIN
                  Tables ON ReservationDetails.TableNO = Tables.TableNO INNER JOIN
                  Reservations ON Reservations.ReservationID = ReservationDetails
                   .ReservationID INNER JOIN
                  Orders ON Reservations.OrderID = Orders.OrderID
GROUP BY Tables.TableNO
go
```

Widok ReservationsToConfirm

Pokazuje rezerwacje, które nie zostały jeszcze zaakceptowane

```
CREATE VIEW ReservationsToConfirm
AS
SELECT ReservationID, ReservationDate, IsConfirmed, IsPaid, OrderID
FROM     Reservations
WHERE  (IsConfirmed = 0) AND (ReservationDate > GETDATE())
go
```

Widok TakeAwayOrders

Pokazuje zamówienia na wynos

```
CREATE VIEW TakeAwayOrders
AS
SELECT Customers.CustomerID, Orders.OrderID, Orders.OrderDate
FROM     Orders INNER JOIN
                 Customers ON Orders.CustomerID = Customers.CustomerID
WHERE  (Orders.ToGo = 1)
go
```

Funkcje

CanThisProductBeOrdered

Zwraca 1 jeżeli produkt spełnia warunki dot. owoców morza, a 0 jeżeli ich nie spełnia

```
CREATE FUNCTION CanThisProductBeOrdered(@ProductID AS INT, @OrderDate AS DATE)
RETURNS BIT AS
BEGIN
    DECLARE @CategoryID AS INT
    SET @CategoryID = (SELECT CategoryID FROM ProductIDToName AS PITN INNER JOIN
    ProductDetails AS PD ON PD.ProductNameID = PITN.ProductNameID
    WHERE ProductID = @ProductID)
    IF @CategoryID = 9
    BEGIN
        IF DATEPART(WEEKDAY, @OrderDate) = 4 OR DATEPART(WEEKDAY, @OrderDate) = 5
         OR DATEPART(WEEKDAY, @OrderDate) = 6
        BEGIN
            IF DATEADD(DAY, 7, GETDATE()) > @OrderDate
            BEGIN
                IF @OrderDate > GETDATE() AND DATEPART(WEEKDAY, GETDATE()) = 1
                BEGIN
                    RETURN 1
                END
                RETURN 0
            END
            RETURN 1
        END
        RETURN 0
    END
    RETURN 1
END
go
```

CompareMenuOnDays

Funkcja, która porównuje menu z 2 dni

```
CREATE FUNCTION CompareMenuOnDays(@day1 AS DATE, @day2 AS DATE = null)
RETURNS @MenuCompared TABLE(ProductName VARCHAR(50), Price MONEY, CategoryName VARCHAR(50),
 ProductState VARCHAR(50)) AS
BEGIN
    IF @day2 IS NULL
    BEGIN
        SET @day2 = DATEADD(WEEK, -2, @day1)
    END
    INSERT INTO @MenuCompared
    SELECT D1.ProductName, D1.Price, D1.CategoryName, 'InBothMenus' AS ProductState
    FROM SelectMenuOnDay( @day: @day1) AS D1
    INNER JOIN SelectMenuOnDay( @day: @day2) AS D2 ON D2.ProductName = D1.ProductName
    UNION
    SELECT D1.ProductName, D1.Price, D1.CategoryName, 'InFirstMenuOnly' AS ProductState
    FROM SelectMenuOnDay( @day: @day1) AS D1
    LEFT OUTER JOIN SelectMenuOnDay( @day: @day2) AS D2 ON D2.ProductName = D1.ProductName
    WHERE D2.ProductName IS NULL
    UNION
    SELECT D2.ProductName, D2.Price, D2.CategoryName, 'InSecondMenuOnly' AS ProductState
    FROM SelectMenuOnDay( @day: @day2) AS D2
    LEFT OUTER JOIN SelectMenuOnDay( @day: @day1) AS D1 ON D2.ProductName = D1.ProductName
    WHERE D1.ProductName IS NULL
    RETURN
END
go
```

CustomerDiscountsInfo

Funkcja zwracająca informacje na temat zniżek klienta

```
CREATE FUNCTION CustomerDiscountsInfo(@CustomerID AS INT)
RETURNS TABLE AS
RETURN
    SELECT DiscountType, CustomerDiscountID, DiscountValue, AppliedDate, WhenUsed
     FROM CustomersDiscounts
    WHERE CustomerID = @CustomerID
go
```

CustomerOrdersHistory

Funkcja umożliwiająca zobaczenie historii zakupów danego klienta

```
CREATE FUNCTION CustomerOrdersHistory(@customerID AS INT)
RETURNS TABLE AS
RETURN
    SELECT [OrderID]
      ,[ProductName]
      ,[Quantity]
      ,[Price]
      ,[Discount]
      ,[OrderDate]
      ,[ToGo]
    FROM [u_miczech].[dbo].[RaportsByCustomer]
    WHERE CustomerID = @customerID
go
```

CustomerOrdersGeneral

Funkcja zwracająca całkowity koszt każdego z zamówień klienta

```
CREATE FUNCTION CustomerOrdersGeneral(@CustomerID AS INT)
RETURNS TABLE AS
RETURN
    SELECT OrderID, SUM(Price*Quantity*(1-Discount)) AS 'OrderValue', OrderDate,
     ToGo
    FROM RaportsByCustomer
    WHERE CustomerID = @CustomerID
    GROUP BY OrderID, OrderDate, ToGo
go
```

GeneratePossibleReservationDates

Funkcja zwracająca możliwe terminy rezerwacji

```
CREATE FUNCTION GeneratePossibleReservationDates()
RETURNS @Dates TABLE(DateValue SMALLDATETIME)
AS
BEGIN
    DECLARE @CUR_DATE SMALLDATETIME
    SET @CUR_DATE = CAST(YEAR(GETDATE()) AS varchar) + '-' + CAST(MONTH(GETDATE())
     AS varchar) + '-' + CAST(DAY(DATEADD(DAY,1,GETDATE())) AS varchar)
    DECLARE @EndDate SMALLDATETIME
    SET @EndDate = DATEADD(WEEK, 4, GETDATE())
    WHILE @CUR_DATE <= @EndDate BEGIN
        INSERT INTO @Dates VALUES(@CUR_DATE)
        SET @CUR_DATE = DATEADD(MINUTE, 30, @CUR_DATE)
    END
    RETURN;
END;
go
```

GetDiscount1ValueForCustomer

Funkcja zwraca wysokość zniżki R1 dla danego klienta, jeżeli ją odblokował, a 0 w przeciwnym przypadku

```
CREATE FUNCTION GetDiscount1ValueForCustomer(@CustomerID INT)
RETURNS DECIMAL(6,2) AS
BEGIN
    DECLARE @Discount DECIMAL(4,2)
    SELECT @Discount = VariableValue FROM DiscountVariablesDetails INNER JOIN
    DiscountVariableR1 ON DiscountVariableR1.VariableID = DiscountVariablesDetails
     .VariableID
    INNER JOIN CustomerHasDiscountR1 ON CustomerHasDiscountR1.CustomerDiscountID =
     DiscountVariableR1.CustomerDiscountID
    WHERE CustomerID = @CustomerID AND VariableName = 'R1' AND DateTo IS NULL AND
     AppliedDate IS NOT NULL
    IF @Discount IS NULL
    BEGIN
        RETURN 0
    END
    RETURN @Discount
END
go
```

GetOrderDetails

Zwraca informacje o zamówieniu o danym ID

```
CREATE FUNCTION GetOrderDetails(@OrderID INT)
RETURNS TABLE AS
RETURN
    SELECT OrderID, CustomerID, ProductName, Quantity, Price, Discount, OrderDate, ToGo
    FROM RaportsByCustomer WHERE OrderID = @OrderID
go
```

GetOrderValue

Funkcja zwracająca wartość danego zamówienia

```
CREATE FUNCTION GetOrderValue(@OrderID INT)
    RETURNS INT
AS
BEGIN
    RETURN (
    SELECT sum(P.Price*OrderDetails.Quantity*(1-OrderDetails.Discount))
    FROM OrderDetails Inner Join Products P on P.ProductID = OrderDetails.ProductID
    WHERE OrderID = @OrderID
    GROUP BY OrderID
    )
END
go
```

IsVisiblyDifferent

Funkcja sprawdzająca czy menu na dany dzień jest poprawne (warunek zmiany co 2 tygodnie)

```
CREATE FUNCTION isVisiblyDifferent(@date AS DATE)
RETURNS bit AS
BEGIN
    DECLARE @repeatedProducts AS INT
    SET @repeatedProducts =(
    SELECT COUNT(*) FROM selectMenuOnDay( @day: @date) AS C
    INNER JOIN selectMenuOnDay( @day: DATEADD(DAY, -14, @date)) AS P ON P
     .ProductName = C.ProductName)
    DECLARE @allProducts AS INT
    SET @allProducts =(
    SELECT COUNT(*) FROM selectMenuOnDay( @day: @date))
    if @repeatedProducts*2 <= @allProducts
    BEGIN
        RETURN 1
    END
    RETURN 0
END
go
```

Raporty miesięczne i tygodniowego

Funkcje WeeklyRaportsByReservation, MonthlyRaportsByReservation, WeeklyRaportsByOrder i MonthlyRaportsByOrder umożliwiają managerowi generowanie raportów tygodniowych/miesięcznych od zadanej daty do tygodnia/miesiąca w przód z wyszczególnieniem zamówień lub rezerwacji

```
CREATE FUNCTION MonthlyRaportsByOrder(@dateFrom AS DATE)
RETURNS TABLE AS
RETURN
    SELECT * FROM RaportsByOrder
    WHERE OrderDate >= @dateFrom AND OrderDate < DATEADD(month, 1, @dateFrom)
go


CREATE FUNCTION MonthlyRaportsByReservation(@dateFrom AS DATE)
RETURNS TABLE AS
RETURN
    SELECT * FROM RaportsByReservation
    WHERE ReservationDate >= @dateFrom AND ReservationDate < DATEADD(month, 1,
     @dateFrom)
go


CREATE FUNCTION WeeklyRaportsByOrder(@dateFrom AS DATE)
RETURNS TABLE AS
RETURN
    SELECT * FROM RaportsByOrder
    WHERE OrderDate >= @dateFrom AND OrderDate < DATEADD(WEEK, 1, @dateFrom)
go


CREATE FUNCTION WeeklyRaportsByReservation(@dateFrom AS DATE)
RETURNS TABLE AS
RETURN
    SELECT * FROM RaportsByReservation
    WHERE ReservationDate >= @dateFrom AND ReservationDate < DATEADD(WEEK, 1,
     @dateFrom)
go
```

Zamówienia z owocami morza z tygodnia i miesiąca

Funkcje OrdersWithSeafoodForMonth i OrdersWithSeafoodForWeek zwracają tabele z zamówieniami z owocami morza z tygodnia i miesiąca

```
CREATE FUNCTION OrdersWithSeafoodForMonth(@dateFrom AS DATE)
RETURNS TABLE AS
RETURN
    SELECT * FROM OrdersWithSeafoodAll
    WHERE OrdersWithSeafoodAll.OrderDate >= @dateFrom AND OrdersWithSeafoodAll
     .OrderDate < DATEADD(MONTH, 1, @dateFrom)
go


CREATE FUNCTION OrdersWithSeafoodForWeek(@date AS DATE)
RETURNS TABLE AS
RETURN
    SELECT * FROM OrdersWithSeafoodAll
    WHERE OrdersWithSeafoodAll.OrderDate >= @date AND OrdersWithSeafoodAll
     .OrderDate < DATEADD(WEEK, 1, @date)
go
```

ProductsInTime

Zwraca raporty o produktach, które rozpoczęły istnienie w menu w zadanym przedziale czasowym

```
CREATE FUNCTION ProductsInTime(@dateFrom AS DATE, @dateTo AS DATE)
RETURNS TABLE AS
RETURN
    SELECT * FROM RaportsByProduct
    WHERE DateFrom >= @dateFrom AND DateFrom <= @dateTo
go
```

SelectMenuOnDay

Funkcja umożliwiająca wyświetlenie menu z dowolnego dnia

```
CREATE FUNCTION selectMenuOnDay(@day AS DATE)
RETURNS TABLE AS
RETURN
    SELECT   ProductDetails.ProductName, Products.Price, Categories.CategoryName
    FROM     Products INNER JOIN
                  ProductIDToName ON Products.ProductID = ProductIDToName.ProductID
                    INNER JOIN
                  ProductDetails ON ProductIDToName.ProductNameID = ProductDetails
                   .ProductNameID INNER JOIN
                  Categories ON ProductDetails.CategoryID = Categories.CategoryID
    WHERE    Products.DateFrom <= @day AND (Products.DateTo IS NULL OR Products
     .DateTo >= @day)
go
```

ShouldDiscount1BeApplied

Funkcja zwraca 1 jeżeli klientowi powinna zostać zniżka 1, a 0 w przeciwnym wypadku

```
CREATE FUNCTION ShouldDiscount1BeApplied(@CustomerID INT)
RETURNS BIT AS
BEGIN
    DECLARE @Z1 DECIMAL(6,2)
    DECLARE @K1 DECIMAL(6,2)
    SELECT @Z1 = VariableValue FROM DiscountVariablesDetails WHERE VariableName =
     'Z1' AND DateTo IS NULL
    SELECT @K1 = VariableValue FROM DiscountVariablesDetails WHERE VariableName =
     'K1' AND DateTo IS NULL
    DECLARE @NumberOfR1Orders INT
    SELECT @NumberOfR1Orders = COUNT(*) FROM CustomerOrdersGeneral
     ( @CustomerID: @CustomerID)
    WHERE OrderValue >= @K1
    IF @NumberOfR1Orders >= @Z1
    BEGIN
        RETURN 1
    END
    RETURN 0
END
go
```

VerifyReservation

Funkcja sprawdza czy dana rezerwacja klienta indywidualnego spełnia warunki

```
CREATE FUNCTION VerifyReservation( @CustomerID AS INT, @OrderDate AS DATE, @ProductNames AS NVARCHAR(MAX),
 @ProductQuantity AS NVARCHAR(MAX) )
RETURNS BIT AS
BEGIN
    -- gets variables values
    DECLARE @CurrentWK INT
    DECLARE @CurrentWZ INT
    SET @CurrentWK = (SELECT VariableValue FROM ReservationConditions WHERE VariableName='WK')
    SET @CurrentWZ = (SELECT VariableValue FROM ReservationConditions WHERE VariableName='WZ')

    -- array with product names
    DECLARE @productNameTable AS TABLE (ProductName NVARCHAR(255),n INT)
    INSERT INTO @productNameTable (ProductName,n)
    SELECT value,ROW_NUMBER() OVER (ORDER BY (SELECT NULL))
    FROM STRING_SPLIT(@ProductNames, ',')

    DECLARE @productNameTableLength INT
    SELECT @productNameTableLength = COUNT(*) FROM @productNameTable

    -- array with product quantity
    DECLARE @productQuantityTable AS TABLE (ProductQuantity INT,n INT)
    INSERT INTO @productQuantityTable (ProductQuantity,n)
    SELECT CONVERT(INT, value),ROW_NUMBER() OVER (ORDER BY (SELECT NULL))
    FROM STRING_SPLIT(@ProductQuantity, ',')

    DECLARE @productQuantityTableLength INT
    SELECT @productQuantityTableLength = COUNT(*) FROM @productQuantityTable

    -- calculates order value
    DECLARE @OrderValue INT
    SET @OrderValue = 0

    DECLARE @index INT = 1;
    WHILE @index <= @productNameTableLength
        BEGIN
            DECLARE @productName NVARCHAR(255);
            SELECT @productName = ProductName FROM @productNameTable WHERE n = @index;

            DECLARE @Quantity INT;
            SELECT @Quantity = ProductQuantity FROM @productQuantityTable WHERE n = @index;

            SET @OrderValue = @OrderValue + @Quantity * (
                select Products.Price
                from Products
                INNER JOIN
                          ProductIDToName ON Products.ProductID = ProductIDToName.ProductID
                INNER JOIN
                          ProductDetails ON ProductIDToName.ProductNameID = ProductDetails.ProductNameID
                where ProductDetails.ProductName LIKE @ProductName
                and
                Products.DateFrom <= @OrderDate
                and (Products.DateTo is null or @OrderDate <= Products.DateTo))

            SET @index = @index + 1;
        END;
```

```
    -- checks condition
    IF @OrderValue < @CurrentWZ
        BEGIN
            RETURN 0
        END
    DECLARE @OrdersNumber INT
    SET @OrdersNumber = (SELECT count(*) FROM CustomerOrdersGeneral( @CustomerID: @CustomerID))
    IF @OrdersNumber < @CurrentWK
        BEGIN
            RETURN 0
        END
    RETURN 1
END
go
```

Procedury

AcceptReservation

Procedura do potwierdzania zamówienia w systemie

```
CREATE PROCEDURE AcceptReservation @ReservationID int,@TableNo int
as
begin
    update Reservations
    set IsConfirmed = 1
    where ReservationID = @ReservationID

    exec AddTableToReseravtion
            @ReseravtionID =@ReservationID,
            @TableNO = @TableNo

end
go
```

AddCategory

Procedura dodająca nową kategorie do bazy

```
CREATE PROCEDURE AddCategory(@CategoryName AS NVARCHAR(50))

AS
BEGIN
    DECLARE @CategoryID INT
    SELECT @CategoryID = ISNULL(MAX(CategoryID), 0) + 1
    FROM Categories
    INSERT INTO Categories(CategoryID, CategoryName)
    VALUES (@CategoryID,@CategoryName)
end
go
```

AddCompany

Procedura dodaje informacje na temat nowej firmy to tabeli Customers, Companies i CustomersDetails

```
CREATE PROCEDURE AddCompany(@email nvarchar(50),@companyname nvarchar(150),@nip nvarchar(10))

as
begin
    DECLARE @new_primary_key INT

    SELECT @new_primary_key = MAX(CustomerID) + 1
    FROM Customers

    insert into Customers([CustomerID],[Email])
    VALUES (@new_primary_key,@email)

    exec addCustomerDetails @customerid = @new_primary_key

    insert into Companies([CustomerID],[CompanyName],[NIP])
    VALUES (@new_primary_key,@companyname,@nip)
end
go
```

AddCompanyCustomersReservation

Procedura dodająca nową rezerwację na firmę z pracownikami firmy do bazy

```
CREATE PROCEDURE AddCompanyCustomersReservation( @ReservationID AS INT = NULL, @ReservationDate AS SMALLDATETIME,
 @CustomerIDs AS NVARCHAR(MAX), @IsPaid AS BIT, @EmployeeID AS INT, @OrderDate AS DATE, @CustomerID AS INT,
 @ProductNames AS NVARCHAR(MAX), @ProductQuantity AS NVARCHAR(MAX))
AS
BEGIN
    -- Tablica IDs klientów
    DECLARE @CustomerIDsTable AS TABLE (CustomerID INT,n INT)
    INSERT INTO @CustomerIDsTable (CustomerID, n)
    SELECT CONVERT(INT, value), ROW_NUMBER() OVER (ORDER BY (SELECT NULL))
    FROM STRING_SPLIT(@CustomerIDs, ',')
    DECLARE @PeopleNumber INT
    SELECT @PeopleNumber = COUNT(*) FROM @CustomerIDsTable

    -- Sprawdź id pracowników
    DECLARE @index INT = 1;
    WHILE @index <= @PeopleNumber
        BEGIN
            DECLARE @CurrentCustomerID INT;
            SELECT @CurrentCustomerID = CustomerID FROM @CustomerIDsTable WHERE n = @index;

            IF NOT @CurrentCustomerID IN (SELECT CustomerID FROM IndividualCustomers)
                BEGIN;
                RAISERROR (N'Błędny ID pracownika firmy!', 10, 1)
                RETURN
              END

            SET @index = @index + 1;
        END;

    -- Sprawdź czy liczba klientów rezerwacji >= 2
    IF @PeopleNumber < 2
      BEGIN;
        RAISERROR (N'Rezerwacja musi być na minimum dwie osoby!', 10, 1)
        RETURN
      END

    -- Nadaj id rezerwacji jeśli null oraz id zamówienia
    IF @ReservationID IS NULL
    BEGIN
        SELECT @ReservationID = ISNULL(MAX(ReservationID),0) + 1
        FROM Reservations
    END
    DECLARE @OrderID AS INT
    SELECT @OrderID = ISNULL(MAX(OrderID),0) + 1
    FROM Orders
```

```
    -- Dodaj zamówienie do bazy
    EXEC AddOrder
         @OrderID: @OrderID,
         @EmployeeID: @EmployeeID,
         @Discount: @OrderDate,
         @OrderDate: @CustomerID,
         @CustomerID: 0,
         @ToGo: @ProductNames,
         @ProductNames: @ProductQuantity

    -- Dodaj do bazy rezerwacje jako firma i jako członkowie firmy `
    INSERT INTO Reservations(ReservationID, ReservationDate, OrderID, IsPaid, IsConfirmed, PeopleNumber)
    VALUES (@ReservationID, @ReservationDate, @OrderID, @IsPaid, 0, @PeopleNumber)
    INSERT INTO CompanyReservations(ReservationID, CustomerID)
    VALUES (@ReservationID, @CustomerID)

    SET @index = 1;
    WHILE @index <= @PeopleNumber
        BEGIN
            SELECT @CurrentCustomerID = CustomerID FROM @CustomerIDsTable WHERE n = @index;

            INSERT INTO IndividualReservations(ReservationID, CustomerID)
            VALUES (@ReservationID, @CurrentCustomerID)

            SET @index = @index + 1;
        END;
END
go
```

AddCompanyReservation

Procedura dodająca nową rezerwację klienta firmowego do bazy

```
CREATE PROCEDURE AddCompanyReservation( @ReservationID AS INT = NULL, @ReservationDate AS SMALLDATETIME, @IsPaid AS
 BIT, @EmployeeID AS INT, @OrderDate AS DATE, @CustomerID AS INT, @PeopleNumber AS INT, @ProductNames AS NVARCHAR(MAX)
 , @ProductQuantity AS NVARCHAR(MAX))
AS
BEGIN
    -- Sprawdź czy liczba klientów rezerwacji >= 2
    IF @PeopleNumber < 2
      BEGIN;
        RAISERROR (N'Rezerwacja musi być na minimum dwie osoby!', 10, 1)
        RETURN
      END

    -- Nadaj id rezerwacji jeśli null oraz id zamówienia
    IF @ReservationID IS NULL
    BEGIN
        SELECT @ReservationID = ISNULL(MAX(ReservationID),0) + 1
        FROM Reservations
    END
    DECLARE @OrderID AS INT
    SELECT @OrderID = ISNULL(MAX(OrderID),0) + 1
    FROM Orders

    -- Dodaj zamówienie do bazy
    EXEC AddOrder
         @OrderID: @OrderID,
         @EmployeeID: @EmployeeID,
         @Discount: @OrderDate,
         @OrderDate: @CustomerID,
         @CustomerID: 0,
         @ToGo: @ProductNames,
         @ProductNames: @ProductQuantity

    -- Dodaj rezerwacje do bazy
    INSERT INTO Reservations(ReservationID, ReservationDate, OrderID, IsPaid, IsConfirmed, PeopleNumber)
    VALUES (@ReservationID, @ReservationDate, @OrderID, @IsPaid, 0, @PeopleNumber)
    INSERT INTO CompanyReservations(ReservationID, CustomerID)
    VALUES (@ReservationID, @CustomerID)
END
go
```

AddCustomerDetails

Procedura dodaje lub modyfikuje informacje dla podanego klienta (używane przez AddCompany albo AddClientIndivdual)

```
CREATE PROCEDURE AddCustomerDetails(@customerid int, @street nvarchar(50)=null, @buildingNO
 nvarchar(10)=null, @ZIP nvarchar(6)=null, @appertmentNO nvarchar(10)=null)

AS
BEGIN
    IF EXISTS (SELECT * FROM CustomersDetails WHERE Customerid = @customerid)
    BEGIN
        UPDATE CustomersDetails
        SET street = ISNULL(@street, street), BuildingNO = ISNULL(@buildingNO, BuildingNO),
         ZIP = ISNULL(@ZIP, ZIP), ApartmentNO = ISNULL(@appertmentNO, ApartmentNO)
        WHERE CustomerID = @customerid
    END
    ELSE
    BEGIN
        INSERT INTO CustomersDetails ([CustomerID], [Street], [BuildingNO], [ZIP],
         [ApartmentNO])
        VALUES (@customerid, @street, @buildingNO, @ZIP, @appertmentNO)
    END
END
go
```

AddCustomerIndvidual

Procedura dodaje informacje na temat nowego klienta indywidualnego do tabeli Customers, IndvidualCustomers i CustomersDetails

```
CREATE PROCEDURE AddCutstomerIndvidual(@email nvarchar(50),@firstname nvarchar(50),@lastname
 nvarchar(50))

AS
BEGIN

    DECLARE @new_primary_key INT

    SELECT @new_primary_key = MAX(CustomerID) + 1
    FROM Customers

    insert into Customers([CustomerID],[Email])
    VALUES (@new_primary_key,@email)


    exec addCustomerDetails @customerid = @new_primary_key
    insert into IndividualCustomers([CustomerID],[Firstname],[Lastname])
    values (@new_primary_key,@firstname,@lastname)
end
go
```

AddDiscountToOrder

Procedura dodaje zniżkę do danego zamówienia

```
CREATE PROCEDURE AddDiscountToOrder(@OrderID int,@Discount AS DECIMAL(3, 2) = 0)
as
begin
    update OrderDetails
    set Discount = @Discount
    where OrderID = @OrderID
end
go

CREATE PROCEDURE AddEmployee(@Firstname AS NVARCHAR(50), @Lastname AS NVARCHAR(50),
 @BankAccount AS NCHAR(26), @Position AS NVARCHAR(50))
AS
BEGIN
    DECLARE @EmployeeID INT
    SELECT @EmployeeID = ISNULL(MAX(EmployeeID), 0) + 1
    FROM Employees
    INSERT INTO Employees(EmployeeID, Firstname, Lastname, BankAccount, Position)
    VALUES (@EmployeeID, @Firstname, @Lastname, @BankAccount, @Position)
end
go
```

AddEmployee

Procedura dodająca nowego pracownika do bazy

```
CREATE PROCEDURE AddEmployee(@Firstname AS NVARCHAR(50), @Lastname AS NVARCHAR(50),
 @BankAccount AS NCHAR(26), @Position AS NVARCHAR(50))
AS
BEGIN
    DECLARE @EmployeeID INT
    SELECT @EmployeeID = ISNULL(MAX(EmployeeID), 0) + 1
    FROM dbo.Employees
    INSERT INTO dbo.Employees(EmployeeID, Firstname, Lastname, BankAccount, Position)
    VALUES (@EmployeeID, @Firstname, @Lastname, @BankAccount, @Position)
end
go
```

AddIndividualReservation

Procedura dodająca nową rezerwację klienta indywidualnego do bazy

```
CREATE PROCEDURE AddIndividualReservation( @ReservationID AS INT = NULL, @ReservationDate AS SMALLDATETIME, @IsPaid AS
  BIT, @PeopleNumber AS INT, @EmployeeID AS INT, @OrderDate AS DATE, @CustomerID AS INT, @Discount AS DECIMAL(3, 2) =
 0, @ProductNames AS NVARCHAR(MAX), @ProductQuantity AS NVARCHAR(MAX))
AS
BEGIN
    -- Sprawdź czy liczba klientów rezerwacji >= 2
    IF @PeopleNumber < 2
      BEGIN;
        RAISERROR (N'Rezerwacja musi być na minimum dwie osoby!', 10, 1)
        RETURN
      END

    -- Zweryfikuj rezerwacje
    DECLARE @VerificationFlag AS BIT
    SELECT @VerificationFlag = VerifyReservation( @CustomerID: @CustomerID,  @OrderDate: @OrderDate,
      @ProductNames: @ProductNames,  @ProductQuantity: @ProductQuantity)
    IF @VerificationFlag = 0
      BEGIN;
        RAISERROR (N'Klient nie spełnia warunków rezerwacji', 10, 1)
        RETURN
      END

    -- Nadaj id rezerwacji jeśli null oraz id zamówienia
    IF @ReservationID IS NULL
    BEGIN
        SELECT @ReservationID = ISNULL(MAX(ReservationID),0) + 1 FROM Reservations
    END
    DECLARE @OrderID AS INT
    SELECT @OrderID = ISNULL(MAX(OrderID),0) + 1 FROM Orders

    -- Dodaj zamówienie do bazy
    EXEC AddOrder
            @OrderID =@OrderID,
            @EmployeeID =@EmployeeID,
            @Discount = @Discount,
            @OrderDate =@OrderDate,
            @CustomerID= @CustomerID,
            @ProductNames =@ProductNames,
            @ProductQuantity =@ProductQuantity

    -- Dodaj rezerwacje do bazy
    INSERT INTO Reservations(ReservationID, ReservationDate, OrderID, IsPaid, IsConfirmed, PeopleNumber)
    VALUES (@ReservationID, @ReservationDate, @OrderID, @IsPaid, 0, @PeopleNumber)
    INSERT INTO IndividualReservations(ReservationID, CustomerID)
    VALUES (@ReservationID, @CustomerID)
END
go
```

AddNewProductDetails

Procedura dodaje nowe danie do tabeli ProductDetails

```
CREATE PROCEDURE AddNewProductDetails(@categoryname nvarchar(50),@productname varchar(50))
as
begin
    DECLARE @newCategoryID INT

    select @newCategoryID = categoryid
    from Categories
    where CategoryName like @categoryname

    DECLARE @newProductNameID int
    select @newProductNameID = max(ProductNameID)+1
    from ProductDetails

    insert into ProductDetails([ProductNameID],[ProductName],[CategoryID])
    VALUES(@newProductNameID,@productname,@newCategoryID)

end
go
```

AddOrder

Procedura dodająca nowe zamówienie do bazy

```
CREATE PROCEDURE AddOrder(
    @OrderID int = null,
    @EmployeeID INT,
    @Discount AS DECIMAL(3, 2) = 0,
    @OrderDate DATE,
    @CustomerID INT,
    @ToGo BIT=0,
    @ProductNames NVARCHAR(MAX),
    @ProductQuantity NVARCHAR(MAX))
as
BEGIN
    IF @OrderID is null
    begin
        SELECT @OrderID = ISNULL(MAX(OrderID),0) + 1
        FROM Orders
    end

    DECLARE @productNameTable AS TABLE (ProductName NVARCHAR(255),n INT)
    INSERT INTO @productNameTable (ProductName,n)
    SELECT value,ROW_NUMBER() OVER (ORDER BY (SELECT NULL))
    FROM STRING_SPLIT(@ProductNames, ',')

    DECLARE @productQuantityTable AS TABLE (ProductQuantity INT,n INT)
    INSERT INTO @productQuantityTable (ProductQuantity,n)
    SELECT CONVERT(INT, value),ROW_NUMBER() OVER (ORDER BY (SELECT NULL))
    FROM STRING_SPLIT(@ProductQuantity, ',')

    DECLARE @productNameArrayLength INT
    SELECT @productNameArrayLength = COUNT(*) FROM @productNameTable

    DECLARE @productQuantityArrayLength INT
    SELECT @productQuantityArrayLength = COUNT(*) FROM @productQuantityTable

    BEGIN TRY
    -- Sprawdź czy obie zmienne są tej samej długości
    IF @productNameArrayLength != @productQuantityArrayLength
      BEGIN;
        THROW 50000, N'Zmienne ProductNames i ProductQuantity muszą mieć tę samą liczbę elementów!', 1;
      END

    DECLARE @index INT = 1;

    INSERT INTO Orders(OrderID, EmployeeID, OrderDate, CustomerID, ToGo)
    VALUES (@OrderID, @EmployeeID, @OrderDate, @CustomerID, @ToGo)
```

```
    WHILE @index <= @productNameArrayLength
        BEGIN
            DECLARE @productName NVARCHAR(255);
            SELECT @productName = ProductName FROM @productNameTable WHERE n = @index;

            SELECT @productQuantity = ProductQuantity FROM @productQuantityTable WHERE n = @index;

            EXEC AddProductToOrder @OrderId = @OrderID,
                                   @ProductName = @productName,
                                   @OrderDate = @OrderDate,
                                   @Quantity = @productQuantity,
                                   @Discount = @Discount;
            SET @index = @index + 1;
        END;
    END TRY
    BEGIN CATCH
    PRINT ERROR_MESSAGE();
  END CATCH
end
go
```

AddProductToOrder

Procedura dodająca produkt do istniejącego zamówienia w bazie

```
CREATE PROCEDURE AddProductToOrder(
@OrderId  INT,
@ProductName NVARCHAR(255),
@OrderDate DATE,
@Quantity AS INT,
@Discount AS DECIMAL(3, 2) = 0)
AS
BEGIN
    BEGIN TRY
        IF NOT EXISTS(SELECT * FROM SelectMenuOnDay( @day: @orderDate))
        BEGIN;
            THROW 50000, N'Nie ma takiej potrawy w menu na ten dzień', 1;
        END

        declare @ProductID int
        select @ProductID = Products.ProductID
        from Products
        INNER JOIN
                  ProductIDToName ON Products.ProductID = ProductIDToName.ProductID
        INNER JOIN
                  ProductDetails ON ProductIDToName.ProductNameID = ProductDetails
                    .ProductNameID
        where ProductDetails.ProductName LIKE @ProductName
        and
        Products.DateFrom <= @OrderDate
        and (Products.DateTo is null or @OrderDate <= Products.DateTo)


        INSERT INTO OrderDetails(OrderID, ProductID, Quantity, Discount)
        VALUES (@OrderID, @ProductID, @Quantity, @Discount)

    END TRY
    BEGIN CATCH
        Print ERROR_MESSAGE();
    END Catch
end
go
```

AddTableToReseravtion

Procedura dodaje stolik do rezerwacji

```
CREATE PROCEDURE AddTableToReseravtion
@ReseravtionID int,
@TableNO int
as
begin
    INSERT INTO ReservationDetails (ReservationID, TableNo)
    VALUES (@ReseravtionID, @TableNO)

end
go
```

AddToMenu

Procedura przyjmuje nazwe produktu, od kiedy będzie sprzedawany oraz jego cene i dodaje do tabeli Products i ProductIDToName

```
CREATE PROCEDURE AddToMenu(@productname varchar(50),@startdate date,@price money)
as
begin
    Declare @newProductID int
    select @newProductID = max(ProductID) + 1
    from ProductIDToName

    Declare @ProductNameID int
    select @ProductNameID = ProductNameID
    from ProductDetails
    where @productname = ProductName


    insert into Products([ProductID],[DateFrom],[DateTo],[Price])
    values (@newproductID,@startdate,null,@price)

    insert into ProductIDToName([ProductID],[ProductNameID])
    values (@newProductID,@ProductNameID)


end
go
```

AllFreeTables

Tabela wszystkich wolnych stolików z datami na najbliższe 4 tygodnie

```
CREATE PROCEDURE AllFreeTables
AS
BEGIN
    Declare @T Table (TableNO int, ImpossibleDate smalldatetime)
    Insert @T Exec ImpossibleReservationDates
        SELECT Ta.TableNO, Ta.SeatsNO, DateValue AS ReservationPossible FROM Tables AS Ta
        CROSS JOIN GeneratePossibleReservationDates() AS Re
        LEFT OUTER JOIN @T AS T2 ON T2.TableNO = Ta.TableNO AND Re.DateValue = T2
          .ImpossibleDate
        WHERE T2.ImpossibleDate IS NULL
END
go
```

CancelReservation

Procedura anuluje konkretną rezerwacje

```
create procedure CancelReservation @ReservationID int
as
begin
    update Reservations
    set IsConfirmed = null
    where ReservationID = @ReservationID

end
go
```

EndProduct

Procedura ustawia date zakończanie sprzedaży produtku

```
CREATE PROCEDURE EndProduct(@productID int,@enddate date)
as
begin
    update Products
    set DateTo = @enddate
    where ProductID = @productID
end
go
```

FreeTablesOn

Tabela stolików wolnych o zadanej dacie

```
CREATE PROCEDURE FreeTablesOn @Date SMALLDATETIME AS
BEGIN
    Declare @T Table (TableNO int, ImpossibleDate smalldatetime)
    Insert @T Exec ImpossibleReservationDates
        SELECT Ta.TableNO, Ta.SeatsNO, DateValue AS ReservationPossible FROM Tables AS Ta
        CROSS JOIN GeneratePossibleReservationDates() AS Re
        LEFT OUTER JOIN @T AS T2 ON T2.TableNO = Ta.TableNO AND Re.DateValue = T2
          .ImpossibleDate
        WHERE T2.ImpossibleDate IS NULL AND Re.DateValue = @Date

END
go
```

ImpossibleReservationDates

Tabela wszystkich dat, gdzie nie można dokonać rezerwacji

```
CREATE PROCEDURE ImpossibleReservationDates
AS
BEGIN
    DECLARE @Dates TABLE(TableNO int, ImpossibleDate smalldatetime)
    SELECT ReservationID, TableNO, ReservationDate
    INTO #ControlTable
    FROM OcupiedTables

    DECLARE @RowID int
    DECLARE @TableNO int
    DECLARE @ReservationDate smalldatetime
    DECLARE @BeginDate smalldatetime
    DECLARE @EndDate smalldatetime

    WHILE EXISTS(SELECT * FROM #ControlTable)
    BEGIN
        SELECT TOP 1 @RowID = ReservationID, @TableNO = TableNO, @ReservationDate =
         ReservationDate
        FROM #ControlTable

        SET @BeginDate = DATEADD(HOUR, -2, @ReservationDate)
        SET @EndDate = DATEADD(HOUR, 2, @ReservationDate)
        WHILE @BeginDate <= @EndDate
        BEGIN
            INSERT @Dates values(@TableNO, @BeginDate)
            SET @BeginDate = DATEADD(MINUTE, 30, @BeginDate)
        END

        DELETE #ControlTable
        WHERE ReservationID = @RowID
    END
    DROP table #ControlTable
    SELECT * FROM @Dates
END
go
```

RemoveIndividualReservation

Procedura usuwająca daną rezerwację klienta indywidualnego z bazy

```
CREATE PROCEDURE RemoveIndividualReservation(@ReservationID INT)
AS
BEGIN
    DELETE FROM IndividualReservations
    WHERE ReservationID = @ReservationID
    DELETE FROM Reservations
    WHERE ReservationID = @ReservationID
    DELETE FROM ReservationDetails
    WHERE ReservationID = @ReservationID
END
go
```

SetDiscountVariablesDetails

Procedura przyjmuje wartość danej zmiennej oraz datę początku obowiązywania tej wartości

```
CREATE PROCEDURE SetDiscoutnVariablesDetails(@start_date date,@variablevalue decimal(4,2),
 @variablename nvarchar(10))
as
begin
    update DiscountVariablesDetails
    set DateTo = DATEADD(day,-1,@start_date)
    where dateTo is null and @variablename = VariableName

    DECLARE @new_primary_key INT

    SELECT @new_primary_key = MAX(VariableID) + 1
    FROM DiscountVariablesDetails

    insert into DiscountVariablesDetails([VariableID],[DateFrom],[DateTo],[VariableValue],
     [VariableName])
    values (@new_primary_key,@start_date,null,@variablevalue,@variablename)

end
go
```

SetReservationConditions

Procedura zmienia wartość podanej nazwy zmiennej związanej z warunkami rezerwacji

```
CREATE PROCEDURE SetReservationConditions(@variablevalue int,@variablename nvarchar(3))
as
begin
    update ReservationConditions
    set VariableValue = @variablevalue
    where VariableName = @variablename
end
go
```

Triggery

Trigger ApplyDiscountIfNeeded

Trigger sprawdza czy klient kwalifikuje się do zniżki pierwszej lub drugiej (tylko klienci indywidualni), a jeżeli tak, to przyznaje ją klientowi. Uruchamia się po dodaniu nowego elementu do zamówienia

```
CREATE TRIGGER ApplyDiscountsIfNeeded
   ON  OrderDetails
   AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @OrderID INT
    SELECT @OrderID = inserted.OrderID FROM inserted
    DECLARE @CustomerID INT
    SELECT @CustomerID = Orders.customerID FROM Orders WHERE Orders.OrderID = @OrderID
    IF EXISTS (SELECT CustomerID FROM Companies WHERE CustomerID = @CustomerID)
    BEGIN
        RETURN
    END
    DECLARE @DiscountID INT
    DECLARE @AppliedDate DATE
    SELECT TOP 1 @DiscountID = CustomerHasDiscountR1.CustomerDiscountID, @AppliedDate = CustomerHasDiscountR1
     .AppliedDate
    FROM CustomerHasDiscountR1 WHERE CustomerID = @CustomerID ORDER BY CustomerDiscountID DESC
    IF @AppliedDate IS NULL
    BEGIN
        DECLARE @Z1 DECIMAL(6,2)
        DECLARE @K1 DECIMAL(6,2)
        SELECT @Z1 = VariableValue FROM DiscountVariablesDetails WHERE VariableName = 'Z1' AND DateTo IS NULL
        SELECT @K1 = VariableValue FROM DiscountVariablesDetails WHERE VariableName = 'K1' AND DateTo IS NULL
        DECLARE @NumberOfR1Orders INT
        SELECT @NumberOfR1Orders = COUNT(*) FROM CustomerOrdersGeneral( @CustomerID: @CustomerID)
        WHERE OrderValue >= @K1
        IF @NumberOfR1Orders >= @Z1
        BEGIN
            UPDATE CustomerHasDiscountR1
            SET AppliedDate = GETDATE()
            WHERE CustomerDiscountID = @DiscountID
        END
    END
    SELECT TOP 1 @DiscountID = CustomerHasDiscountR2.CustomerDiscountID, @AppliedDate = CustomerHasDiscountR2
     .AppliedDate
    FROM CustomerHasDiscountR2 WHERE CustomerID = @CustomerID ORDER BY CustomerDiscountID DESC
    IF @AppliedDate IS NULL
    BEGIN
        DECLARE @K2 DECIMAL(6,2)
        SELECT @K2 = VariableValue FROM DiscountVariablesDetails WHERE VariableName = 'K2' AND DateTo IS NULL
        DECLARE @SumaricOrderValue MONEY
        SELECT @SumaricOrderValue = SUM(OrderValue) FROM CustomerOrdersGeneral( @CustomerID: @CustomerID)
        IF @SumaricOrderValue > @K2
        BEGIN
            UPDATE CustomerHasDiscountR2
            SET AppliedDate = GETDATE()
            WHERE CustomerDiscountID = @DiscountID
        END
    END
END
go
```

Trigger DeleteOrderAndReservationIfCancelled

Trigger usuwa dane zamówienia i reze z tabeli OrderDetails, jeżeli powiązana z nim rezerwacja została anulowana.

```
CREATE TRIGGER DeleteOrderAndReservationIfCancelled
   ON  Reservations
   AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @OrderIDToDelete INT
    SELECT @OrderIDToDelete = (SELECT O.OrderID FROM Orders O INNER JOIN Reservations R on O.OrderID =
      R.OrderID WHERE R.IsConfirmed is null)

    DECLARE @ReservationIDToDelete INT
    SELECT @ReservationIDToDelete = (SELECT R.ReservationID FROM Reservations R WHERE R.IsConfirmed is
      null)

    DELETE FROM OrderDetails
    WHERE OrderDetails.OrderID = @OrderIDToDelete

    DELETE FROM IndividualReservations
    WHERE IndividualReservations.ReservationID = @ReservationIDToDelete

    DELETE FROM CompanyReservations
    WHERE CompanyReservations.ReservationID = @ReservationIDToDelete

    DELETE FROM Reservations
    WHERE Reservations.ReservationID = @ReservationIDToDelete

    DELETE FROM Orders
    WHERE Orders.OrderID = @OrderIDToDelete

END
go
```

Trigger DiscountsModified

Dodaje nowe ID zniżek przy zmianie wartości parametrów K1, Z1, R1, R2 i K2. Jeżeli klientowi przy nowych parametrach przysługuje zniżka 1 - przyznaje ją.

```
CREATE TRIGGER DiscountsModified ON DiscountVariablesDetails
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @VarName NVARCHAR(10)
    DECLARE @VarValue DECIMAL(6,2)
    DECLARE @VarID INT
    SELECT  @VarName = inserted.VariableName, @VarID = inserted.VariableID, @VarValue = inserted.VariableValue FROM
     inserted
    DECLARE @CustomersIDs TABLE(CustomerID INT)
    DECLARE @CurrCustomerID INT
    DECLARE @NewDiscountID INT
    DECLARE @AppliedDate DATE
    IF @VarName = 'K1' OR @VarName = 'Z1' OR @VarName = 'R1'
    BEGIN
        INSERT INTO @CustomersIDs SELECT CustomerID FROM IndividualCustomers
        IF @VarName != 'R1'
        BEGIN
            SELECT @VarID=VariableID FROM DiscountVariablesDetails WHERE VariableName = 'R1' AND DateTo IS NULL
        END
        WHILE EXISTS (SELECT * FROM @CustomersIDs)
        BEGIN
            SET @CurrCustomerID = (SELECT TOP 1 CustomerID FROM @CustomersIDs)
            DELETE FROM @CustomersIDs WHERE CustomerID = @CurrCustomerID
            SELECT @NewDiscountID = MAX(CustomerDiscountID)+1 FROM CustomerHasDiscountR1
            SET @AppliedDate = NULL
            IF dbo.ShouldDiscount1BeApplied( @CustomerID: @CurrCustomerID) = 1
            BEGIN
                SET @AppliedDate = GETDATE()
            END
            INSERT INTO CustomerHasDiscountR1 VALUES(@NewDiscountID, @AppliedDate, @CurrCustomerID)
            INSERT INTO DiscountVariableR1 VALUES(@VarID, @NewDiscountID)
        END
    END
    ELSE IF @VarName = 'K2' OR @VarName = 'R2'
    BEGIN
        INSERT INTO @CustomersIDs SELECT CustomerID FROM IndividualCustomers
        IF @VarName != 'R2'
        BEGIN
            SELECT @VarID=VariableID FROM DiscountVariablesDetails WHERE VariableName = 'R2' AND DateTo IS NULL
        END
        WHILE EXISTS (SELECT * FROM @CustomersIDs)
        BEGIN
            SET @CurrCustomerID = (SELECT TOP 1 CustomerID FROM @CustomersIDs)
            DELETE FROM @CustomersIDs WHERE CustomerID = @CurrCustomerID
            SELECT @NewDiscountID = MAX(CustomerDiscountID)+1 FROM CustomerHasDiscountR2
            INSERT INTO CustomerHasDiscountR2 VALUES(@NewDiscountID, NULL, @CurrCustomerID, NULL)
            INSERT INTO DiscountVariableR2 VALUES(@VarID, @NewDiscountID)
        END
    END
END
go
```

Trigger ProperDiscountVar

Trigger sprawdza czy dodane nowe zmienne R1 lub R2 są z zakresu 0-1.

```
CREATE TRIGGER ProperDiscountVariable
   ON DiscountVariablesDetails
   AFTER INSERT
AS
BEGIN
    IF ((SELECT inserted.VariableValue FROM inserted) NOT BETWEEN 0 AND 1) AND ((SELECT inserted
      .VariableName FROM inserted) = 'R1')
    BEGIN
        RAISERROR(N'Wprowadzona zmienna R1 jest nieprawidłowa', 16, 1)
        ROLLBACK TRAN
    END
    ELSE IF ((SELECT inserted.VariableValue FROM inserted) NOT BETWEEN 0 AND 1) AND ((SELECT
      inserted.VariableName FROM inserted) = 'R2')
    BEGIN
        RAISERROR(N'Wprowadzona zmienna R2 jest nieprawidłowa', 16, 1)
        ROLLBACK TRAN
    END
END
go
```

Trigger ProperReservationConditions

Trigger sprawdza czy zmieniona wartość w tabeli ReservationConditions jest wieksza od 0

```
CREATE TRIGGER ProperReservationConditions
ON ReservationConditions
AFTER UPDATE
AS
BEGIN
    IF EXISTS (SELECT 1 FROM inserted WHERE VariableValue < 0)
    BEGIN
        RAISERROR (N'Zmienna nie może być mniejsza od 0', 16, 1);
        ROLLBACK TRAN;
    END
END;
go
```

Trigger SeaFoodCheckMonday

Trigger blokuje zamówienie owoców morza gdy zamówienie nie jest składane na odpowiedni dzień lub gdy jest składane za późno

```
CREATE TRIGGER SeaFoodCheckMonday
   ON  OrderDetails
   AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @WeekDiff INT
    SET DATEFIRST 2
    IF EXISTS(
    select * from inserted as I
    INNER JOIN Orders as O ON O.OrderID = I.OrderID
    INNER JOIN Reservations as R ON O.OrderID = I.OrderID
    INNER JOIN Products as P ON P.ProductID = I.ProductID
    INNER JOIN ProductIDToName as PN ON PN.ProductID = P.ProductID
    INNER JOIN ProductDetails as PD ON PD.ProductNameID = PN.ProductNameID
    INNER JOIN Categories as C ON PD.CategoryID = C.CategoryID
    where C.CategoryName = 'Sea Food'
    and (DATENAME(WEEKDAY, R.ReservationDate) NOT IN ('Thursday', 'Friday',
     'Saturday')
    OR DATEPART(WEEK, R.ReservationDate) = DATEPART(WEEK, O.OrderDate)))
    BEGIN;
        THROW 50000, N'Takie zamówienie z owcami morza nie może być złożone na ten
         termin',1
    END
END
go
```

Indeksy

Indeks PK_Categories

Ustawienie indeksu na CategoryID w tabeli Category.

```
create unique clustered index PK_Categories
    on dbo.Categories (CategoryID)
go
```

Indeks IX_Categories

Ustawienie indeksu na CategoryID w tabeli Category.

```
create unique index IX_Categories
    on dbo.Categories (CategoryID)
go
```

Indeks PK_Companies

Ustawienie indeksu na CustomerID w tabeli Companies.

```
create unique clustered index PK_Companies
    on dbo.Companies (CustomerID)
go
```

Indeks IX_NIP

Ustawienie indeksu na NIP w tabeli Companies.

```
create unique index IX_NIP
    on dbo.Companies (NIP)
go
```

Indeks IX_CompanyName

Ustawienie indeksu na CompanyName w tabeli Companies.

```
create unique index IX_CompanyName
    on dbo.Companies (CompanyName)
go
```

Indeks PK_CompanyReservations

Ustawienie indeksu na ReservationID w tabeli CompanyReservations.

```
create unique clustered index PK_CompanyReservations
    on dbo.CompanyReservations (ReservationID)
go
```

Indeks PK_CustomerHasDiscountR1

Ustawienie indeksu na CustomerDiscountID w tabeli CustomerHasDiscountR1.

```
create unique clustered index PK_CustomerHasDiscountR1
    on dbo.CustomerHasDiscountR1 (CustomerDiscountID)
go
```

Indeks PK_CustomerHasDiscountR2

Ustawienie indeksu na CustomerDiscountID w tabeli CustomerHasDiscountR2.

```
create unique clustered index PK_CustomerHasDiscountR2
    on dbo.CustomerHasDiscountR2 (CustomerDiscountID)
go
```

Indeks PK_Customers

Ustawienie indeksu na CustomerID w tabeli Customers.

```
create unique clustered index PK_Customers
    on dbo.Customers (CustomerID)
go
```

Indeks IX_Email

Ustawienie indeksu na Email w tabeli Customers.

```
create unique index IX_Email
    on dbo.Customers (Email)
go
```

Indeks PK_CustomersDetails

Ustawienie indeksu na CustomerID w tabeli CustomersDetails.

```
create unique clustered index PK_CustomersDetails
    on dbo.CustomersDetails (CustomerID)
go
```

Indeks PK_DiscountVariableR1

Ustawienie indeksów na VariableID i CustomerDiscountID w tabeli DiscountVariableR1.

```
create unique clustered index PK_DiscountVariableR1
    on dbo.DiscountVariableR1 (VariableID, CustomerDiscountID)
go
```

Indeks PK_DiscountVariableR2

Ustawienie indeksów na VariableID i CustomerDiscountID w tabeli DiscountVariableR2.

```
create unique clustered index PK_DiscountVariableR2
    on dbo.DiscountVariableR2 (VariableID, CustomerDiscountID)
go
```

Indeks PK_DiscountVariableNames

Ustawienie indeksu na VariableID w tabeli DiscountVariablesDetails.

```
create unique clustered index PK_DiscountVariableNames
    on dbo.DiscountVariablesDetails (VariableID)
go
```

Indeks PK_Employees

Ustawienie indeksu na EmployeeID w tabeli Employees.

```
create unique clustered index PK_Employees
    on dbo.Employees (EmployeeID)
go
```

Indeks PK_IndividualCustomers

Ustawienie indeksu na CustomerID w tabeli IndividualCustomers.

```
create unique clustered index PK_IndividualCustomers
    on dbo.IndividualCustomers (CustomerID)
go
```

Indeks PK_IndividualReservations

Ustawienie indeksów na ReservationID i CustomerID w tabeli IndividualReservations.

```
create unique clustered index PK_IndividualReservations
    on dbo.IndividualReservations (ReservationID, CustomerID)
go
```

Indeks PK_OrderDetails

Ustawienie indeksu na OrderID i ProductID w tabeli OrderDetails.

```
create unique clustered index PK_OrderDetails
    on dbo.OrderDetails (OrderID, ProductID)
go
```

Indeks PK_Orders

Ustawienie indeksu na OrderID w tabeli Orders.

```
create unique clustered index PK_Orders
    on dbo.Orders (OrderID)
go
```

Indeks PK_ProductDetails_1

Ustawienie indeksu na ProductNameID w tabeli ProductDetails.

```
create unique clustered index PK_ProductDetails_1
    on dbo.ProductDetails (ProductNameID)
go
```

Indeks PK_ProductIDToName

Ustawienie indeksów na ProductID i ProductNameID w tabeli ProductIDToName.

```
create unique clustered index PK_ProductIDToName
    on dbo.ProductIDToName (ProductID, ProductNameID)
go
```

Indeks PK_Products_1

Ustawienie indeksu na ProductID w tabeli Products.

```
create unique clustered index PK_Products_1
    on dbo.Products (ProductID)
go
```

Indeks PK_Variables

Ustawienie indeksu na VariableName w tabeli ReservationConditions.

```
create unique clustered index PK_Variables
    on dbo.ReservationConditions (VariableName)
go
```

Indeks PK_ReservationDetails

Ustawienie indeksu na ReservationID w tabeli ReservationDetails.

```
create unique clustered index PK_ReservationDetails
    on dbo.ReservationDetails (ReservationID)
go
```

Indeks PK_Reservations

Ustawienie indeksu na ReservationID w tabeli Reservations.

```
create unique clustered index PK_Reservations
    on dbo.Reservations (ReservationID)
go
```

Indeks PK_Tables

Ustawienie indeksu na TableNO w tabeli Tables.

```
create unique clustered index PK_Tables
    on dbo.Tables (TableNO)
go
```

Indeks IDX_ProductDetails_ProductName

Ustawienie indeksu na ProductName w tabeli ProductDetails

```
create index IDX_ProductDetails_ProductName
    on ProductDetails(ProductName)
go
```

Indeks IDX_Products_Price

Ustawienie indeksu na Price w tabeli Products

```
create index IDX_Products_Price
    on Products(Price)
go
```

Indeks IDX_Companies_CompanyName

Ustawienie indeksu na CompanyName w tabeli Companies

```
create index IDX_Companies_CompanyName
    on Companies(CompanyName)
go
```

Indeks IDX_Categories_CategoryName

Ustawienie indeksu na CategoryName w tabeli Categories

```
create index IDX_Categories_CategoryName
    on Categories(CategoryName)
go
```

Indeks IDX_IndividualCustomers_Firstname

Ustawienie indeksu na Firstname w tabeli IndividualCustomers

```
create index IDX_IndividualCustomers_Firstname
    on IndividualCustomers(Firstname)
go
```

Indeks IDX_IndividualCustomers_Lastname

Ustawienie indeksu na Lastname w tabeli IndividualCustomers

```
create index IDX_IndividualCustomers_Lastname
    on IndividualCustomers(Lastname)
go
```

Indeks IDX_Products_DateFrom

Ustawienie indeksu na DateFrom w tabeli Products

```
create index IDX_Products_DateFrom
    on Products(DateFrom)
go
```

Indeks IDX_Products_DateTo

Ustawienie indeksu na DateTo w tabeli Products

```
create index IDX_Products_DateTo
    on Products(DateTo)
go
```

Indeks IDX_Orders_OrderDate

Ustawienie indeksu na OrderDate w tabeli Orders

```
create index IDX_Orders_OrderDate
    on Orders(OrderDate)
go
```

Indeks IDX_Reservations_ReservationDate

Ustawienie indeksu na ReservationDate w tabeli Reservations

```
create index IDX_Reservations_ReservationDate
    on Reservations(ReservationDate)
go
```

Role

Klient no-name (NoNameCustomer)

Kilent no-name czyli taki nie zarejestrowany w bazie może jedynie sprawdzić aktualne menu

```
CREATE ROLE NoNameCustomer

GRANT SELECT ON [dbo].[MenuToday] TO [NoNameCustomer]
```

Klient indywidualny (IndivdualCustomer)

Klient indywidualny może sprawdzić aktualne menu, składać rezerwacje oraz wygenerować statystyki swoich zamówień

```
CREATE ROLE IndividualCustomer

GRANT EXECUTE ON [dbo].[AddOrder] TO [IndividualCustomer]
GRANT EXECUTE ON [dbo].[AddIndividualReservation] TO [IndividualCustomer]

GRANT SELECT ON [dbo].[MenuToday] TO [IndividualCustomer]
GRANT SELECT ON [dbo].[CustomerOrderHistory] TO [IndividualCustomer]
GRANT SELECT ON [dbo].[CustomerOrdersGeneral] TO [IndividualCustomer]
GRANT SELECT ON [dbo].[CustomerOrdersGeneral] TO [IndividualCustomer]
GRANT SELECT ON [dbo].[CustomerDiscountsInfo] TO [IndividualCustomer]
```

Klient firmowy (CompanyCustomer)

Klient firmowy może sprawdzić aktualne menu,składać rezerwacje na firme lub indywidualnie na pracowników firmy oraz wygenerować statystyki swoich zamówień

```
CREATE ROLE CompanyCustomer

GRANT EXECUTE ON [dbo].[AddCompanyCustomersReservation] TO [CompanyCustomer]
GRANT EXECUTE ON [dbo].[AddOrder] TO [CompanyCustomer]
GRANT EXECUTE ON [dbo].[AddCompanyReservation] TO [CompanyCustomer]

GRANT SELECT ON [dbo].[CustomerOrderHistory] TO [CompanyCustomer]
GRANT SELECT ON [dbo].[CustomerOrdersGeneral] TO [CompanyCustomer]
GRANT SELECT ON [dbo].[MenuToday] TO [CompanyCustomer]
```

Pracownik - kelner (RegularEmployee)

Pracownik może odczytywać aktualne menu, akceptować rezerwacje, przydzielać stoliki, sprawdzać zamówienia z owcami morza lub zamówienia na wynos

```
CREATE ROLE RegularEmployee

GRANT EXECUTE ON [dbo].[AddOrder] TO [RegularEmployee]
GRANT EXECUTE ON [dbo].[AddProductToOrder] TO [RegularEmployee]
GRANT EXECUTE ON [dbo].[AcceptReservation] TO [RegularEmployee]
GRANT EXECUTE ON [dbo].[FreeTablesOn] TO [RegularEmployee]
GRANT EXECUTE ON [dbo].[GetOrderValue] TO [RegularEmployee]

GRANT SELECT ON [dbo].[SelectMenuOnDay] TO [RegularEmployee]
GRANT SELECT ON [dbo].[MenuToday] TO [RegularEmployee]
GRANT SELECT ON [dbo].[SelectMenuOnDay] TO [RegularEmployee]
GRANT SELECT ON [dbo].[ReservationsToConfirm] TO [RegularEmployee]
GRANT SELECT ON [dbo].[OrdersWithSeafoodSoon] TO [RegularEmployee]
GRANT SELECT ON [dbo].[NotPaidReservations] TO [RegularEmployee]
GRANT SELECT ON [dbo].[TakeAwayOrders] TO [RegularEmployee]
```

Manager restauracji (RestaurantManager)

Manager ma dostęp do tych samych opcji co pracownik, może również dodawać nowe produkty do bazy danych, zmieniać menu oraz ustawiać wartości związane z rezerwacjami i zniżkami dla klientów, ma dostęp do generowanie raportów związanych ze statystykami menu, rezerwacji, stolików, rabatów oraz zamówień

```
CREATE ROLE RestaurantManager

GRANT EXECUTE ON [dbo].[AcceptReservation] TO [RestaurantManager]
GRANT EXECUTE ON [dbo].[AddCategory] TO [RestaurantManager]
GRANT EXECUTE ON [dbo].[AddCompany] TO [RestaurantManager]
GRANT EXECUTE ON [dbo].[AddCompanyCustomersReservation] TO [RestaurantManager]
GRANT EXECUTE ON [dbo].[AddCompanyReservation] TO [RestaurantManager]
GRANT EXECUTE ON [dbo].[AddCustomerDetails] TO [RestaurantManager]
GRANT EXECUTE ON [dbo].[AddCustomerIndvidual] TO [RestaurantManager]
GRANT EXECUTE ON [dbo].[AddEmployee] TO [RestaurantManager]
GRANT EXECUTE ON [dbo].[AddIndividualReservation] TO [RestaurantManager]
GRANT EXECUTE ON [dbo].[AddNewProductDetails] TO [RestaurantManager]
GRANT EXECUTE ON [dbo].[AddOrder] TO [RestaurantManager]
GRANT EXECUTE ON [dbo].[AddProductToOrder] TO [RestaurantManager]
GRANT EXECUTE ON [dbo].[AddReservation] TO [RestaurantManager]
GRANT EXECUTE ON [dbo].[AddTableToReseravtion] TO [RestaurantManager]
GRANT EXECUTE ON [dbo].[AddToMenu] TO [RestaurantManager]
GRANT EXECUTE ON [dbo].[AllFreeTables] TO [RestaurantManager]
GRANT EXECUTE ON [dbo].[CanThisProductBeOrdered] TO [RestaurantManager]
GRANT EXECUTE ON [dbo].[EndProduct] TO [RestaurantManager]
GRANT EXECUTE ON [dbo].[FreeTablesOn] TO [RestaurantManager]
GRANT EXECUTE ON [dbo].[GetOrderValue] TO [RestaurantManager]
GRANT EXECUTE ON [dbo].[ImpossibleReservationDates] TO [RestaurantManager]
GRANT EXECUTE ON [dbo].[IsVisiblyDifferent] TO [RestaurantManager]
GRANT EXECUTE ON [dbo].[RemoveIndividualReservation] TO [RestaurantManager]
GRANT EXECUTE ON [dbo].[SetDiscountVariablesDetails] TO [RestaurantManager]
GRANT EXECUTE ON [dbo].[SetReservationConditions] TO [RestaurantManager]
GRANT EXECUTE ON [dbo].[VerifyReservation] TO [RestaurantManager]
```

```
GRANT SELECT ON [dbo].[Categories] TO [RestaurantManager]
GRANT SELECT ON [dbo].[Companies] TO [RestaurantManager]
GRANT SELECT ON [dbo].[CompanyReservations] TO [RestaurantManager]
GRANT SELECT ON [dbo].[CustomerDiscountsInfo] TO [RestaurantManager]
GRANT SELECT ON [dbo].[CustomerHasDiscountR1] TO [RestaurantManager]
GRANT SELECT ON [dbo].[CustomerHasDiscountR2] TO [RestaurantManager]
GRANT SELECT ON [dbo].[CustomerOrderHistory] TO [RestaurantManager]
GRANT SELECT ON [dbo].[CustomerOrdersGeneral] TO [RestaurantManager]
GRANT SELECT ON [dbo].[Customers] TO [RestaurantManager]
GRANT SELECT ON [dbo].[CustomersDetails] TO [RestaurantManager]
GRANT SELECT ON [dbo].[CustomersDiscounts] TO [RestaurantManager]
GRANT SELECT ON [dbo].[DiscountVariableR1] TO [RestaurantManager]
GRANT SELECT ON [dbo].[DiscountVariableR2] TO [RestaurantManager]
GRANT SELECT ON [dbo].[DiscountVariablesDetails] TO [RestaurantManager]
GRANT SELECT ON [dbo].[Employees] TO [RestaurantManager]
GRANT SELECT ON [dbo].[IndividualCustomers] TO [RestaurantManager]
GRANT SELECT ON [dbo].[IndividualReservations] TO [RestaurantManager]
GRANT SELECT ON [dbo].[MenuToday] TO [RestaurantManager]
GRANT SELECT ON [dbo].[MonthlyRaportsByOrder] TO [RestaurantManager]
GRANT SELECT ON [dbo].[MonthlyRaportsByReservation] TO [RestaurantManager]
GRANT SELECT ON [dbo].[NotPaidReservations] TO [RestaurantManager]
GRANT SELECT ON [dbo].[OcupiedTables] TO [RestaurantManager]
GRANT SELECT ON [dbo].[OrderDetails] TO [RestaurantManager]
GRANT SELECT ON [dbo].[Orders] TO [RestaurantManager]
GRANT SELECT ON [dbo].[OrdersWithSeafoodAll] TO [RestaurantManager]
GRANT SELECT ON [dbo].[OrdersWithSeafoodForMonth] TO [RestaurantManager]
GRANT SELECT ON [dbo].[OrdersWithSeafoodForWeek] TO [RestaurantManager]
GRANT SELECT ON [dbo].[OrdersWithSeafoodSoon] TO [RestaurantManager]
GRANT SELECT ON [dbo].[ProductDetails] TO [RestaurantManager]
GRANT SELECT ON [dbo].[ProductIDToName] TO [RestaurantManager]
GRANT SELECT ON [dbo].[Products] TO [RestaurantManager]
GRANT SELECT ON [dbo].[ProductsByDaysInMenu] TO [RestaurantManager]
GRANT SELECT ON [dbo].[ProductsInTime] TO [RestaurantManager]
GRANT SELECT ON [dbo].[ProductsOrderHistory] TO [RestaurantManager]
GRANT SELECT ON [dbo].[RaportsByCustomer] TO [RestaurantManager]
GRANT SELECT ON [dbo].[RaportsByOrder] TO [RestaurantManager]
GRANT SELECT ON [dbo].[RaportsByProduct] TO [RestaurantManager]
GRANT SELECT ON [dbo].[RaportsByReservation] TO [RestaurantManager]
GRANT SELECT ON [dbo].[ReportsByTable] TO [RestaurantManager]
GRANT SELECT ON [dbo].[ReservationConditions] TO [RestaurantManager]
GRANT SELECT ON [dbo].[ReservationDetails] TO [RestaurantManager]
GRANT SELECT ON [dbo].[Reservations] TO [RestaurantManager]
GRANT SELECT ON [dbo].[ReservationsToConfirm] TO [RestaurantManager]
GRANT SELECT ON [dbo].[SelectMenuOnDay] TO [RestaurantManager]
GRANT SELECT ON [dbo].[Tables] TO [RestaurantManager]
GRANT SELECT ON [dbo].[TakeAwayOrders] TO [RestaurantManager]
GRANT SELECT ON [dbo].[WeeklyRaportsByOrder] TO [RestaurantManager]
GRANT SELECT ON [dbo].[WeeklyRaportsByReservation] TO [RestaurantManager]
```

Administrator (Administrator)

Administrator ma dostęp do wszystkich funkcji systemowych

```
CREATE ROLE Administrator

GRANT ALL PRIVILEGES TO [Administrator]
```