Akademia Górniczo-Hutnicza
Wydział Informatyki, Elektroniki i Telekomunikacji
Podstawy Baz Danych
dr inż. Leszek Siwik

System wspomagania działalności firmy świadczącej usługi gastronomiczne dla klientów indywidualnych oraz firm

Zofia Grodecka
Kseniya Fiodarava

Spis treści

Opis problemu

Projekt dotyczy systemu wspomagania działalności firmy świadczącej usługi gastronomiczne dla klientów indywidualnych oraz firm. System winien być możliwy do równoległego użytkowania przez kilka tego typu firm.

Ogólne informacje

W ofercie jest żywność (np. ciastka, lunch, drobne przekąski) oraz napoje bezalkoholowe (np. kawa, koktajle, woda). Usługi świadczone są na miejscu oraz na wynos. Zamówienie na wynos może być zlecone na miejscu lub z wyprzedzeniem (z wykorzystaniem formularza WWW i wyboru preferowanej daty i godziny odbioru zamówienia).

Firma dysponuje ograniczoną liczbą stolików, w tym miejsc siedzących. Istnieje możliwość wcześniejszej rezerwacji stolika dla co najmniej dwóch osób. Z uwagi na zmieniające się ograniczenia związane z COVID-19, w poszczególnych dniach może być dostępna ograniczona liczba miejsc (w odniesieniu do powierzchni lokalu), zmienna w czasie.

Klientami są osoby indywidualne oraz firmy, odbierające większe ilości posiłków w porze lunchu lub jako catering (bez dostawy). Dla firm istnieje możliwość wystawienia faktury dla danego zamówienia lub faktury zbiorczej raz na miesiąc.

Menu

Menu ustalane jest co najmniej dziennym wyprzedzeniem. W firmie panuje zasada, że co najmniej połowa pozycji menu zmieniana jest co najmniej raz na dwa tygodnie, przy czym pozycja zdjęta może powtórzyć się nie wcześniej niż za 1 miesiąc.

Ponadto, w dniach czwartek-piątek-sobota istnieje możliwość wcześniejszego zamówienia dań zawierających owoce morza. Z uwagi na indywidualny import takie zamówienie winno być złożone co maksymalniem do poniedziałku poprzedzającego zamówienie. Istnieje możliwość, że pozycja w menu zostanie usunięta na skutek wyczerpania się półproduktów.

Wcześniejsza rezerwacja zamówienia/stolika

Internetowy formularz umożliwia klientowi indywidualnemu rezerwację stolika, przy jednoczesnym złożeniu zamówienia, z opcją płatności przed lub po zamówieniu, przy minimalnej wartości zamówienia 50 zł, w przypadku klientów, którzy dokonali wcześniej co najmniej 5 zamówień i/lub mniej, ale w tym przypadku na kwotę co najmniej 200 zł. Informacja wraz z potwierdzeniem zamówienia oraz wskazaniem stolika. Wysyłana jest po akceptacji przez obsługę.

Internetowy formularz umożliwia także rezerwację stolików dla firm, w dwóch opcjach: rezerwacji stolików na firmę i/lub rezerwację stolików dla konkretnych pracowników firmy (imiennie).

Rabaty

System umożliwia realizację programów rabatowych dla klientów indywidualnych:

- Po realizacji ustalonej liczby zamówień Z1 (przykładowo Z1 = 10) za co najmniej określoną kwotę K1 (np. K1 = 30 zł każde zamówienie): R1% (np. R1 = 3%) zniżki na wszystkie zamówienia
- Po realizacji kolejnych Z1 zamówień (np. Z1 = 10) za co najmniej określoną kwotę K1 każde (np. K1 = 30 zł): dodatkowe R1% zniżki na wszystkie zamówienia (np. R1 = 3%)
- Po realizacji zamówień za łączną kwotę K2 (np. 1000 zł): jednorazowa zniżka R2% (np. 5%) na zamówienia złożone przez D1 dni (np. D1 = 7), począwszy od dnia przyznania zniżki
- Po realizacji zamówień za łączną kwotę K3 (np. K3 = 5000 zł): jednorazowa zniżka R3% (np. R3 = 5%) na zamówienia złożone przez D2 dni (np. D2 = 7), począwszy od dnia przyznania zniżki.

W przypadku firm rabat udzielany jest zgodnie z zasadami:

- Za każdy kolejny miesiąc, w którym dokonano co najmniej FZ zamówień (np. FZ = 5) za łączną kwotę co najmniej FK1 (np. FK1 = 500 zł): rabat FR1% (np. FR1 = 0,1%). W przypadku braku ciągłości w zamówieniach rabat zeruje się. Łączny, maksymalny rabat, to FM% (np. FM = 4%).
- Za każdy kolejny kwartał, w którym dokonano zamówień za łączną kwotę FK2 (np. FK2 = 10000 zł): rabat kwotowy równy FR2% (np. FR2 = 5%) z łącznej kwoty, z którą zrealizowano zamówienie.

Raporty

System umożliwia 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.

System umożliwia także generowanie raportów dotyczących zamówień oraz rabatów dla klienta indywidualnego oraz firm.

Analiza wymagań

Aktorzy

- Pracownicy firmy gastronomicznej
    - kierownik zamówień
    - kierownik magazynu
    - szef kuchni
    - księgowa
    - administrator bazy danych
- Klienci
    - indywidualni
    - firmowi

User stories

Klienci

- Jako klient chciałbym mieć dostęp do informacji o cenach danych posiłków w danym dniu, aby móc wybrać najlepszy dla mnie posiłek.
- Jako klient chciałbym mieć dostęp do informacji o ilości dostępnych miejsc w danym dniu, aby wiedzieć, ile osób mogę zaprosić na posiłek.
- Jako klient chciałbym mieć możliwość złożenia zamówienia na miejscu oraz zjedzenia go na miejscu, aby nie stracić na jakości jedzenia podczas jego dowozu na miejsce.
- Jako klient chciałbym mieć możliwość złożenia zamówienia na miejscu wraz z możliwością odebrania go na wynos, aby ograniczyć kontakty międzyludzkie w związku z pandemią.
- Jako klient chciałbym mieć możliwość złożenia zamówienia na wynos z wyprzedzeniem, wykorzystując formularz internetowy, aby całkowicie uniknąć spotkań międzyludzkich w związku z pandemią.
- Jako klient chciałbym mieć możliwość wyboru preferowanej daty i godziny odbioru zamówienia w formularzu internetowym, aby móc zaplanować czas posiłku.
- Jako klient chciałbym mieć możliwość dokonania rezerwacji stolika dla co najmniej dwóch osób, aby mieć zagwarantowany stolik na miejscu.
- Jako klient chciałbym mieć możliwość wcześniejszego anulowania rezerwacji stolika, w razie gdybym musiał odwołać spotkanie.
- Jako klient indywidualny chciałbym mieć dostęp do informacji o przysługujących mi rabatach, aby nie stracić szansy na zaoszczędzenie pieniędzy.
- Jako klient firmowy chciałbym mieć możliwość zamówienia większej ilości posiłków w porze lunchu, lub jako catering, aby zapewnić obiad pracownikom mojej firmy w środku dnia pracy.
- Jako klient firmowy chciałbym mieć możliwość otrzymania faktury dla danego zamówienia, lub faktury zbiorczej raz na miesiąc, aby mieć potwierdzenie dokonania zakupu.

Pracownicy

Kierownik zamówień

- Jako kierownik zamówień chciałbym mieć możliwość przyjmowania od klientów zamówień, w tym również przyjmowania przedwczesnych zamówień dań zawierających owoce morza, aby prawidłowo wykonywać moją pracę.
- Jako kierownik zamówień chciałbym mieć możliwość dodawania, usuwania oraz anulowania rezerwacji stolików, aby sprostać oczekiwaniom klientów.
- Jako kierownik zamówień chciałbym mieć dostęp do informacji o ilości dostępnych miejsc w danym dniu, aby umożliwić klientom dokonanie rezerwacji.
- Jako kierownik zamówień chciałbym mieć możliwość zmiany liczby dostępnych do rezerwacji miejsc, po dokonaniu, lub anulowaniu przez klienta rezerwacji.

Kierownik magazynu

- Jako kierownik magazynu chciałbym mieć możliwość zapisywania i aktualizacji stanu magazynu, aby być na bieżąco z ilością dostępnych produktów.
- Jako kierownik magazynu chciałbym mieć możliwość usunięcia pozycji z menu na skutek wyczerpania zapasów produktów, aby nie oszukiwać klientów o dostępności danych dań.

Szef kuchni

- Jako szef kuchni chciałbym mieć możliwość edycji menu, aby ustalać go z co najmniej dziennym wyprzedzeniem, przestrzegając panujących w firmie zasad.
- Jako szef kuchni chciałbym mieć możliwość edycji cen potraw w menu, aby ustalać je zgodnie z kosztem produktów oraz nakładem pracy na przygotowanie potrawy.

Księgowa

- Jako księgowa chciałabym mieć ma dostęp do wystawionych faktur i paragonów, aby móc wydawać je klientom.
- Jako księgowa chciałabym mieć możliwość edycji rabatów, aby móc zwiększać jego wartość dla poszczególnych klientów.

Administrator bazy danych

- Jako administrator bazy danych chciałbym mieć dostęp do wszystkich tabel, procedur oraz widoków, aby móc wprowadzać zmiany w razie takiej potrzeby.
- Jako administrator bazy danych chciałbym móc dodawać i usuwać klientów z bazy danych oraz edytować ich dane, aby być na bieżąco z aktualnym stanem klientów.
- Jako administrator bazy danych chciałbym móc dodawać i usuwać pracowników z bazy danych oraz edytować ich dane, aby być na bieżąco z aktualnym stanem pracowników.
- Jako administrator bazy danych chciałbym mieć dostęp do danych o rezerwacjach stolików, rabatach i menu, aby móc generować raporty miesięczne i tygodniowe.
- Jako administrator bazy danych chciałbym mieć dostęp do danych o kwotach oraz czasach składania zamówienia, aby móc generować statystyki zamówień dla klientów indywidualnych oraz firm.

Opis bazy danych

Tabele

- Klienci (*Customers*)
- Klienci indywidualni (*IndividualCustomers*)
- Klienci firmowi (*CompanyCustomers*)
- Pracownicy (*Employees*)
- Zamówienia (*Orders*)
- Szczegóły zamówień *(OrderDetails)*
- Rezerwacje stolików (*Reservations*)
- Stoliki *(Tables)*
- Produkty (*Products*)
- Kategorie produktów (*ProductCategories*)
- Dania (*Dishes*)
- Szczegóły dań *(DishDetails)*
- Menu (*Menu*)
- Dania z menu *(MenuDishes)*
- Rabaty (*Discounts)*
- Zamówienia z rabatem *(OrderDiscounts)*

Opis tabel

Customers

Tabela przechowuje wspólne informacje dla klientów indywidualnych oraz firmowych, takie jak:

- ID klienta *(CustomerID)*
  - integer
  - nie może być puste
  - klucz główny
- Adres *(Address)*
  - łańcuch znaków o długości 50
  - może być puste
- Miasto *(City)*
  - łańcuch znaków o długości 30
  - może być puste
- Kod pocztowy *(PostalCode)*
  - łańcuch znaków o długości 30
  - może być puste
- Kraj *(Country)*
  - łańcuch znaków o długości 30
  - może być puste
- Numer telefonu *(Phone)*
  - łańcuch znaków o długości 20
  - nie może być puste

- e-mail *(Email)*
    - łańcuch znaków o długości 30
    - nie może być puste
- Flaga określająca, czy klient jest firmowy (*IsCompany)*
    - bit
    - nie może być puste

```
IF OBJECT_ID('Customers', 'U') IS NOT NULL
    DROP TABLE Customers
CREATE TABLE Customers(
    CustomerID INT NOT NULL PRIMARY KEY IDENTITY(1, 1),
    Address VARCHAR(50),
    City VARCHAR(30),
    PostalCode VARCHAR(30),
    Country VARCHAR(30),
    Phone VARCHAR(20) NOT NULL,
    Email VARCHAR(30) NOT NULL,
    IsCompany BIT NOT NULL
)
```

IndividualCustomers

Tabela przechowuje podstawowe informacje o klientach indywidualnych. Każdy klient posiada dane:

- ID klienta *(CustomerID)*
    - integer
    - nie może być puste
    - klucz główny
    - klucz obcy
- Nazwisko (*LastName*)
    - łańcuch znaków o długości 30
    - nie może być puste
- Imię (*FirstName)*
    - łańcuch znaków o długości 30
    - nie może być puste

```
IF OBJECT_ID('IndividualCustomers', 'U') IS NOT NULL
    DROP TABLE IndividualCustomers
CREATE TABLE IndividualCustomers(
    CustomerID INT NOT NULL PRIMARY KEY,
    FirstName VARCHAR(30) NOT NULL,
    LastName VARCHAR(30) NOT NULL,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
)
```

CompanyCustomers

Tabela przechowuje podstawowe informacje o klientach firmowych. Każdy klient posiada dane:

- ID klienta *(CustomerID)*
    - integer
    - nie może być puste
    - klucz główny
    - klucz obcy
- Nazwa firmy (*CompanyName)*
    - łańcuch znaków o długości 50
    - nie może być puste
    - wartość unikalna
- Nazwa kontaktowa *(ContactPersonName)*
    - łańcuch znaków o długości 50
    - może być puste
- Tytuł *(ContactPersonTitle)*
    - łańcuch znaków o długości 50
    - może być puste
- NIP
    - łańcuch znaków o długości 10
    - nie może być puste
    - unikalne

```
IF OBJECT_ID('CompanyCustomers', 'U') IS NOT NULL
   DROP TABLE CompanyCustomers
CREATE TABLE CompanyCustomers(
   CustomerID INT NOT NULL PRIMARY KEY FOREIGN KEY REFERENCES
Customers(CustomerID),
   CompanyName VARCHAR(50) NOT NULL UNIQUE,
   ContactPersonName VARCHAR(50),
   ContactPersonTitle VARCHAR(50),
   NIP VARCHAR(10) NOT NULL UNIQUE
)
```

Discounts

Tabela przechowuje informacje o zniżkach przysługujących klientom, takie jak:

- ID rabatu (*DiscountID*)
    - integer
    - nie może być puste
    - klucz główny
- ID klienta (*CustomerID*)
    - integer
    - nie może być puste
    - klucz obcy
- Wartość rabatu (*Value)*
    - float
    - należy do zakresu wartości od 0 do 1
    - nie może być puste
- Termin ważności (*DueDate*)
    - data

    - może być puste
- Data wystawienia rabatu (*IssueDate*)
    - data
    - nie może być puste
- Flaga określająca, czy rabat jest jednorazowy (*IsOneTime*)
    - bit
    - nie może być puste
    - wartość domyślna: 1
- Flaga określająca, czy rabat jest dostępny (*IsAvailable*)
    - bit
    - nie może być puste
    - wartość domyślna: 1

```
IF OBJECT_ID('Discounts', 'U') IS NOT NULL
   DROP TABLE Discounts
CREATE TABLE Discounts(
   DiscountID INT NOT NULL PRIMARY KEY IDENTITY(1, 1),
   CustomerID INT NOT NULL FOREIGN KEY REFERENCES Customers(CustomerID),
   Value FLOAT NOT NULL CHECK (Value > 0 AND Value <= 1),
   DueDate DATETIME,
   IssueDate DATETIME NOT NULL,
   IsOneTime BIT NOT NULL DEFAULT 1,
   IsAvailable BIT NOT NULL DEFAULT 1
)
```

OrderDiscounts

Tabela łącznikowa realizująca relację wiele do wielu pomiędzy tabelami `Orders` i `Discounts`. Zawiera następujące informacje:

- ID rabatu (*DiscountID*)
    - integer
    - może być puste
    - klucz obcy
- ID zamówienia (*OrderID)*
    - integer
    - nie może być puste
    - klucz obcy

```
IF OBJECT_ID('OrderDiscounts', 'U') IS NOT NULL
    DROP TABLE OrderDiscounts
CREATE TABLE OrderDiscounts(
    DiscountID INT FOREIGN KEY REFERENCES Discounts(DiscountID),
    OrderID INT NOT NULL FOREIGN KEY REFERENCES Orders(OrderID)
)
```

Employees

Tabela przechowuje informacje o pracownikach firmy. Każdy pracownik posiada dane:

- ID pracownika *(EmployeeID)*
  - integer
  - nie może być puste
  - klucz główny
- Nazwisko (*LastName*)
  - łańcuch znaków o długości 30
  - nie może być puste
- Imię (*FirstName)*
  - łańcuch znaków o długości 30
  - nie może być puste
- Tytuł *(Title)*
  - łańcuch znaków o długości 30
  - nie może być puste
- Data zatrudnienia *(HireDate)*
  - data
  - może być puste
- Adres *(Address)*
  - łańcuch znaków o długości 50
  - może być puste
- Miasto *(City)*
  - łańcuch znaków o długości 30
  - może być puste
- Kod pocztowy *(PostalCode)*
  - łańcuch znaków o długości 30
  - może być puste
- Kraj *(Country)*
  - łańcuch znaków o długości 30
  - może być puste
- Numer telefonu *(Phone)*
  - łańcuch znaków o długości 20
  - nie może być puste
- e-mail *(Email)*
  - łańcuch znaków o długości 30
  - nie może być puste

```
IF OBJECT_ID('Employees', 'U') IS NOT NULL
    DROP TABLE Employees
CREATE TABLE Employees(
    EmployeeID INT NOT NULL PRIMARY KEY IDENTITY(1, 1),
    FirstName VARCHAR(30) NOT NULL,
    LastName VARCHAR(30) NOT NULL,
    Title VARCHAR(30) NOT NULL,
    HireDate DATETIME,
```

```
    Address VARCHAR(50),
    City VARCHAR(30),
    PostalCode VARCHAR(30),
    Country VARCHAR(30),
    Phone VARCHAR(20) NOT NULL,
    Email VARCHAR(30) NOT NULL
)
```

Orders

Tabela przechowuje informacje o zamówieniach. Każde zamówienie posiada dane:

- ID zamówienia *(OrderID)*
  - integer
  - nie może być puste
  - klucz główny
- ID pracownika, który przyjął zamówienie *(EmployeeID)*
  - integer
  - nie może być puste
  - klucz obcy
- ID klienta, który złożył zamówienie *(CustomerID)*
  - integer
  - nie może być puste
  - klucz obcy
- Datę złożenia zamówienia *(OrderDate)*
  - datetime
  - nie może być puste
  - ma wartość mniejszą niż wymagana data realizacji zamówienia
- Datę realizacji zamówienia *(RequiredRealizationDate)*
  - datetime
  - nie może być nullem
  - ma wartość większą niż data złożenia zamówienia
  - ma wartość mniejszą niż data odbioru zamówienia
- Datę odbioru zamówienia *(PickUpDate)*
  - datetime
  - może być puste - jeśli zamówienie jeszcze nie zostało odebrane
  - ma wartość większą niż wymagana data realizacji zamówienia
- Flagę określającą, czy zamówienie ma być zrealizowane na wynos *(IsTakeAway)*
  - bit
  - nie może być puste
  - wartość domyślna: 1

```
IF OBJECT_ID('Orders', 'U') IS NOT NULL
 DROP TABLE Orders
CREATE TABLE Orders (
    OrderID INT NOT NULL PRIMARY KEY IDENTITY(1, 1),
    EmployeeID INT NOT NULL FOREIGN KEY REFERENCES Employees(EmployeeID),
    CustomerID INT NOT NULL FOREIGN KEY REFERENCES Customers(CustomerID),
```

```
    OrderDate DATETIME NOT NULL,
    RequiredRealisationDate DATETIME NOT NULL,
    PickUpDate DATETIME,
    IsTakeAway BIT NOT NULL DEFAULT 1,
    CHECK(OrderDate < RequiredRealisationDate),
    CHECK(RequiredRealisationDate < PickUpDate)
)
```

OrderDetails

Tabela przechowuje informacje dotyczące szczegółów zamówień.

- ID zamówienia *(OrderID)*
  - integer
  - nie może być puste
  - klucz obcy
- ID dania *(DishID)*
  - integer
  - nie może być puste
  - klucz obcy
- Cena jednostkowa *(UnitPrice)*
  - float
  - nie może być puste
  - cena jest z zakresu od 0 do 5000 zł
- Ilość zamówionych sztuk potrawy *(Quantity)*
  - integer
  - nie może być puste
  - wartość z zakresu od 0 do 5000

```
IF OBJECT_ID('OrderDetails', 'U') IS NOT NULL
 DROP TABLE OrderDetails
CREATE TABLE OrderDetails (
    OrderID INT NOT NULL FOREIGN KEY REFERENCES Orders(OrderID),
    DishID INT NOT NULL FOREIGN KEY REFERENCES Dishes(DishID),
    UnitPrice FLOAT NOT NULL CHECK(UnitPrice >= 0 and UnitPrice <= 5000),
    Quantity INT NOT NULL CHECK(Quantity >= 0 and Quantity <= 5000)
)
```

Tables

Tabela przechowuje dane dotyczące maksymalnej i obecnie dozwolonej (w związku z pandemią) pojemności stolików:

- ID stolika (*TableID*)
  - integer
  - nie może być puste
  - klucz główny
- Maksymalna pojemność stolika (*MaxCapacity*)
  - integer
  - nie może być puste

    - przyjmuje wartości w zakresie od 2 do 100
    - jest większe bądź równe *CurrentCapacity*
- Dozwolona obecnie pojemność stolika (*CurrentCapacity*)
    - integer
    - nie może być puste
    - przyjmuje wartości w zakresie od 2 do 100
    - jest mniejsze bądź równe pojemności maksymalnej

```
IF OBJECT_ID('Tables', 'U') IS NOT NULL
   DROP TABLE Tables
CREATE TABLE Tables(
   TableID INT NOT NULL PRIMARY KEY IDENTITY(1, 1),
   MaxCapacity INT NOT NULL CHECK (MaxCapacity >= 2 AND MaxCapacity <= 100),
   CurrentCapacity INT NOT NULL CHECK (CurrentCapacity >= 2 AND
CurrentCapacity <= 100),
   CHECK (CurrentCapacity <= MaxCapacity)
)
```

Reservations

Tabela przechowuje dane dotyczące rezerwacji stolików, takie jak:

- ID Rezerwacji *(ReservationID)*
    - integer
    - nie może być puste
    - klucz główny
- ID Zamówienia (*OrderID*)
    - integer
    - nie może być puste
    - klucz obcy
- ID Pracownika, który zatwierdził rezerwację *(EmployeeID)*
    - integer
    - nie może być puste
    - klucz obcy
- ID Klienta *(CustomerID)*
    - integer
    - nie może być puste
    - klucz obcy
- Data dokonania rezerwacji *(ReservationDate)*
    - data
    - nie może być puste
- Początek okresu, na który został zarezerwowany stolik *(RealizationDateStart)*
    - data
    - nie może być puste
    - jest większa od daty dokonania rezerwacji
- Koniec okresu, na który został zarezerwowany stolik *(RealizationDateEnd)*
    - data
    - nie może być puste
    - jest większa od daty początku okresu rezerwacji

- Ilość osób przy stoliku *(NumberOfPeople)*
  - integer
  - nie może być puste
  - należy do zakresu wartości od 0 do 100
- Numer zarezerwowanego stolika *(TableID)*
  - integer
  - nie może być puste
  - klucz obcy
- Flaga określająca, czy ta rezerwacja była anulowana *(isCancelled)*
  - bit
  - może być puste
  - wartość domyślna 0
- Flaga określająca, czy ta rezerwacja została dokonana przez osobę (nie firmę) *(isByPerson)*
  - bit
  - może być puste
  - wartość domyślna 1

```
IF OBJECT_ID('Reservations', 'U') IS NOT NULL
   DROP TABLE Reservations
CREATE TABLE Reservations(
   ReservationID INT NOT NULL PRIMARY KEY IDENTITY(1, 1),
   OrderID INT NOT NULL FOREIGN KEY REFERENCES Orders(OrderID),
   EmployeeID INT NOT NULL FOREIGN KEY REFERENCES Employees(EmployeeID),
   CustomerID INT NOT NULL FOREIGN KEY REFERENCES Customers(CustomerID),
   ReservationDate DATETIME NOT NULL,
   RealizationDateStart DATETIME NOT NULL,
   RealizationDateEnd DATETIME NOT NULL,
   NumberOfPeople INT NOT NULL CHECK (NumberOfPeople > 0 AND NumberOfPeople <= 100),
   TableID INT NOT NULL FOREIGN KEY REFERENCES Tables(TableID),
   IsCancelled BIT DEFAULT 0,
   IsByPerson BIT DEFAULT 1,
   CHECK (ReservationDate <= RealizationDateStart),
   CHECK (RealizationDateStart < RealizationDateEnd)
)
```

Products

Tabela przechowuje dane dotyczące produktów wykorzystywanych do przygotowania dań, takie jak:

- ID produktu *(ProductID)*
  - integer
  - nie może być puste
  - klucz główny
- Nazwa produktu *(ProductName)*
  - łańcuch znaków o długości 50
  - nie może być puste
- ID kategorii produktu *(ProductCategoryID)*

    - integer
    - nie może być puste
    - klucz obcy
- Ilość sztuk w magazynie *(UnitsInStock)*
    - integer
    - nie może być puste
    - wartość w zakresie od 0 do 5000 sztuk

```
IF OBJECT_ID('Products', 'U') IS NOT NULL
 DROP TABLE Products
CREATE TABLE Products(
    ProductID INT NOT NULL PRIMARY KEY IDENTITY(1, 1),
    ProductName VARCHAR(50) NOT NULL,
    ProductCategoryID INT NOT NULL FOREIGN KEY REFERENCES
ProductCategories(ProductCategoryID),
    UnitsInStock INT NOT NULL CHECK(UnitsInStock >= 0 AND UnitsInStock <= 5000)
)
```

ProductCategories

Tabela przechowuje informacje o kategorii produktów, takie jak:

- ID kategorii produktu *(ProductCategoryID)*
    - integer
    - nie może być psute
    - klucz główny
- Nazwa kategorii produktu *(ProductCategoryName)*
    - łańcuch znaków o długości 30
    - nie może być puste
    - unikalne wartości
- Opis kategorii *(Description)*
    - łańcuch znaków o długości 50
    - może być puste

```
IF OBJECT_ID('ProductCategories', 'U') IS NOT NULL
 DROP TABLE ProductCategories
    CREATE TABLE ProductCategories(
    ProductCategoryID INT NOT NULL PRIMARY KEY IDENTITY(1, 1),
    ProductCategoryName VARCHAR(30) NOT NULL UNIQUE,
    Description VARCHAR(50)
)
```

Dishes

Tabela przechowuje informacje o daniach serwowanych w firmie. Każde danie posiada dane:

- ID dania *(DishID)*
    - integer
    - nie może być puste
    - klucz główny

- Nazwę dania *(DishName)*
    - łańcuch znaków o długości 50
    - nie może być puste
    - unikalne wartości
- Cenę *(UnitPrice)*
    - float
    - nie może być nullem
    - wartość z zakresu od 0 do 5000 zł
- Flagę określającą, czy to danie jest dostępne (*isAvailable*)
    - bit
    - nie może być puste
    - domyślna wartość: 1

```
IF OBJECT_ID('Dishes', 'U') IS NOT NULL
 DROP TABLE Dishes
CREATE TABLE Dishes (
    DishID INT NOT NULL PRIMARY KEY IDENTITY(1, 1),
    DishName VARCHAR(50) NOT NULL UNIQUE,
    UnitPrice FLOAT NOT NULL CHECK(UnitPrice >= 0 AND UnitPrice <= 5000),
    isAvailable BIT NOT NULL DEFAULT 1
)
```

DishDetails

Tabela łącznikowa pomiędzy daniami, a produktami w nich zawartymi, która przechowuje informacje, takie jak:

- ID dania *(DishID)*
    - integer
    - nie może być puste
    - klucz obcy
- ID produktu, znajdującego się w tym daniu *(ProductID)*
    - integer
    - nie może być puste
    - klucz obcy

```
IF OBJECT_ID('DishDetails', 'U') IS NOT NULL
 DROP TABLE DishDetails
CREATE TABLE DishDetails (
    DishID INT NOT NULL FOREIGN KEY REFERENCES Dishes(DishID),
    ProductID INT NOT NULL FOREIGN KEY REFERENCES Products(ProductID)
)
```

Menu

Tabela przechowuje informacje o menu, takie jak:

- ID menu *(MenuID)*
    - integer
    - nie może być puste

    - klucz główny
- Datę ustalenia menu *(ArrangementDate)*
    - datetime
    - nie może być puste
    - ma wartość mniejszą niż datę, od której obowiązuje menu
- Datę, w której menu zaczęło obowiązywać (*StartDate)*
    - datetime
    - nie może być puste
    - ma wartość większą niż data ustalenia menu
- Datę, do której menu obowiązywało *(EndDate)*
    - datetime
    - nie może być puste

```
IF OBJECT_ID('Menu', 'U') IS NOT NULL
DROP TABLE Menu
CREATE TABLE Menu(
   MenuID INT NOT NULL PRIMARY KEY IDENTITY(1, 1),
   ArrangementDate DATETIME NOT NULL,
   StartDate DATETIME NOT NULL,
   EndDate DATETIME NOT NULL,
   CHECK(ArrangementDate < StartDate),
   CHECK(StartDate < EndDate),
   CHECK (DATEDIFF(day, ArrangementDate, StartDate) >= 1),
   CHECK (DATEDIFF(week, StartDate, EndDate) <= 2)
)
```

MenuDishes

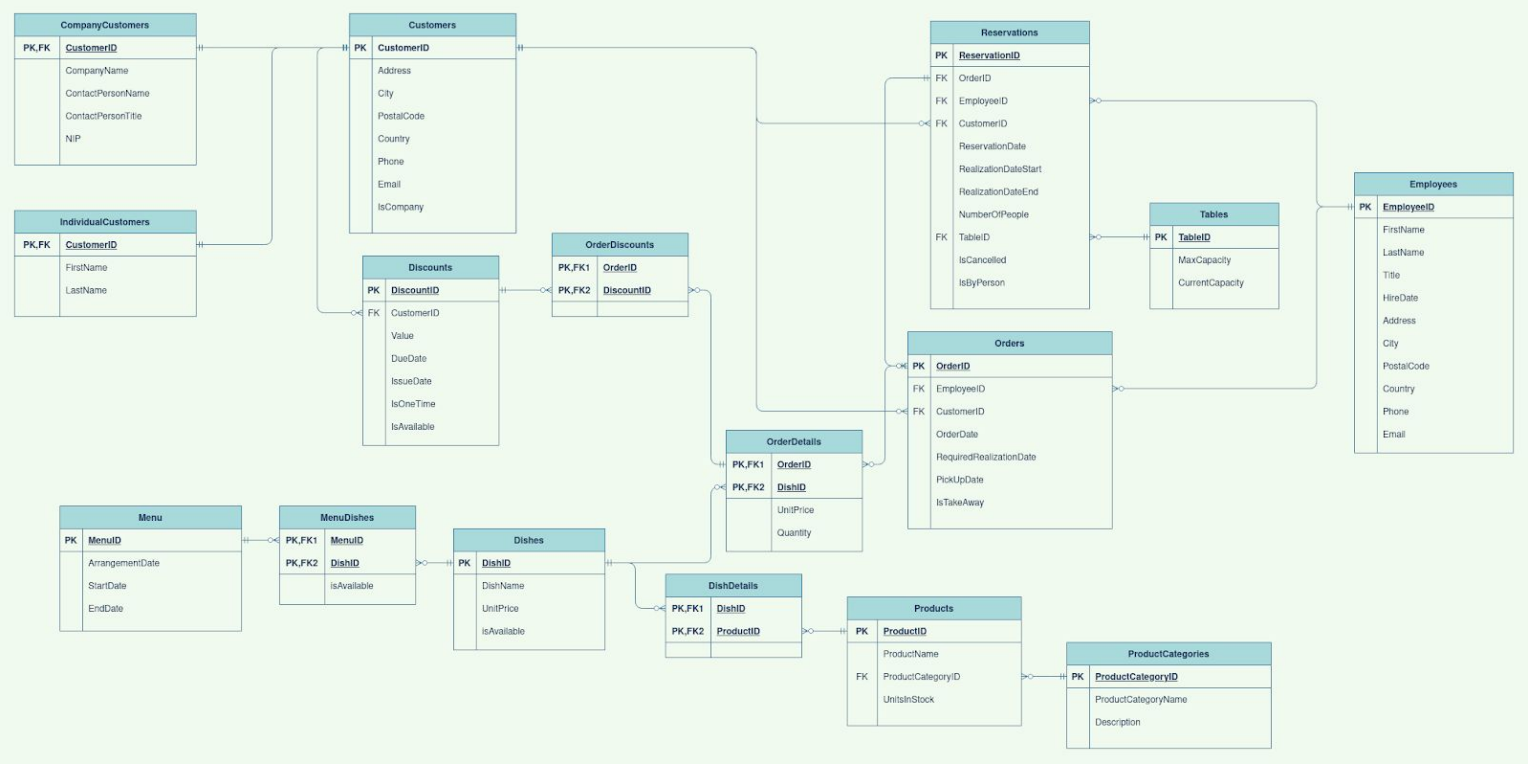
Tabela łącznikowa między daniami a menu, przechowująca informacje o daniach, które się znajdują na menu:

- ID Menu *(MenuID)*
    - integer
    - nie może być puste
    - klucz obcy
- ID Dania znajdującego się na menu *(DishID)*
    - integer
    - nie może być puste
    - klucz obcy
- Flagę określającą, czy dane danie nie zostało usunięte z menu z powodu wyczerpania się półproduktów
    - bit
    - nie może być puste
    - domyślna wartość: 1

```
IF OBJECT_ID('MenuDishes', 'U') IS NOT NULL
 DROP TABLE MenuDishes
```

```
CREATE TABLE MenuDishes(
    MenuID INT NOT NULL FOREIGN KEY REFERENCES Menu(MenuID),
    DishID INT NOT NULL FOREIGN KEY REFERENCES Dishes(DishID),
    isAvailable BIT NOT NULL DEFAULT 1
)
```

Schemat bazy danych



Procedury

AddNewCustomer

Procedura dodająca nowego klienta restauracji.
Argumenty:

- Numer telefonu
- Email
- Czy to klient indywidualny, czy firmowy
- Dane adresowe - opcjonalne (są domyślnie ustawione na null)

```
DROP PROCEDURE IF EXISTS AddNewCustomer
GO
CREATE PROCEDURE AddNewCustomer
   @PhONe VARCHAR(20),
   @Email VARCHAR(30),
   @IsCompany BIT,
   @Address VARCHAR(50) = NULL,
```

```
   @City VARCHAR(30) = NULL,
   @PostalCode VARCHAR(30) = NULL,
   @Country VARCHAR(30) = NULL
AS
BEGIN
   INSERT INTO Customers(Address, City, PostalCode, Country, PhONe,
Email, IsCompany)
   VALUES (@Address, @City, @PostalCode, @Country, @PhONe, @Email,
@IsCompany)
END
```

AddNewCompanyCustomer

Procedura dodająca nowego klienta firmowego restauracji.
Argumenty:

- Nazwa firmy
- Numer telefonu
- Email
- Czy to klient indywidualny, czy firmowy
- Dane teleadresowe - opcjonalne (są domyślnie ustawione na null)

```
DROP PROCEDURE IF EXISTS AddNewCompanyCustomer
GO
CREATE PROCEDURE AddNewCompanyCustomer
   @CompanyName VARCHAR(50),
   @Phone VARCHAR(20),
   @Email VARCHAR(30),
   @NIP VARCHAR(10),
   @ContactPersonName VARCHAR(50) = NULL,
   @ContactPersonTitle VARCHAR(50) = NULL,
   @Address VARCHAR(50) = NULL,
   @City VARCHAR(30) = NULL,
   @PostalCode VARCHAR(30) = NULL,
   @Country VARCHAR(30) = NULL
AS
BEGIN
   EXEC AddNewCustomer @Phone, @Email, 1, @Address, @City, @PostalCode, @Country
   DECLARE @CustomerID INT
   SET @CustomerID = (SELECT CustomerID FROM Customers WHERE Email = @Email AND
Phone = @Phone)
   INSERT INTO CompanyCustomers(CustomerID, CompanyName, ContactPersonName,
ContactPersonTitle, NIP)
   VALUES (@CustomerID, @CompanyName, @ContactPersonName, @ContactPersonTitle,
@NIP)
END
```

Przykłady użycia:

```
EXEC AddNewCompanyCustomer 'Good Company', '123456789', 'email@gmail.com',
'1223456799'
EXEC AddNewCompanyCustomer 'Good Company 2', '123456799', 'email2@gmail.com',
'1234567889', @country = 'Poland'
```

AddNewIndividualCustomer

Procedura dodająca nowego klienta indywidualnego restauracji.
Argumenty:

- Imię
- Nazwisko
- Numer telefonu
- Email
- Czy to klient indywidualny, czy firmowy
- Dane adresowe - opcjonalne (są domyślnie ustawione na null)

```
DROP PROCEDURE IF EXISTS AddNewIndividualCustomer
GO
CREATE PROCEDURE AddNewIndividualCustomer
   @FirstName VARCHAR(30),
   @LastName VARCHAR(30),
   @Phone VARCHAR(20),
   @Email VARCHAR(30),
   @Address VARCHAR(50) = NULL,
   @City VARCHAR(30) = NULL,
   @PostalCode VARCHAR(30) = NULL,
   @Country VARCHAR(30) = NULL
AS
BEGIN
   EXEC AddNewCustomer @Phone, @Email, 0, @Address, @City, @PostalCode, @Country
   DECLARE @CustomerID INT
   SET @CustomerID = (SELECT CustomerID FROM Customers WHERE Email = @Email AND
Phone = @Phone)
   INSERT INTO IndividualCustomers(CustomerID, FirstName, LastName)
   VALUES (@CustomerID, @FirstName, @LastName)
END
```

Przykłady użycia:

```
EXEC AddNewIndividualCustomer 'John', 'Doe', '123456123', 'john@gmail.com'

EXEC AddNewIndividualCustomer 'John', 'Doe', '123456124', 'john2@gmail.com',
@city = 'Warsaw'
```

AddNewEmployee

Procedura dodająca nowego pracownika restauracji.
Argumenty:

- imię
- nazwisko
- tytuł
- numer telefonu
- dane teleadresowe - opcjonalne

```
DROP PROCEDURE IF EXISTS AddNewEmployee
GO
CREATE PROCEDURE AddNewEmployee
  @FirstName VARCHAR(30),
  @LastName VARCHAR(30),
  @Title VARCHAR(30),
  @Phone VARCHAR(20),
  @Email VARCHAR(30),
  @HireDate DATETIME = NULL,
  @Address VARCHAR(50) = NULL,
  @PostalCode NVARCHAR(30) = NULL,
  @City NVARCHAR(30) = NULL,
  @Country NVARCHAR(30) = NULL
AS
BEGIN
  INSERT INTO Employees (FirstName, LastName, Title, Phone, Email,
HireDate, Address, PostalCode, City, Country)
  VALUES (@FirstName, @LastName, @Title, @Phone, @Email, @HireDate,
@Address, @PostalCode, @City, @Country)
END
```

Przykłady użycia:

```
EXEC AddNewEmployee 'Anne', 'Smith', 'head of procurement', '165935616',
'asmith@gmail.com'

EXEC AddNewEmployee 'James', 'Cameron', 'Warehouse manager',
'789243666', 'jamesCameron@restaurant.com', @Address = 'NorthStreet
31/16', @PostalCode = '30-200', @City = 'Naples', @Country = 'Italy'
```

AddNewOrder

Procedura dodająca nowe zamówienie.
Argumenty:

- Identyfikator klienta
- Identyfikator pracownika, który je przyjął
- Data złożenia zamówienia

- Data realizacji zamówienia
- Data odbioru zamówienia - opcjonalne
- Czy jest na wynos - domyślnie 1

```
DROP PROCEDURE IF EXISTS AddNewOrder
GO
CREATE PROCEDURE AddNewOrder
  @EmployeeID INT,
  @CustomerID INT,
  @OrderDate DATETIME,
  @RequiredRealisationDate DATETIME,
  @PickUpDate DATETIME = NULL,
  @IsTakeAway BIT = 1
AS
BEGIN
  IF (SELECT COUNT(EmployeeID) FROM Employees WHERE EmployeeID =
@EmployeeID) < 1
     THROW 50001, 'No such employee.', 1
  IF (SELECT COUNT(CustomerID) FROM Customers WHERE CustomerID =
@CustomerID) < 1
     THROW 50002, 'No such customer.', 1
  INSERT INTO Orders (EmployeeID, CustomerID, OrderDate,
RequiredRealisationDate, PickUpDate, IsTakeAway)
  VALUES (@EmployeeID, @CustomerID, @OrderDate,
@RequiredRealisationDate, @PickUpDate, @IsTakeAway)
END
```

Przykład użycia:

```
EXEC AddNewOrder 5, 4, '2021/01/17', '2021/01/27'
```

AddNewOrderDetails

Procedura dodająca nowe szczegóły zamówienia.
Argumenty:

- ID zamówienia
- ID dania
- cena, po której sprzedano danie
- ilość sztuk - domyślnie 1

```
DROP PROCEDURE IF EXISTS AddNewOrderDetails
GO
CREATE PROCEDURE AddNewOrderDetails
  @OrderID INT,
  @DishID INT,
  @UnitPrice FLOAT,
  @Quantity INT = 1
AS
BEGIN
```

```
  IF (SELECT COUNT(OrderID) FROM Orders WHERE OrderID = @OrderID) < 1
     THROW 50001, 'No such order.', 1
  IF (SELECT COUNT(DishID) FROM Dishes WHERE DishID = @DishID) < 1
     THROW 50002, 'No such dish.', 1
  INSERT INTO OrderDetails (OrderID, DishID, UnitPrice, Quantity)
  VALUES (@OrderID, @DishID, @UnitPrice, @Quantity)
END
```

Przykład użycia:

```
EXEC AddNewOrderDetails 1, 1, 25.5
```

AddNewDish

Procedura dodająca nowe danie.
Argumenty:

- Nazwa
- Cena

```
DROP PROCEDURE IF EXISTS AddNewDish
GO
CREATE PROCEDURE AddNewDish
  @Name VARCHAR(50),
  @UnitPrice FLOAT
AS
BEGIN
  INSERT INTO Dishes(DishName, UnitPrice)
  VALUES (@Name, @UnitPrice)
END
```

Przykład użycia:

```
EXEC AddNewDish 'Penne al forno', 25.50
```

AddProductToDish

Procedura dodająca produkt do dania.
Argumenty:

- ID produktu
- ID dania

```
DROP PROCEDURE IF EXISTS AddProductToDish
GO
CREATE PROCEDURE AddProductToDish
  @ProductID INT,
  @DishID INT
AS
BEGIN
  IF (SELECT COUNT(ProductID) FROM Products WHERE ProductID = @ProductID) < 1
     THROW 50001, 'No such product.', 1
```

```
  IF (SELECT COUNT(DishID) FROM Dishes WHERE DishID = @DishID) < 1
     THROW 50002, 'No such dish.', 1
  INSERT INTO DishDetails (ProductID, DishID)
  VALUES (@ProductID, @DishID)
END
```

Przykłady użycia:

```
EXEC AddProductToDish 2, 1
```

AddNewProduct

Procedura dodająca nowy produkt-składnik dania.
Argumenty:

- Nazwa
- Kategoria
- Ilość sztuk - opcjonalne

```
DROP PROCEDURE IF EXISTS AddNewProduct
GO
CREATE PROCEDURE AddNewProduct
  @Name VARCHAR(50),
  @ProductCategoryID INT,
  @UnitsInStock INT = 0
AS
BEGIN
  IF (SELECT COUNT(ProductCategoryID) FROM ProductCategories WHERE
ProductCategoryID = @ProductCategoryID) < 1
     THROW 50001, 'No such product category', 1
  INSERT INTO Products (ProductName, ProductCategoryID, UnitsInStock)
  VALUES (@Name, @ProductCategoryID, @UnitsInStock)
END
```

Przykład użycia:

```
EXEC AddNewProduct 'Penne' , 2
```

AddNewProductCategory

Procedura dodająca nową kategorię produktu.
Argumenty:

- Nazwa kategorii
- Opis - opcjonalne

```
DROP PROCEDURE IF EXISTS AddNewProductCategory
GO
CREATE PROCEDURE AddNewProductCategory
  @Name VARCHAR(30),
  @Description VARCHAR(50) = NULL
AS
```

```
BEGIN
  INSERT INTO ProductCategories (ProductCategoryName, Description)
  VALUES (@Name, @Description)
END
```

Przykłady użycia:

```
EXEC AddNewProductCategory 'Seafood', 'Seaweed, sushi and fish'
EXEC AddNewProductCategory 'Pasta'
```

AddNewDiscount

Procedura dodająca nowy rodzaj rabatu.
Argumenty:

- ID klienta
- Wartość
- Data wystawienia rabatu
- Termin ważności rabatu (domyślnie null)
- Czy rabat jest jednorazowy (domyślnie tak)
- Czy rabat jest dostępny (domyślnie tak)

```
DROP PROCEDURE IF EXISTS AddNewDiscount
GO
CREATE PROCEDURE AddNewDiscount
   @CustomerID INT,
   @Value FLOAT,
   @IssueDate DATETIME,
   @DueDate DATETIME = NULL,
   @IsOneTime BIT = 1,
   @IsAvailable BIT = 1
AS
BEGIN
   INSERT INTO Discounts(CustomerID, Value, DueDate, IssueDate, IsOneTime,
IsAvailable)
   VALUES (@CustomerID, @Value, @DueDate, @IssueDate, @IsOneTime, @IsAvailable)
END
```

Przykłady użycia:

```
EXEC AddNewDiscount 4, 0.25, '2021-01-16'
EXEC AddNewDiscount 4, 0.30, '2021-01-16', @IsOneTime = 0, @DueDate = '2021-02-16'
```

AddNewReservation

Procedura dodająca nową rezerwację stolika.
Argumenty:

- ID klienta, który dokonał rezerwacji
- ID pracownika, który ją zatwierdził
- Ilość członków
- Data, na którą został zarezerwowany stolik

- Data dokonania rezerwacji
- Numer stolika
- Czy rezerwacja została odwołana (domyślnie nie)
- Czy rezerwacja została dokonana przez klienta indywidualnego (domyślnie tak)

```
DROP PROCEDURE IF EXISTS AddNewReservation
GO
CREATE PROCEDURE AddNewReservation
   @CustomerID INT,
   @EmployeeID INT,
   @OrderID INT,
   @NumberOfPeople INT,
   @RealizationDateStart DATETIME,
   @RealizationDateEnd DATETIME,
   @ReservationDate DATETIME,
   @TableID INT,
   @IsCancelled BIT = 0,
   @IsByPerson BIT = 1
AS
BEGIN
   IF (SELECT COUNT(CustomerID) FROM Customers WHERE CustomerID = @CustomerID) < 1
       THROW 50001, 'No such customer.', 1
   IF (SELECT COUNT(EmployeeID) FROM Employees WHERE EmployeeID = @EmployeeID) < 1
       THROW 50001, 'No such employee.', 1
   IF (SELECT COUNT(TableID) FROM Tables WHERE TableID = @TableID) < 1
       THROW 50001, 'No such table.', 1
   IF @NumberOfPeople > (SELECT CurrentCapacity FROM Tables WHERE TableID = @TableID)
       THROW 50002, 'Number of people is too big.', 1
   IF (SELECT COUNT(OrderIDID) FROM Orders WHERE OrderID = @OrderID) < 1
       THROW 50001, 'No such order.', 1
   INSERT INTO Reservations(EmployeeID, CustomerID, ReservationDate,
RealizationDateStart, RealizationDateEnd, NumberOfPeople, TableID, IsByPerson,
IsCancelled, OrderID)
   VALUES (@EmployeeID, @CustomerID, @ReservationDate, @RealizationDateStart,
@RealizationDateEnd, @NumberOfPeople, @TableID, @IsByPerson, @IsCancelled, @OrderID)
END
```

Przykład użycia:

```
EXEC AddNewReservation 5, 1, 1, 3, '2021-01-29 18:00', '2021-01-29
20:00','2021-01-16', 2
```

AddDishToMenu

Procedura dodająca danie do menu.
Argumenty:

- ID menu
- ID dania
- Czy danie jest dostępne (domyślnie tak)

```
DROP PROCEDURE IF EXISTS AddDishToMenu
GO
CREATE PROCEDURE AddDishToMenu
   @MenuID INT,
   @DishID INT,
   @IsAvailable BIT = 1
AS
BEGIN
   IF (SELECT COUNT(MenuID) FROM Menu WHERE MenuID = @MenuID) < 1
       THROW 50001, 'No such menu.', 1
   IF (SELECT COUNT(DishID) FROM Dishes WHERE DishID = @DishID) < 1
       THROW 50001, 'No such dish.', 1
   INSERT INTO MenuDishes(MenuID, DishID, IsAvailable)
   VALUES (@MenuID, @DishID, @IsAvailable)
END
```

Przykłady użycia:

```
EXEC AddDishToMenu 1, 10
EXEC AddDishToMenu 1, 10, 0
```

AddNewMenu

Procedura dodająca nowe menu do bazy.
Argumenty:

- Data, w której zostało ono ustalone
- Data, od której obowiązuje
- Data, do której obowiązuje

```
DROP PROCEDURE IF EXISTS AddNewMenu
GO
CREATE PROCEDURE AddNewMenu
   @ArrangementDate DATETIME,
   @StartDate DATETIME,
   @EndDate DATETIME
AS
BEGIN
   INSERT INTO Menu(ArrangementDate, StartDate, EndDate)
   VALUES (@ArrangementDate, @StartDate, @EndDate)
END
```

Przykład użycia:

```
EXEC AddNewMenu '2021-01-16', '2021-01-18', '2021-01-22'
```

AddNewTable

Procedura dodająca nowy stolik do bazy.
Argumenty:

- Maksymalna pojemność stolika
- Obecnie dozwolona obecność stolika (domyślnie równa pojemności maksymalnej)

```
DROP PROCEDURE IF EXISTS AddNewTable
GO
CREATE PROCEDURE AddNewTable
   @MaxCapacity INT,
   @CurrentCapacity INT = NULL
AS
BEGIN
   IF @CurrentCapacity IS NOT NULL
       INSERT INTO Tables(MaxCapacity, CurrentCapacity)
       VALUES (@MaxCapacity, @CurrentCapacity)
   ELSE
       INSERT INTO Tables(MaxCapacity, CurrentCapacity)
       VALUES (@MaxCapacity, @MaxCapacity)
END
```

Przykłady użycia:

```
EXEC AddNewTable 10
EXEC AddNewTable 10, 4
```

ChangeTableCurrentCapacity

Procedura do zmiany aktualnie dozwolonej liczby miejsc danego stolika.
Argumenty:

- ID stolika
- Nowa dozwolona liczba miejsc

```
DROP PROCEDURE IF EXISTS ChangeTableCurrentCapacity
GO
CREATE PROCEDURE ChangeTableCurrentCapacity
   @TableID INT,
   @NewCurrentCapacity INT
AS
BEGIN
   IF (SELECT COUNT(TableID) FROM Tables WHERE TableID = @TableID) < 1
       THROW 50001, 'No such tables.', 1
   UPDATE Tables
   SET CurrentCapacity = @NewCurrentCapacity
```

```
   WHERE TableID = @TableID
END
```

Przykład użycia:

```
EXEC ChangeTableCurrentCapacity 1, 8
```

ChangeTablesLimit

Procedura do zmiany limitu liczby miejsc w restauracji. Limit jest zmieniany tylko w przypadku gdy podana liczba jest mniejsza od pojemności maksymalnej restauracji.
Argumenty:

- Liczba, na którą jest zmieniany limit

```
DROP PROCEDURE IF EXISTS ChangeTablesLimit
GO
CREATE PROCEDURE ChangeTablesLimit
   @NewLimit INT
AS
BEGIN
   DECLARE @GeneralCapacity INT
   SET @GeneralCapacity = (SELECT SUM(MaxCapacity) FROM Tables)
   IF @NewLimit > @GeneralCapacity
       THROW 50002, 'New limit is too big.', 1
   DECLARE @Coef FLOAT
   SET @Coef =  CAST(@NewLimit AS FLOAT) / @GeneralCapacity
   UPDATE Tables
   SET CurrentCapacity = CAST(MaxCapacity * @Coef AS INT)
END
```

Przykład użycia:

```
EXEC ChangeTablesLimit 15
```

RemoveTablesLimits

Procedura do usuwania wszelkich ograniczeń miejsc i przywrócenia dozwolonej liczbie miejsc maksymalnej wartości na wszystkich stolikach.

```
DROP PROCEDURE IF EXISTS RemoveTablesLimits
GO
CREATE PROCEDURE RemoveTablesLimits
AS
BEGIN
   UPDATE Tables
   SET CurrentCapacity = MaxCapacity
END
```

Przykład użycia:

```
EXEC RemoveTablesLimits
```

ChangeTableParticipants

Procedura do zmiany liczby członków rezerwacji.
Argumenty:

- ID rezerwacji
- Nowa liczba członków

```
DROP PROCEDURE IF EXISTS ChangeTableParticipants
GO
CREATE PROCEDURE ChangeTableParticipants
   @ReservationID INT,
   @NewNumberOfPeople INT
AS
BEGIN
   IF (SELECT COUNT(ReservationID) FROM Reservations WHERE ReservationID =
@ReservationID) < 1
       THROW 50001, 'No such reservation.', 1
   IF @NewNumberOfPeople > (
                               SELECT CurrentCapacity
                               FROM Reservations R
                               INNER JOIN Tables T ON T.TableID = R.TableID
                               WHERE ReservationID = @ReservationID
                           )
       THROW 50002, 'Number of people is too big.', 1
   UPDATE Reservations
   SET NumberOfPeople = @NewNumberOfPeople
   WHERE ReservationID = @ReservationID
END
```

Przykład użycia:

```
EXEC ChangeTableParticipants 1, 6
```

ChangeCustomerData

Procedura do zmiany danych klienta.
Argument wymagany to ID klienta.
Zmienia wartości tylko tych pól, które zostały podane jako argumenty.

```
DROP PROCEDURE IF EXISTS ChangeCustomerData
GO
CREATE PROCEDURE ChangeCustomerData
   @CustomerID INT,
   @Phone VARCHAR(20) = NULL,
   @Email VARCHAR(30) = NULL,
   @Address VARCHAR(50) = NULL,
   @City VARCHAR(30) = NULL,
   @PostalCode VARCHAR(30) = NULL,
   @Country VARCHAR(30) = NULL
AS
```

```
BEGIN
   IF (SELECT COUNT(CustomerID) FROM Customers WHERE CustomerID =
@CustomerID) < 1
       THROW 50001, 'No such customer.', 1
   IF @Phone IS NOT NULL
       UPDATE Customers
       SET Phone = @Phone
       WHERE CustomerID = @CustomerID
   IF @Email IS NOT NULL
       UPDATE Customers
       SET Email = @Email
       WHERE CustomerID = @CustomerID
   IF @Address IS NOT NULL
       UPDATE Customers
       SET Address = @Address
       WHERE CustomerID = @CustomerID
   IF @City IS NOT NULL
       UPDATE Customers
       SET City = @City
       WHERE CustomerID = @CustomerID
   IF @PostalCode IS NOT NULL
       UPDATE Customers
       SET PostalCode = @PostalCode
       WHERE CustomerID = @CustomerID
   IF @Country IS NOT NULL
       UPDATE Customers
       SET Country = @Country
       WHERE CustomerID = @CustomerID
END
```

Przykłady użycia:

```
EXEC ChangeCustomerData 4, @Email = 'good_email@Gmail.com'
EXEC ChangeCustomerData 5, @Email = 'john@Gmail.com', @City = 'Gdansk'
```

ChangeCompanyCustomerData

Procedura do zmiany danych klienta firmowego.
Argument wymagany to ID klienta.
Zmienia wartości tylko tych pól, które zostały podane jako argumenty.

```
DROP PROCEDURE IF EXISTS ChangeCompanyCustomerData
GO
CREATE PROCEDURE ChangeCompanyCustomerData
   @CustomerID INT,
   @CompanyName VARCHAR(50) = NULL,
   @ContactPersonName VARCHAR(50) = NULL,
   @ContactPersonTitle VARCHAR(50) = NULL,
```

```
   @NIP VARCHAR(10) = NULL,
   @Phone VARCHAR(20) = NULL,
   @Email VARCHAR(30) = NULL,
   @Address VARCHAR(50) = NULL,
   @City VARCHAR(30) = NULL,
   @PostalCode VARCHAR(30) = NULL,
   @Country VARCHAR(30) = NULL
AS
BEGIN
   IF (SELECT COUNT(CustomerID) FROM CompanyCustomers WHERE CustomerID =
@CustomerID) < 1
       THROW 50001, 'No such company customer.', 1
   EXEC ChangeCustomerData @CustomerID, @Phone, @Email, @Address, @City,
@PostalCode, @Country
   IF @CompanyName IS NOT NULL
       UPDATE CompanyCustomers
       SET CompanyName = @CompanyName
       WHERE CustomerID = @CustomerID
   IF @ContactPersonName IS NOT NULL
       UPDATE CompanyCustomers
       SET ContactPersonName = @ContactPersonName
       WHERE CustomerID = @CustomerID
   IF @ContactPersonName IS NOT NULL
       UPDATE CompanyCustomers
       SET ContactPersonTitle = @ContactPersonTitle
       WHERE CustomerID = @CustomerID
   IF @NIP IS NOT NULL
       UPDATE CompanyCustomers
       SET NIP = @NIP
       WHERE CustomerID = @CustomerID
END
```

Przykład użycia:

```
EXEC ChangeCompanyCustomerData 4, @CompanyName = 'Very Good Company',
@ContactPersonName = 'John Doe', @City = 'Krakow'
```

ChangeIndividualCustomerData

Procedura do zmiany danych klienta indywidualnego.
Argument wymagany to ID klienta.
Zmienia wartości tylko tych pól, które zostały podane jako argumenty.

```
DROP PROCEDURE IF EXISTS ChangeIndividualCustomerData
GO
CREATE PROCEDURE ChangeIndividualCustomerData
   @CustomerID INT,
   @FirstName VARCHAR(30) = NULL,
```

```
   @LastName VARCHAR(30) = NULL,
   @Phone VARCHAR(20) = NULL,
   @Email VARCHAR(30) = NULL,
   @Address VARCHAR(50) = NULL,
   @City VARCHAR(30) = NULL,
   @PostalCode VARCHAR(30) = NULL,
   @Country VARCHAR(30) = NULL
AS
BEGIN
   IF (SELECT COUNT(CustomerID) FROM IndividualCustomers WHERE
CustomerID = @CustomerID) < 1
       THROW 50001, 'No such individual customer.', 1
   EXEC ChangeCustomerData @CustomerID, @Phone, @Email, @Address, @City,
@PostalCode, @Country
   IF @FirstName IS NOT NULL
       UPDATE IndividualCustomers
       SET FirstName = @FirstName
       WHERE CustomerID = @CustomerID
   IF @LastName IS NOT NULL
       UPDATE IndividualCustomers
       SET LastName = @LastName
       WHERE CustomerID = @CustomerID
END
```

Przykład użycia:

```
EXEC ChangeIndividualCustomerData 6, @LastName = 'Smith', @Country = 'Poland'
```

ChangeEmployeeData

Procedura do zmiany danych pracownika.
Argument wymagany to ID pracownika.
Zmienia wartości tylko tych pól, które zostały podane jako argumenty.

```
DROP PROCEDURE IF EXISTS ChangeEmployeeData
GO
CREATE PROCEDURE ChangeEmployeeData
   @EmployeeID INT,
   @FirstName VARCHAR(30) = NULL,
   @LastName VARCHAR(30) = NULL,
   @Title VARCHAR(30) = NULL,
   @HireDate DATETIME = NULL,
   @Phone VARCHAR(20) = NULL,
   @Email VARCHAR(30) = NULL,
   @Address VARCHAR(50) = NULL,
   @City VARCHAR(30) = NULL,
   @PostalCode VARCHAR(30) = NULL,
   @Country VARCHAR(30) = NULL
```

```
AS
BEGIN
   IF (SELECT COUNT(EmployeeID) FROM Employees WHERE EmployeeID =
@EmployeeID) < 1
       THROW 50001, 'No such employee.', 1
   IF @FirstName IS NOT NULL
       UPDATE Employees
       SET FirstName = @FirstName
       WHERE EmployeeID = @EmployeeID
   IF @LastName IS NOT NULL
       UPDATE Employees
       SET LastName = @LastName
       WHERE EmployeeID = @EmployeeID
   IF @Title IS NOT NULL
       UPDATE Employees
       SET Title = @Title
       WHERE EmployeeID = @EmployeeID
   IF @HireDate IS NOT NULL
       UPDATE Employees
       SET HireDate = @HireDate
       WHERE EmployeeID = @EmployeeID
   IF @Phone IS NOT NULL
       UPDATE Employees
       SET Phone = @Phone
       WHERE EmployeeID = @EmployeeID
   IF @Email IS NOT NULL
       UPDATE Employees
       SET Email = @Email
       WHERE EmployeeID = @EmployeeID
   IF @Address IS NOT NULL
       UPDATE Employees
       SET Address = @Address
       WHERE EmployeeID = @EmployeeID
   IF @City IS NOT NULL
       UPDATE Employees
       SET City = @City
       WHERE EmployeeID = @EmployeeID
   IF @PostalCode IS NOT NULL
       UPDATE Employees
       SET PostalCode = @PostalCode
       WHERE EmployeeID = @EmployeeID
   IF @Country IS NOT NULL
       UPDATE Employees
       SET Country = @Country
       WHERE EmployeeID = @EmployeeID
END
```

Przykład użycia:

```
EXEC ChangeEmployeeData 1, @City = 'Warsaw', @Country = 'Poland'
```

RemoveEmployee

Procedura do usunięcia pracownika z bazy danych (w przypadku zwolnienia).
Argumenty:

- ID pracownika

```
DROP PROCEDURE IF EXISTS RemoveEmployee
GO
CREATE PROCEDURE RemoveEmployee
   @EmployeeID INT
AS
BEGIN
   IF (SELECT COUNT(EmployeeID) FROM Employees WHERE EmployeeID = @EmployeeID) < 1
       THROW 50001, 'No such employee.', 1
   DELETE FROM Employees
   WHERE EmployeeID = @EmployeeID
END
```

Przykład użycia:

```
EXEC RemoveEmployee 1
```

CancelReservation

Procedura do anulowania rezerwacji.
Argumenty:

- ID rezerwacji

```
DROP PROCEDURE IF EXISTS CancelReservation
GO
CREATE PROCEDURE CancelReservation
   @ReservationID INT
AS
BEGIN
   IF (SELECT COUNT(ReservationID) FROM Reservations WHERE ReservationID
= @ReservationID) < 1
       THROW 50001, 'No such reservation.', 1
   UPDATE Reservations
   SET IsCancelled = 1
   WHERE ReservationID = @ReservationID
END
```

Przykład użycia:

```
EXEC CancelReservation 1
```

CancelOrder

Procedura do usunięcia zamówienia z bazy.
Argumenty:

- ID zamówienia

```
DROP PROCEDURE IF EXISTS CancelOrder
GO
CREATE PROCEDURE CancelOrder
  @OrderID INT
AS
BEGIN
  IF (SELECT COUNT(OrderID) FROM Orders WHERE OrderID = @OrderID) < 1
      THROW 50001, 'No such order.', 1
  DELETE FROM Orders
  WHERE OrderID = @OrderID
  DELETE FROM OrderDetails
  WHERE OrderID = @OrderID
END
```

ChangeDishAvailability

Procedura do zmiany dostępności dania.
Argumenty:

- ID dania
- Nowa dostępność dania

```
DROP PROCEDURE IF EXISTS ChangeDishAvailability
GO
CREATE PROCEDURE ChangeDishAvailability
  @DishID INT,
  @NewAvailability BIT
AS
BEGIN
  IF (SELECT COUNT(DishID) FROM Dishes WHERE DishID = @DishID) < 1
      THROW 50001, 'No such dish.', 1
  UPDATE Dishes
  SET isAvailable = @NewAvailability
  WHERE DishID = @DishID
END
```

Przykład użycia:

```
EXEC ChangeDishAvailability 1, 1
```

ChangeProductUnitsInStock

Procedura do zmiany dostępnych produktów na magazynie.
Argumenty:

- ID produktu
- Nowa liczba dostępnych jednostek

```
DROP PROCEDURE IF EXISTS ChangeProductUnitsInStock
GO
CREATE PROCEDURE ChangeProductUnitsInStock
  @ProductID INT,
  @NewNumberOfUnits INT
AS
BEGIN
  IF (SELECT COUNT(ProductID) FROM Products WHERE ProductID = @ProductID) < 1
     THROW 50001, 'No such product.', 1
  UPDATE Products
  SET UnitsInStock = @NewNumberOfUnits
  WHERE ProductID = @ProductID
END
```

Przykład użycia:

```
EXEC ChangeProductUnitsInStock 1, 10
```

ChangeMenuDishAvailability

Procedura do zmiany dostępności dania menu.
Argumenty:

- ID menu
- ID dania
- Nowa dostępność dania

```
DROP PROCEDURE IF EXISTS ChangeMenuDishAvailability
GO
CREATE PROCEDURE ChangeMenuDishAvailability
  @MenuID INT,
  @DishID INT,
  @NewAvailability BIT
AS
BEGIN
  IF (SELECT COUNT(MenuID) FROM Menu WHERE MenuID = @MenuID) < 1
     THROW 50001, 'No such menu.', 1
  IF (SELECT COUNT(DishID) FROM Dishes WHERE DishID = @DishID) < 1
     THROW 50002, 'No such dish.', 1
  UPDATE MenuDishes
  SET isAvailable = @NewAvailability
```

```
  WHERE DishID = @DishID AND MenuID = @MenuID
END
```

Przykład użycia:

```
EXEC ChangeMenuDishAvailability 1, 1, 0
```

ChangeDishPrice

Procedura do zmiany ceny danego dania.
Argumenty:

- ID dania
- Nowa cena

```
DROP PROCEDURE IF EXISTS ChangeDishPrice
GO
CREATE PROCEDURE ChangeDishPrice
  @DishID INT,
  @Price FLOAT
AS
BEGIN
  IF (SELECT COUNT(DishID) FROM Dishes WHERE DishID = @DishID) < 1
     THROW 50002, 'No such dish.', 1
  UPDATE Dishes
  SET UnitPrice = @Price
  WHERE DishID = @DishID
END
```

Przykład użycia:

```
EXEC ChangeDishPrice 1, 12.25
```

UseOneTimeDiscount

Procedura ustawiająca jednorazowy rabat jako wykorzystany, czyli niedostępny.
Argumenty:

- ID rabatu

```
DROP PROCEDURE IF EXISTS UseOneTimeDiscount
GO
CREATE PROCEDURE UseOneTimeDiscount
  @DiscountID INT
AS
BEGIN
  IF (SELECT COUNT(DiscountID) FROM Discounts WHERE DiscountID =
@DiscountID) < 1
     THROW 50001, 'No such discount.', 1
  IF (SELECT COUNT(DiscountID) FROM Discounts WHERE DiscountID =
@DiscountID AND IsOneTime = 0) > 0
     THROW 50002, 'This discount is not one time', 1
```

```
  UPDATE Discounts
  SET IsAvailable = 0
  WHERE DiscountID = @DiscountID
END
```

Przykład użycia:

```
EXEC UseOneTimeDiscount 1
```

ChangeDiscountValue

Procedura zmieniająca wartość podanego rabatu.
Argumenty:

- ID rabatu
- Nowa wartość rabatu

```
DROP PROCEDURE IF EXISTS ChangeDiscountValue
GO
CREATE PROCEDURE ChangeDiscountValue
  @DiscountID INT,
   @NewValue FLOAT
AS
BEGIN
  IF (SELECT COUNT(DiscountID) FROM Discounts WHERE DiscountID = @DiscountID) < 1
     THROW 50001, 'No such discount.', 1
  UPDATE Discounts
  SET Value = @NewValue
  WHERE DiscountID = @DiscountID
END
```

Przykład użycia:

```
EXEC ChangeDiscountValue 1, 0.33
```

DeactivateDiscount

```
DROP PROCEDURE IF EXISTS DeactivateDiscount
GO
CREATE PROCEDURE DeactivateDiscount
  @DiscountID INT
AS
BEGIN
  IF (SELECT COUNT(DiscountID) FROM Discounts WHERE DiscountID = @DiscountID) < 1
     THROW 50001, 'No such discount.', 1
  UPDATE Discounts
  SET IsAvailable = 0
  WHERE DiscountID = @DiscountID
END
```

Przykład użycia:

```
EXEC DeactivateDiscount 1
```

TryAssignNewDiscountToCompanyCustomer

Procedura, która sprawdza, czy wszystkie wymagane warunki do przypisania nowego rabatu zostały spełnione przez klienta firmowego, i jeśli tak, to tworzy nowy rabat dla danego klienta.
Argumentami procedury są wszystkie parametry opisane w wymaganiach dotyczących rabatów klientów firmowych, czyli:

- FZ
- FK1
- FR1
- FM
- FK2
- FR2

oraz ID klienta.

```
DROP PROCEDURE IF EXISTS TryAssignNewDiscountToCompanyCustomer
GO
CREATE PROCEDURE TryAssignNewDiscountToCompanyCustomer
   @CustomerID INT,
   @FZ INT,
   @FK1 INT,
   @FR1 FLOAT,
   @FM FLOAT,
   @FK2 INT,
   @FR2 FLOAT
AS
BEGIN
   DECLARE @Date DATETIME
   SET @Date = GETDATE()
   DECLARE @DiscountID INT
   SET @DiscountID = (SELECT DiscountID FROM Discounts
       WHERE CustomerID = @CustomerID AND IsAvailable = 1 AND IsOneTime = 0)
   DECLARE @MonthOrdersNumber INT
   SET @MonthOrdersNumber = dbo.CustomerMonthOrdersNumberOfGivenValue(@CustomerID,
@FK1, @Date)
   IF @MonthOrdersNumber >= @FZ
       BEGIN
           IF @DiscountID IS NULL
                   EXEC AddNewDiscount @CustomerID, @FR1, @Date, @IsOneTime = 0
           ELSE
               BEGIN
                   DECLARE @OldDiscount FLOAT
                   SET @OldDiscount = (SELECT Value FROM Discounts
                       WHERE CustomerID = @CustomerID AND IsAvailable = 1 AND
IsOneTime = 0)
```

```
                    DECLARE @NewDiscount FLOAT
                    SET @NewDiscount = @OldDiscount + @FR1
                    SET @NewDiscount = IIF((@NewDiscount > @FM), @FM, @NewDiscount)
                    EXEC ChangeDiscountValue @DiscountID, @NewDiscount
                END
        END
    ELSE IF @MonthOrdersNumber = 0 AND @DiscountID IS NOT NULL
        BEGIN
            EXEC ChangeDiscountValue @DiscountID, 0
        END

    DECLARE @Quarter INT
    SET @Quarter = DATEPART(quarter, @Date)
    DECLARE @QuarterWorth INT
    SET @QuarterWorth = dbo.CustomerQuarterWorth(@CustomerID, @Quarter)
    IF @QuarterWorth >= @FK2
        EXEC AddNewDiscount @CustomerID, @FR2, @Date
END
```

Przykład użycia:

```
EXEC TryAssignNewDiscountToCompanyCustomer 1, 5, 500, 0.001, 0.04, 10000, 0.05
```

TryAssignNewDiscountToIndividualCustomer

Procedura, która sprawdza, czy wszystkie wymagane warunki do przypisania nowego rabatu zostały spełnione przez klienta indywidualnego, i jeśli tak, to tworzy nowy rabat dla danego klienta.
Argumentami procedury są wszystkie parametry opisane w wymaganiach dotyczących rabatów klientów indywidualnych, czyli:

- Z1
- K1
- R1
- K2
- R2
- D1
- K3
- R3
- D2

oraz ID klienta.

```
DROP PROCEDURE IF EXISTS TryAssignNewDiscountToIndividualCustomer
GO
CREATE PROCEDURE TryAssignNewDiscountToIndividualCustomer
    @CustomerID INT,
    @Z1 INT,
    @K1 INT,
    @R1 FLOAT,
```

```
    @K2 INT,
    @R2 FLOAT,
    @D1 INT,
    @K3 INT,
    @R3 FLOAT,
    @D2 INT
AS
BEGIN
    DECLARE @Date DATETIME
    SET @Date = GETDATE()
    IF dbo.CustomerHasNOrdersOfGivenValue(@CustomerID, @Z1, @K1) = 1
        EXEC AddNewDiscount @CustomerID, @R1, @Date

    IF dbo.CustomerOrderWorth(@CustomerID) >= @K2
        BEGIN
            DECLARE @DueDate DATETIME
            SET @DueDate = DATEADD(DAY, @D1, @Date)
            EXEC AddNewDiscount @CustomerID, @R2, @Date, @DueDate = @DueDate
        END
    IF dbo.CustomerOrderWorth(@CustomerID) >= @K3
        BEGIN
            DECLARE @NewDueDate DATETIME
            SET @NewDueDate = DATEADD(DAY, @D2, @Date)
            EXEC AddNewDiscount @CustomerID, @R3, @Date, @DueDate = @NewDueDate
        END
END
```

Przykład użycia:

```
EXEC TryAssignNewDiscountToIndividualCustomer 1, 10, 30, 0.03, 1000, 0.05,7,
5000, 0.05, 7
```

Funkcje

FreeTables

Funkcja wyświetlająca listę wolnych stolików w danym okresie czasowym.
Argumenty:

- Początek okresu (data i czas)
- Koniec okresu (data i czas)

```
DROP FUNCTION IF EXISTS FreeTables
GO
CREATE FUNCTION FreeTables
    (@StartDate DATETIME, @EndDate DATETIME)
RETURNS @FreeTables TABLE
    (
```

```
        TableID INT,
        MaxCapacity INT,
        CurrentCapacity INT
    )
AS
BEGIN
    INSERT INTO @FreeTables
    SELECT T.TableID, T.MaxCapacity, T.CurrentCapacity FROM Tables T
    LEFT JOIN Reservations R on T.TableID = R.TableID
    WHERE RealizationDateStart > @EndDate OR RealizationDateEnd <
@StartDate OR RealizationDateStart IS NULL
    RETURN
END
```

Przykład użycia:

```
SELECT * FROM FreeTables('2021-01-27', '2021-02-01')
```

CustomerDiscounts

Funkcja zwracająca listę rabatów podanego klienta.
Argumenty:

- ID klienta

```
DROP FUNCTION IF EXISTS CustomerDiscounts
GO
CREATE FUNCTION CustomerDiscounts
  (@CustomerID INT)
RETURNS TABLE
AS
RETURN
(
  SELECT * FROM Discounts
  WHERE CustomerID = @CustomerID
)
```

Przykład użycia:

```
SELECT * FROM CustomerDiscounts(4)
```

CustomerOrdersNumberForValue

Funkcja zwracająca liczbę zamówień podanego klienta za co najmniej określoną kwotę.
Argumenty:

- ID klienta
- Wysokość kwoty

```
DROP FUNCTION IF EXISTS CustomerOrdersNumberForValue
GO
CREATE FUNCTION CustomerOrdersNumberForValue
```

```
  (@CustomerID INT, @Money FLOAT)
RETURNS INT
AS
BEGIN
  DECLARE @OrderNumber INT
  SELECT @OrderNumber = COUNT(Orders.OrderID)
  FROM Orders
  INNER JOIN OrderDetails
  ON Orders.OrderID = OrderDetails.OrderID
  WHERE Orders.CustomerID = @CustomerID AND OrderDetails.UnitPrice >=
@Money;
  IF (@OrderNumber IS NULL)
     SET @OrderNumber = 0;
  RETURN @OrderNumber;
END
```

Przykład użycia:

```
SELECT dbo.CustomerOrdersNumberForValue(4, 20.0) AS OrdersNumber
```

CustomerOrdersNumber

Funkcja zwracająca liczbę zamówień podanego klienta.
Argumenty:

- ID klienta

CustomerOrderWorth

Funkcja zwracająca łączną kwotę wszystkich zamówień podanego klienta (przydatne w przypadku klienta indywidualnego dla ewentualnego obliczenia rabatu).
Argumenty:

- ID klienta

```
DROP FUNCTION IF EXISTS CustomerOrderWorth
GO
CREATE FUNCTION CustomerOrderWorth
  (@CustomerID INT)
RETURNS FLOAT
AS
BEGIN
  DECLARE @Worth INT
  SELECT @Worth = SUM(OrderDetails.UnitPrice * OrderDetails.Quantity *
(1 - ISNULL(Discounts.Value, 0)))
  FROM Orders
  INNER JOIN OrderDetails ON Orders.OrderID = OrderDetails.OrderID
  LEFT JOIN OrderDiscounts ON Orders.OrderID = OrderDiscounts.OrderID
   LEFT JOIN Discounts ON OrderDiscounts.DiscountID =
Discounts.DiscountID
  WHERE Orders.CustomerID = @CustomerID
```

```
  GROUP BY Orders.OrderID;

  IF (@Worth IS NULL)
     SET @Worth = 0;
  RETURN @Worth;
END
```

Przykład użycia:

```
SELECT dbo.CustomerOrderWorth(4) AS OrdersWorth
```

CustomerMonthOrdersNumber

Funkcja zwracająca liczbę zamówień podanego klienta w podanym miesiącu (przydatne w przypadku klienta firmowego dla ewentualnego obliczenia rabatu).
Argumenty:

- ID klienta
- Miesiąc (w postaci liczby int)
- Rok (w postaci liczby int)

```
DROP FUNCTION IF EXISTS CustomerMonthOrdersNumber
GO
CREATE FUNCTION CustomerMonthOrdersNumber
  (@CustomerID INT, @Month INT, @Year INT)
RETURNS INT
AS
BEGIN
  DECLARE @OrdersNumber INT
  SELECT @OrdersNumber = COUNT(Orders.OrderID) FROM Orders
  WHERE MONTH(OrderDate) = @Month AND YEAR(OrderDate) = @Year AND
CustomerID = @CustomerID
  IF (@OrdersNumber IS NULL)
     SET @OrdersNumber = 0
  RETURN @OrdersNumber
END
```

Przykład użycia:

```
SELECT dbo.CustomerMonthOrdersNumber(4, 1, 2021)
```

CustomerQuarterWorth

Funkcja zwracająca łączną kwotę wszystkich zamówień podanego klienta w podanym kwartale (przydatne w przypadku klienta indywidualnego dla ewentualnego obliczenia rabatu).
Argumenty:

- ID klienta
- Kwartał (w postaci liczby int)

```
DROP FUNCTION IF EXISTS CustomerQuarterWorth
```

```
GO
CREATE FUNCTION CustomerQuarterWorth
  (@CustomerID INT, @Quarter INT)
RETURNS FLOAT
AS
BEGIN
  DECLARE @Worth INT
  SELECT @Worth = SUM(OrderDetails.UnitPrice * OrderDetails.Quantity *
(1 - ISNULL(Discounts.Value, 0)))
  FROM Orders
  INNER JOIN OrderDetails ON Orders.OrderID = OrderDetails.OrderID
  LEFT JOIN OrderDiscounts ON Orders.OrderID = OrderDiscounts.OrderID
  LEFT JOIN Discounts ON OrderDiscounts.DiscountID =
Discounts.DiscountID
  WHERE Orders.CustomerID = @CustomerID AND DATEPART(QUARTER,
Orders.OrderDate) = @Quarter
  GROUP BY Orders.OrderID;
  IF (@Worth IS NULL)
     SET @Worth = 0;
  RETURN @Worth;
END
```

Przykład użycia:

```
SELECT dbo.CustomerQuarterWorth(4, 1)
```

CustomerHasNOrdersOfGivenValue

Funkcja zwracająca informacje o tym, czy podany klient złożył podaną liczbę zamówień, każde z których na co najmniej podaną kwotę.
Argumenty:

- ID klienta
- Liczba zamówień
- Minimalna wartość zamówienia

```
DROP FUNCTION IF EXISTS CustomerHasNOrdersOfGivenValue
GO
CREATE FUNCTION CustomerHasNOrdersOfGivenValue
   (@CustomerID INT, @N INT, @Value INT)
RETURNS BIT
AS
BEGIN
   DECLARE @OrdersNumber INT
   SET @OrdersNumber = (SELECT
dbo.CustomerOrdersNumberForValue(@CustomerID, @Value))
   RETURN IIF((@OrdersNumber >= @N), 1, 0)
END
```

Przykład użycia:

```
SELECT dbo.CustomerHasNOrdersOfGivenValue(4, 1, 10)
```

CustomerMonthOrdersNumberOfGivenValue

Funkcja zwracająca liczbę zamówień co najmniej podanej wartości w podanym miesiącu.
Argumenty:

- ID klienta
- Minimalna wartość zamówienia
- Koniec okresu czasu

```
DROP FUNCTION IF EXISTS CustomerMonthOrdersNumberOfGivenValue
GO
CREATE FUNCTION CustomerMonthOrdersNumberOfGivenValue
   (@CustomerID INT, @Value INT, @PeriodEndDate DATETIME)
RETURNS INT
AS
BEGIN
   DECLARE @PeriodStartDate DATETIME
   SET @PeriodStartDate = DATEADD(month, -1, @PeriodEndDate)
   DECLARE @OrdersNumber INT
   SET @OrdersNumber = (SELECT COUNT(O.OrderID) FROM Orders O
       INNER JOIN OrderDetails ODet ON O.OrderID = ODet.OrderID
       LEFT JOIN OrderDiscounts ODis ON ODet.OrderID = ODis.OrderID
       LEFT JOIN Discounts D ON ODis.DiscountID = D.DiscountID
           WHERE O.CustomerID = @CustomerID AND @PeriodStartDate <=
OrderDate AND OrderDate <= @PeriodEndDate
       HAVING SUM(Quantity * UnitPrice * (1 - ISNULL(Value, 0))) >
@Value
       )
   RETURN @OrdersNumber
END
```

Przykład użycia:

```
SELECT dbo.CustomerMonthOrdersNumberOfGivenValue(4, 10, '2021-01-20')
```

ReservationReport

Funkcja zwracająca raport dotyczący rezerwacji stolików w podanym okresie czasowym.
Argumenty:

- Początek okresu (data i czas)
- Koniec okresu (data i czas)

```
DROP FUNCTION IF EXISTS ReservationReport
GO
CREATE FUNCTION ReservationReport
  (@StartDate DATETIME, @EndDate DATETIME)
RETURNS TABLE
AS
```

```
RETURN
(
  SELECT ReservationID, Reservations.EmployeeID, Reservations.CustomerID,
Orders.OrderID, ReservationDate, RealizationDateStart, RealizationDateEnd,
NumberOfPeople, IsCancelled, IsByPerson, Tables.TableID
  FROM Reservations
  INNER JOIN Tables ON Tables.TableID = Reservations.TableID
  INNER JOIN Orders ON Orders.OrderID = Reservations.OrderID
  WHERE RealizationDateStart >= @StartDate AND RealizationDateEnd <= @EndDate
)
```

Przykład użycia:

```
SELECT * FROM ReservationReport('2021-01-17', '2021-03-15')
```

MenuReport

Funkcja zwracająca raport dotyczący menu w podanym okresie czasowym.
Argumenty:

- Początek okresu (data i czas)
- Koniec okresu (data i czas)

```
DROP FUNCTION IF EXISTS MenuReport
GO
CREATE FUNCTION MenuReport
   (@StartDate DATETIME, @EndDate DATETIME)
RETURNS @Report TABLE
   (
       MenuID INT,
       ArrangementDate DATETIME,
       StartDate DATETIME,
       EndDate DATETIME,
       DishName VARCHAR(50),
       UnitPrice FLOAT
   )
AS
BEGIN
   INSERT INTO @Report
   SELECT M.MenuID, ArrangementDate, StartDate, EndDate, DishName,
UnitPrice
   FROM Menu M
   INNER JOIN MenuDishes MD ON M.MenuID = MD.MenuID
   INNER JOIN Dishes D ON D.DishID = MD.DishID
   WHERE StartDate >= @StartDate AND EndDate <= @EndDate
   RETURN
END
```

Przykład użycia:

```
SELECT * FROM MenuReport('2021-01-15', '2021-02-01')
```

DiscountReport

Funkcja zwracająca raport dotyczący rabatów, które się pojawiły w podanym okresie czasowym.
Argumenty:

- Początek okresu (data i czas)
- Koniec okresu (data i czas)

```
DROP FUNCTION IF EXISTS DiscountReport
GO
CREATE FUNCTION DiscountReport
    (@StartDate DATETIME, @EndDate DATETIME)
RETURNS @Report TABLE
    (
        DiscountID INT,
        CustomerID INT,
        Value FLOAT,
        DueDate DATETIME,
        IssueDate DATETIME,
        IsOneTime BIT,
        IsAvailable BIT
    )
AS
BEGIN
    INSERT INTO @Report
    SELECT * FROM Discounts
    WHERE IssueDate >= @StartDate AND IssueDate <= @EndDate
    RETURN
END
```

Przykład użycia:

```
SELECT * FROM DiscountReport('2021-01-15', '2021-02-01')
```

CustomerOrders

Funkcja zwracająca informacje dotyczące kwot oraz czasu składania zamówień dla podanego klienta.
Argumenty:

- ID klienta

```
DROP FUNCTION IF EXISTS CustomerOrders
GO
CREATE FUNCTION CustomerOrders
   (@CustomerID INT)
RETURNS @Stats TABLE
   (
```

```
       Date DATETIME,
       Worth FLOAT
   )
AS
BEGIN
   INSERT INTO @Stats
   SELECT OrderDate, SUM(UnitPrice * Quantity * (1 - ISNULL(Value, 0)))
   FROM Customers C
   INNER JOIN Orders O ON C.CustomerID = O.CustomerID
   INNER JOIN OrderDetails ODet ON O.OrderID = ODet.OrderID
   LEFT JOIN OrderDiscounts ODisc ON O.OrderID = ODisc.OrderID
   LEFT JOIN Discounts D ON ODisc.DiscountID = D.DiscountID
   WHERE C.CustomerID = @CustomerID
   GROUP BY O.OrderID, OrderDate
   RETURN
END
```

Przykład użycia:

```
SELECT * FROM CustomerOrders(4)
```

MonthDishes

Funkcja wypisująca listę dań, które były w podanym menu w danym miesiącu lub wcześniej.
Argumenty:

- ID Menu
- Numer miesiąca (integer)
- Numer roku (integer)

```
DROP FUNCTION IF EXISTS MonthDishes
GO
CREATE FUNCTION MonthDishes
   (@MenuD INT, @month INT, @year INT)
RETURNS TABLE
AS
RETURN
(
    Date DATETIME,
    Worth FLOAT
    SELECT MenuDishes.DishID FROM MenuDishes
    INNER JOIN Dishes ON MenuDishes.DishID = Dishes.DishID
    INNER JOIN Menu ON MenuDishes.MenuID = Menu.MenuID
    WHERE MenuDishes.MenuID = @MenuID AND MONTH(Menu.StartDate) = @month
AND YEAR(Menu.StartDate) = @year
)
```

CanMakeReservation

Funkcja sprawdzająca, czy podanemu klientowi przysługuje prawo do rezerwacji stolika. Sprawdza, czy dany klient, który chce dokonać rezerwacji za co najmniej podaną wartość zamówienia, czy dokonał wcześniej co najmniej podaną liczbę zamówień, a jeśli dokonał mniej zamówień, to czy były one na inną podaną kwotę (wyższą).
Argumenty:

- ID klienta
- Wartość zamówienia
- dolna granica wartości zamówienia dopuszczającej klienta do rezerwacji
- Liczba zamówień
- Wyższa kwota

```
DROP FUNCTION IF EXISTS CanMakeReservation
GO
CREATE PROCEDURE CanMakeReservation
   (@CustomerID INT, @OrderValue FLOAT, @LowerLimitValue FLOAT, @PreviousOrders
INT, @ElseValue FLOAT
RETURNS BIT
AS
BEGIN
   DECLARE @Res BIT
   SET @Res = (CASE WHEN ((@LowerLimitValue <= @OrderValue AND
dbo.CustomersOrdersNumber(@CustomerID) >= @PreviousOrders)
OR (@LowerLimitValue >= @OrderValue AND
dbo.CustomersOrdersNumberForValue(@CustomerID, @ElseValue) <= @PreviousOrders))
THEN 1 ELSE 0 END);
   RETURN @Res;
END
```

Przykład użycia:

```
SELECT * FROM dbo.CanMakeReservation(4, 25, 15, 2, 26)
```

GenerateInvoice

Funkcja wypisująca fakturę dla danego klienta firmowego.
Argumenty:

- ID zamówienia
- Nazwa firmy gastronomicznej, wystawiającej fakturę
- Dane adresowe firmy gastronomicznej, wystawiającej fakturę

```
DROP FUNCTION IF EXISTS GenerateInvoice
GO
```

```
CREATE FUNCTION GenerateInvoice
  (@OrderID INT, @RestaurantName VARCHAR(50), @RestaurantAddress VARCHAR(200))
RETURNS TABLE
AS
RETURN
(
  SELECT GETDATE() AS 'Date of issue',
         @RestaurantName AS 'Seller',
         @RestaurantAddress AS 'Seller address',
         O.CustomerID,
         CompanyName AS 'Buyer',
         Address+' '+PostalCode+' '+City+' '+Country AS 'Buyer address',
         OrderDate,
         DishName,
         OD.UnitPrice,
         Quantity,
         Value,
         SUM(OD.UnitPrice * OD.Quantity * (1 - ISNULL(Discounts.Value, 0))) AS
'TotalValue'
  FROM Orders O
  INNER JOIN Customers ON Customers.CustomerID = O.CustomerID
  INNER JOIN CompanyCustomers ON CompanyCustomers.CustomerID =
Customers.CustomerID
  INNER JOIN OrderDetails OD ON OD.OrderID = O.OrderID
  INNER JOIN Dishes ON Dishes.DishID = OD.DishID
  LEFT JOIN OrderDiscounts ON OrderDiscounts.OrderID = OD.OrderID
  LEFT JOIN Discounts ON Discounts.DiscountID = OrderDiscounts.DiscountID
  WHERE O.OrderID = @OrderID
  GROUP BY OD.OrderID, O.CustomerID, CompanyName, Address, PostalCode, City,
Country, OrderDate, DishName, Quantity, Discounts.Value, OD.UnitPrice
)
```

Przykład użycia:

```
SELECT * FROM GenerateInvoice(1, 'Restaurant', 'Address')
```

GenerateCollectiveInvoice

Funkcja wypisująca fakturę zbiorczą dla danego klienta firmowego.
Argumenty:

- ID klienta
- Nazwa firmy gastronomicznej, wystawiającej fakturę
- Dane adresowe firmy gastronomicznej, wystawiającej fakturę

```
DROP FUNCTION IF EXISTS GenerateCollectiveInvoice
GO
CREATE FUNCTION GenerateCollectiveInvoice
  (@CustomerID INT, @RestaurantName VARCHAR(50), @RestaurantAddress
```

```
VARCHAR(200), @Month INT, @Year INT)
RETURNS TABLE
AS
RETURN
(
  SELECT GETDATE() AS 'Date of issue',
         @RestaurantName AS 'Seller',
         @RestaurantAddress AS 'Seller address',
         O.CustomerID,
         CompanyName AS 'Buyer',
         Address+' '+PostalCode+' '+City+' '+Country AS 'Buyer address',
         OrderDate,
         DishName,
         OD.UnitPrice,
         Quantity,
         Value,
         SUM(OD.UnitPrice * OD.Quantity * (1 - ISNULL(Discounts.Value, 0)))
AS 'TotalValue'
  FROM Orders O
  INNER JOIN Customers ON Customers.CustomerID = O.CustomerID
  INNER JOIN CompanyCustomers ON CompanyCustomers.CustomerID =
Customers.CustomerID
  INNER JOIN OrderDetails OD ON OD.OrderID = O.OrderID
  INNER JOIN Dishes ON Dishes.DishID = OD.DishID
  LEFT JOIN OrderDiscounts ON OrderDiscounts.OrderID = OD.OrderID
  LEFT JOIN Discounts ON Discounts.DiscountID = OrderDiscounts.DiscountID
  WHERE O.CustomerID = @CustomerID AND MONTH(OrderDate) = @Month AND
YEAR(OrderDate) = @Year
  GROUP BY OD.OrderID, O.CustomerID, CompanyName, Address, PostalCode, City,
Country, OrderDate, DishName, Quantity, Discounts.Value, OD.UnitPrice
)
```

Przykład użycia:

```
SELECT * FROM GenerateCollectiveInvoice(4, 'Restaurant', 'Address', 1, 2021)
```

Widoki

ToDoOrders

Widok zwracający listę zamówień, które należy jeszcze zrealizować.

```
DROP VIEW IF EXISTS ToDoOrders
GO
CREATE VIEW ToDoOrders
AS
```

```
SELECT * FROM Orders
WHERE GETDATE() < RequiredRealisationDate AND PickUpDate IS NULL
```

CurrentMenu

Widok wyświetlający aktualnie obowiązujące menu.

```
DROP VIEW IF EXISTS CurrentMenu
GO
CREATE VIEW CurrentMenu
AS
SELECT * FROM Menu
WHERE StartDate <= GETDATE() AND GETDATE() <= EndDate
```

UpcomingReservations

Widok zwracający listę rezerwacji na przyszłe terminy, oraz tych, które są realizowane obecnie. Dla każdej rezerwacji wyświetla numer zarezerwowanego stolika, ilość członków, datę dokonania rezerwacji oraz datę, na którą został zarezerwowany stolik.

```
DROP VIEW IF EXISTS UpcomingReservations
GO
CREATE VIEW UpcomingReservations
AS
SELECT ReservationID, Reservations.CustomerID,
      (CASE WHEN Customers.IsCompany = 1 THEN (CASE WHEN
CompanyCustomers.ContactPersonName IS NOT NULL THEN
CompanyCustomers.ContactPersonName ELSE CompanyCustomers.CompanyName END)
          ELSE IndividualCustomers.FirstName + ' ' +IndividualCustomers.LastName
END) AS ContactName,
      RealizationDateStart, RealizationDateEnd, NumberOfPeople, TableID, OrderID,
IsByPerson
FROM Reservations
INNER JOIN Customers ON Customers.CustomerID = Reservations.CustomerID
INNER JOIN CompanyCustomers ON Customers.CustomerID = CompanyCustomers.CustomerID
INNER JOIN IndividualCustomers ON IndividualCustomers.CustomerID =
Customers.CustomerID
WHERE IsCancelled = 0 AND RealizationDateStart >= GETDATE()
```

WeekReservationsReport

Widok zwracający raport dotyczący rezerwacji stolików z bieżącego tygodnia.

```
DROP VIEW IF EXISTS WeekReservationsReport
GO
CREATE VIEW WeekReservationsReport
AS
SELECT ReservationID, Reservations.CustomerID,
```

```
      (CASE WHEN Customers.IsCompany = 1 THEN (CASE WHEN
CompanyCustomers.ContactPersonName IS NOT NULL THEN
CompanyCustomers.ContactPersonName ELSE CompanyCustomers.CompanyName
END)
          ELSE IndividualCustomers.FirstName + ' '
+IndividualCustomers.LastName END) AS ContactName,
      RealizationDateStart, RealizationDateEnd, NumberOfPeople, TableID,
OrderID, IsByPerson
FROM Reservations
INNER JOIN Customers ON Customers.CustomerID = Reservations.CustomerID
INNER JOIN CompanyCustomers ON Customers.CustomerID =
CompanyCustomers.CustomerID
INNER JOIN IndividualCustomers ON IndividualCustomers.CustomerID =
Customers.CustomerID
WHERE DATEPART(WEEK,RealizationDateStart) = DATEPART(WEEK,GETDATE())
 AND YEAR(GETDATE()) = YEAR(RealizationDateStart)
 AND MONTH(RealizationDateStart) = MONTH(GETDATE())
```

MonthReservationsReport

Widok zwracający raport dotyczący rezerwacji stolików z bieżącego miesiąca.

```
DROP VIEW IF EXISTS MonthReservationsReport
GO
CREATE VIEW MonthReservationsReport
AS
SELECT ReservationID, Reservations.CustomerID,
      (CASE WHEN Customers.IsCompany = 1 THEN (CASE WHEN
CompanyCustomers.ContactPersonName IS NOT NULL THEN
CompanyCustomers.ContactPersonName ELSE CompanyCustomers.CompanyName END)
          ELSE IndividualCustomers.FirstName + ' ' +IndividualCustomers.LastName
END) AS ContactName,
      RealizationDateStart, RealizationDateEnd, NumberOfPeople, TableID, OrderID,
IsByPerson
FROM Reservations
INNER JOIN Customers ON Customers.CustomerID = Reservations.CustomerID
INNER JOIN CompanyCustomers ON Customers.CustomerID = CompanyCustomers.CustomerID
INNER JOIN IndividualCustomers ON IndividualCustomers.CustomerID =
Customers.CustomerID
WHERE MONTH(RealizationDateStart) = MONTH(GETDATE()) AND YEAR(GETDATE()) =
YEAR(RealizationDateStart)
```

WeekDiscountsReport

Widok zwracający raport dotyczący rabatów z bieżącego tygodnia.

```
DROP VIEW IF EXISTS WeekDiscountsReport
```

```
GO
CREATE VIEW WeekDiscountsReport
AS
SELECT DiscountID, Value, Customers.CustomerID,
      (CASE WHEN Customers.IsCompany = 1 THEN
CompanyCustomers.CompanyName
          ELSE IndividualCustomers.FirstName + ' '
+IndividualCustomers.LastName END) AS CustomerName,
      IssueDate, DueDate, IsOneTime, IsAvailable
FROM Discounts
INNER JOIN Customers ON Customers.CustomerID = Discounts.CustomerID
INNER JOIN CompanyCustomers ON Customers.CustomerID =
CompanyCustomers.CustomerID
INNER JOIN IndividualCustomers ON IndividualCustomers.CustomerID =
Customers.CustomerID
WHERE DATEPART(WEEK,IssueDate) = DATEPART(WEEK,GETDATE())
 AND MONTH(IssueDate) = MONTH(GETDATE())
 AND YEAR(GETDATE()) = YEAR(IssueDate)
```

MonthDiscountsReport

Widok zwracający raport dotyczący rabatów z bieżącego miesiąca.

```
DROP VIEW IF EXISTS MonthDiscountsReport
GO
CREATE VIEW MonthDiscountsReport
AS
SELECT DiscountID, Value, Customers.CustomerID,
      (CASE WHEN Customers.IsCompany = 1 THEN CompanyCustomers.CompanyName
          ELSE IndividualCustomers.FirstName + ' ' +IndividualCustomers.LastName
END) AS CustomerName,
      IssueDate, DueDate, IsOneTime, IsAvailable
FROM Discounts
INNER JOIN Customers ON Customers.CustomerID = Discounts.CustomerID
INNER JOIN CompanyCustomers ON Customers.CustomerID =
CompanyCustomers.CustomerID
INNER JOIN IndividualCustomers ON IndividualCustomers.CustomerID =
Customers.CustomerID
WHERE MONTH(IssueDate) = MONTH(GETDATE()) AND YEAR(GETDATE()) = YEAR(IssueDate)
```

WeekMenuReport

Widok zwracający raport dotyczący menu z bieżącego tygodnia.

```
DROP VIEW IF EXISTS WeekMenuReport
GO
CREATE VIEW WeekMenuReport
```

```
AS
SELECT M.MenuID, ArrangementDate, StartDate, EndDate, DishName,
UnitPrice, MD.IsAvailable
   FROM Menu M
   INNER JOIN MenuDishes MD ON M.MenuID = MD.MenuID
   INNER JOIN Dishes D ON D.DishID = MD.DishID
   WHERE  DATEPART(WEEK,StartDate) = DATEPART(WEEK,GETDATE())
     AND MONTH(StartDate) = MONTH(GETDATE())
     AND YEAR(GETDATE()) = YEAR(StartDate)
```

MonthMenuReport

Widok zwracający raport dotyczący menu z bieżącego miesiąca.

```
DROP VIEW IF EXISTS MonthMenuReport
GO
CREATE VIEW MonthMenuReport
AS
SELECT M.MenuID, ArrangementDate, StartDate, EndDate, DishName,
UnitPrice, MD.IsAvailable
   FROM Menu M
   INNER JOIN MenuDishes MD ON M.MenuID = MD.MenuID
   INNER JOIN Dishes D ON D.DishID = MD.DishID
   WHERE MONTH(StartDate) = MONTH(GETDATE()) AND YEAR(GETDATE()) =
YEAR(StartDate)
```

WeekOrderReport

Widok zwracający raport dotyczący zamówień z bieżącego tygodnia.

```
DROP VIEW IF EXISTS WeekOrderReport
GO
CREATE VIEW WeekOrderReport
AS
SELECT Orders.OrderID, OrderDate, RequiredRealisationDate, PickUpDate,
IsTakeAway,
      SUM(OrderDetails.UnitPrice * OrderDetails.Quantity * (1 -
ISNULL(Discounts.Value, 0))) AS TotalPrice
FROM Orders
INNER JOIN OrderDetails ON Orders.OrderID = OrderDetails.OrderID
LEFT JOIN OrderDiscounts ON Orders.OrderID = OrderDiscounts.OrderID
LEFT JOIN Discounts ON OrderDiscounts.DiscountID = Discounts.DiscountID
WHERE DATEPART(WEEK,OrderDate) = DATEPART(WEEK,GETDATE())
 AND MONTH(OrderDate) = MONTH(GETDATE())
 AND YEAR(GETDATE()) = YEAR(OrderDate)
GROUP BY Orders.OrderID, OrderDate, RequiredRealisationDate, PickUpDate,
IsTakeAway
```

MonthOrderReport

Widok zwracający raport dotyczący zamówień z bieżącego miesiąca.

```
DROP VIEW IF EXISTS MonthOrderReport
GO
CREATE VIEW MonthOrderReport
AS
SELECT Orders.OrderID, OrderDate, RequiredRealisationDate, PickUpDate,
IsTakeAway,
       SUM(OrderDetails.UnitPrice * OrderDetails.Quantity * (1 -
ISNULL(Discounts.Value, 0))) AS TotalPrice
FROM Orders
INNER JOIN OrderDetails ON Orders.OrderID = OrderDetails.OrderID
LEFT JOIN OrderDiscounts ON Orders.OrderID = OrderDiscounts.OrderID
LEFT JOIN Discounts ON OrderDiscounts.DiscountID = Discounts.DiscountID
WHERE MONTH(OrderDate) = MONTH(GETDATE()) AND YEAR(GETDATE()) =
YEAR(OrderDate)
GROUP BY Orders.OrderID, OrderDate, RequiredRealisationDate, PickUpDate,
IsTakeAway
```

OccupiedTablesNow

Widok zwracający numery aktualnie zarezerwowanych stolików.

```
DROP VIEW IF EXISTS OccupiedTablesNow
GO
CREATE VIEW OccupiedTablesNow
AS
SELECT T.TableID, T.MaxCapacity, T.CurrentCapacity FROM Tables T
INNER JOIN Reservations R ON T.TableID = R.TableID
WHERE RealizationDateStart <= GETDATE() AND GETDATE() <=
RealizationDateEnd
```

FreeTablesNow

Widok zwracający numery aktualnie dostępnych stolików.

```
DROP VIEW IF EXISTS FreeTablesNow
GO
CREATE VIEW FreeTablesNow
AS
SELECT T.TableID, T.MaxCapacity, T.CurrentCapacity FROM Tables T
LEFT JOIN Reservations R ON T.TableID = R.TableID
WHERE RealizationDateStart > GETDATE() OR RealizationDateEnd < GETDATE()
OR RealizationDateStart IS NULL
```

UnavailableProducts

Widok zwracający listę aktualnie brakujących produktów.

```
DROP VIEW IF EXISTS UnavailableProducts
GO
CREATE VIEW UnavailableProducts
AS
SELECT * FROM Products
WHERE UnitsInStock = 0
```

ExhaustingProducts

Widok zwracający listę produktów na wyczerpaniu, czyli których liczba jednostek na magazynie jest mniejsza od 10.

```
DROP VIEW IF EXISTS ExhaustingProducts
GO
CREATE VIEW ExhaustingProducts
AS
SELECT * FROM Products
WHERE UnitsInStock < 10
```

Triggery

ChangeMenuDishAvailabilityTrigger

Zmienia dostępność dania w menu po zmianie dostępności dania ogólnie.

```
DROP TRIGGER IF EXISTS ChangeMenuDishAvailabilityTrigger
GO
CREATE TRIGGER ChangeMenuDishAvailabilityTrigger
ON Dishes
FOR UPDATE AS
BEGIN
  IF UPDATE(isAvailable)
  BEGIN
     DECLARE @DishID AS INT
       SET @DishID = (SELECT DishID FROM INSERTED)

       UPDATE MenuDishes
       SET isAvailable = (SELECT isAvailable FROM INSERTED)
       WHERE DishID = @DishID
  END
END
GO
```

MakeUnavailableDish

Czyni danie niedostępnym, jeżeli jakiś z jego składników się wyczerpał.

```
DROP TRIGGER IF EXISTS MakeUnavailableDish
GO
CREATE TRIGGER MakeUnavailableDish
ON Products
FOR UPDATE
AS
BEGIN
  IF UPDATE(UnitsInStock)
  BEGIN
     IF (SELECT UnitsInStock FROM INSERTED) = 0
     BEGIN
        DECLARE @ProductID AS INT
        SET @ProductID = (SELECT i.ProductID FROM INSERTED AS i)
        DECLARE @DishID AS INT
        SET @DishID = (SELECT DishID FROM DishDetails WHERE ProductID =
@ProductID)

        UPDATE Dishes
        SET isAvailable = 0
        WHERE DishID = @DishID
     END
  END
END
```

ChangeMenuEndDate

Zmienia datę zakończenia menu, jeśli została ona źle podana (czyli jest ona większa niż 2 tygodnie od daty rozpoczęcia menu).

```
DROP TRIGGER IF EXISTS ChangeMenuEndDate
GO
CREATE TRIGGER ChangeMenuEndDate
ON Menu
FOR INSERT, UPDATE AS
BEGIN
  UPDATE Menu
  SET EndDate = DATEADD(WEEK, 2, StartDate)
  WHERE EndDate > DATEADD(WEEK, 2, StartDate)
END
```

CanOrderSeafood

Sprawdza, czy klient składający zamówienie na owoce morza zrobił to w odpowiednim czasie.

```
DROP TRIGGER IF EXISTS CanOrderSeafood
GO
CREATE TRIGGER CanOrderSeafood
ON Orders
FOR INSERT
AS
BEGIN
  DECLARE @DishID AS INT
  SET @DishID = (SELECT DishID FROM OrderDetails
              INNER JOIN INSERTED i ON i.OrderID = OrderDetails.OrderID)
  DECLARE @ProductID AS INT
  SET @ProductID = (SELECT DishDetails.ProductID FROM DishDetails
                 INNER JOIN Products ON DishDetails.ProductID =
Products.ProductID
                 INNER JOIN ProductCategories ON
ProductCategories.ProductCategoryID = Products.ProductCategoryID
                 WHERE DishID = @DishID AND ProductCategoryName = 'Seafood'
                  )
  IF @ProductID IS NOT NULL
  BEGIN
     IF DATEPART(WEEKDAY, (SELECT OrderDate FROM INSERTED)) NOT IN (5, 6, 7)
     BEGIN
        RAISERROR ('Seafood were ordered in a wrong time', -1, -1)
        ROLLBACK TRANSACTION
     END
  END
END
```

IndividualCustomerTrigger

Ustawia wartość pola IsCompany na 0 po dodawaniu nowego klienta indywidualnego do bazy danych.

```
DROP TRIGGER IF EXISTS IndividualCustomerTrigger
GO
CREATE TRIGGER IndividualCustomerTrigger
ON IndividualCustomers
FOR INSERT
AS
BEGIN
  UPDATE dbo.Customers
  SET IsCompany = 0
  WHERE CustomerID = (SELECT CustomerID FROM Inserted)
END
```

CompanyCustomerTrigger

Ustawia wartość pola IsCompany na 1 po dodawaniu nowego klienta firmowego do bazy danych.

```
DROP TRIGGER IF EXISTS CompanyCustomerTrigger
GO
CREATE TRIGGER CompanyCustomerTrigger
ON CompanyCustomers
FOR INSERT
AS
BEGIN
  UPDATE dbo.Customers
  SET IsCompany = 1
  WHERE CustomerID = (SELECT CustomerID FROM Inserted)
END
```

Indeksy

- Nazwisko w tabeli klientów indywidualnych.

```
CREATE INDEX CustomerLastName ON IndividualCustomers(LastName)
```

- Nazwa firmy w tabeli klientów firmowych.

```
CREATE INDEX CustomerCompanyName ON CompanyCustomers(CompanyName)
```

- Email w tabeli klientów.

```
CREATE INDEX CustomerEmail ON Customers(Email)
```

- Numer telefonu w tabeli klientów.

```
CREATE INDEX CustomerPhone ON Customers(Phone)
```

- ID klienta w tabeli rabatów.

```
CREATE INDEX RabatCustomer ON Discounts(CustomerID)
```

- ID klienta w tabeli rezerwacji.

```
CREATE INDEX ReservationCustomer ON Reservations(CustomerID)
```

- ID pracownika w tabeli rezerwacji.

```
CREATE INDEX ReservationEmployee ON Reservations(EmployeeID)
```

- ID klienta w tabeli zamówień.

```
CREATE INDEX OrderCustomer ON Orders(CustomerID)
```

- ID pracownika w tabeli zamówień.

```
CREATE INDEX OrderEmployee ON Orders(EmployeeID)
```

- Nazwa produktu w tabeli produktów.

```
CREATE INDEX ProductNameIndex ON Products(ProductName)
```

- Nazwa dania w tabeli dań.

```
CREATE INDEX DishNameIndex ON Dishes(DishName)
```

- Nazwa kategorii produktu w tabeli ProductCategories

```
CREATE INDEX ProductCategoryIndex ON ProductCategories(ProductCategoryName)
```

Role

Kierownik zamówień

Kierownik zamówień ma uprawnienia do następujących procedur:

- AddNewCompanyCustomer
- AddNewIndividualCustomer
- AddNewOrder
- AddNewOrderDetails
- AddNewDiscount
- AddNewReservation
- AddNewTable
- ChangeTableCurrentCapacity
- ChangeTablesLimit
- RemoveTablesLimits
- ChangeTableParticipants
- CancelReservation
- CancelOrder
- UseOneTimeDiscount
- DeactivateDiscount
- ChangeDiscountValue
- TryAssignNewDiscountToCompanyCustomer
- TryAssignNewDiscountToIndividualCustomer

Kierownik zamówień ma uprawnienia do następujących funkcji:

- FreeTables
- CustomerDiscounts
- CustomerOrdersNumberForValue
- CustomerOrdersNumber
- CustomerOrderWorth
- CustomerMonthOrdersNumber
- CustomerQuarterWorth
- CustomerHasNOrdersOfGivenValue
- CustomerMonthOrdersNumberOfGivenValue
- ReservationReport
- DiscountReport
- CustomerOrders
- CanMakeReservation

Kierownik zamówień ma uprawnienia do następujących widoków:

- ToDoOrders
- CurrentMenu
- UpcomingReservations
- WeekReservationsReport
- MonthReservationsReport
- WeekDiscountsReport
- MonthDiscountsReport
- WeekOrderReport
- MonthOrderReport
- OccupiedTablesNow
- FreeTablesNow

```
CREATE ROLE OrdersManager

GRANT EXECUTE ON AddNewCompanyCustomer TO OrdersManager
GRANT EXECUTE ON AddNewIndividualCustomer TO OrdersManager
GRANT EXECUTE ON AddNewOrder TO OrdersManager
GRANT EXECUTE ON AddNewOrderDetails TO OrdersManager
GRANT EXECUTE ON AddNewDiscount TO OrdersManager
GRANT EXECUTE ON AddNewReservation TO OrdersManager
GRANT EXECUTE ON AddNewTable TO OrdersManager
GRANT EXECUTE ON ChangeTableCurrentCapacity TO OrdersManager
GRANT EXECUTE ON ChangeTablesLimit TO OrdersManager
GRANT EXECUTE ON RemoveTablesLimits TO OrdersManager
GRANT EXECUTE ON ChangeTableParticipants TO OrdersManager
GRANT EXECUTE ON CancelReservation TO OrdersManager
GRANT EXECUTE ON CancelOrder TO OrdersManager
GRANT EXECUTE ON UseOneTimeDiscount TO OrdersManager
GRANT EXECUTE ON DeactivateDiscount TO OrdersManager
GRANT EXECUTE ON ChangeDiscountValue TO OrdersManager
GRANT EXECUTE ON TryAssignNewDiscountToCompanyCustomer TO OrdersManager
GRANT EXECUTE ON TryAssignNewDiscountToIndividualCustomer TO OrdersManager

GRANT SELECT ON FreeTables TO OrdersManager
GRANT SELECT ON CustomerDiscounts TO OrdersManager
GRANT EXECUTE ON CustomerOrdersNumberForValue TO OrdersManager
GRANT EXECUTE ON CustomerOrdersNumber TO OrdersManager
GRANT EXECUTE ON CustomerOrderWorth TO OrdersManager
GRANT EXECUTE ON CustomerMonthOrdersNumber TO OrdersManager
GRANT EXECUTE ON CustomerQuarterWorth TO OrdersManager
GRANT EXECUTE ON CustomerHasNOrdersOfGivenValue TO OrdersManager
GRANT EXECUTE ON CustomerMonthOrdersNumberOfGivenValue TO OrdersManager
GRANT SELECT ON ReservationReport TO OrdersManager
GRANT SELECT ON DiscountReport TO OrdersManager
GRANT SELECT ON CustomerOrders TO OrdersManager
```

```
GRANT EXECUTE ON CanMakeReservation TO OrdersManager

GRANT SELECT ON ToDoOrders TO OrdersManager
GRANT SELECT ON CurrentMenu TO OrdersManager
GRANT SELECT ON UpcomingReservations TO OrdersManager
GRANT SELECT ON WeekReservationsReport TO OrdersManager
GRANT SELECT ON MonthReservationsReport TO OrdersManager
GRANT SELECT ON WeekDiscountsReport TO OrdersManager
GRANT SELECT ON MonthDiscountsReport TO OrdersManager
GRANT SELECT ON WeekOrderReport TO OrdersManager
GRANT SELECT ON MonthOrderReport TO OrdersManager
GRANT SELECT ON OccupiedTablesNow TO OrdersManager
GRANT SELECT ON FreeTablesNow TO OrdersManager
```

Kierownik menu (kierownik magazynu i szef kuchni)

Kierownik menu ma uprawnienia do następujących procedur:

- AddNewDish
- AddProductToDish
- AddNewProduct
- AddNewProductCategory
- AddDishToMenu
- AddNewMenu
- ChangeDishAvailability
- ChangeProductUnitsInStock
- ChangeMenuDishAvailability
- ChangeDishPrice

Kierownik menu ma uprawnienia do następujących funkcji:

- MenuReport
- MonthDishes

Kierownik menu ma uprawnienia do następujących widoków:

- CurrentMenu
- WeekMenuReport
- MonthMenuReport
- UnavailableProducts
- ExhaustingProducts

```
CREATE ROLE MenuManager

GRANT EXECUTE ON AddNewDish TO MenuManager
GRANT EXECUTE ON AddProductToDish TO MenuManager
GRANT EXECUTE ON AddNewProduct TO MenuManager
GRANT EXECUTE ON AddNewProductCategory TO MenuManager
GRANT EXECUTE ON AddDishToMenu TO MenuManager
GRANT EXECUTE ON AddNewMenu TO MenuManager
GRANT EXECUTE ON ChangeDishAvailability TO MenuManager
```

```
GRANT EXECUTE ON ChangeProductUnitsInStock TO MenuManager
GRANT EXECUTE ON ChangeMenuDishAvailability TO MenuManager
GRANT EXECUTE ON ChangeDishPrice TO MenuManager

GRANT SELECT ON MenuReport TO MenuManager
GRANT SELECT ON MonthDishes TO MenuManager

GRANT SELECT ON CurrentMenu TO MenuManager
GRANT SELECT ON WeekMenuReport TO MenuManager
GRANT SELECT ON MonthMenuReport TO MenuManager
GRANT SELECT ON UnavailableProducts TO MenuManager
GRANT SELECT ON ExhaustingProducts TO MenuManager
```

Księgowa

Księgowa ma uprawnienia do następujących procedur:

- AddNewDiscount
- UseOneTimeDiscount
- DeactivateDiscount
- ChangeDiscountValue
- TryAssignNewDiscountToCompanyCustomer
- TryAssignNewDiscountToIndividualCustomer

Księgowa ma uprawnienia do następujących funkcji:

- CustomerDiscounts
- CustomerOrdersNumberForValue
- CustomerOrdersNumber
- CustomerOrderWorth
- CustomerMonthOrdersNumber
- CustomerQuarterWorth
- CustomerHasNOrdersOfGivenValue
- CustomerMonthOrdersNumberOfGivenValue
- ReservationReport
- DiscountReports
- MonthMenuReport
- WeekOrderReport
- MonthOrderReport
- CustomerOrders
- GenerateInvoice
- GenerateCollectiveInvoice

Księgowa ma uprawnienia do następujących widoków:

- WeekReservationsReport
- MonthReservationsReport
- WeekDiscountsReport

- MonthDiscountsReport
- WeekOrdersReport
- MonthOrdersReport

```
CREATE ROLE Accountant

GRANT EXECUTE ON AddNewDiscount TO Accountant
GRANT EXECUTE ON UseOneTimeDiscount TO Accountant
GRANT EXECUTE ON DeactivateDiscount TO Accountant
GRANT EXECUTE ON ChangeDiscountValue TO Accountant
GRANT EXECUTE ON TryAssignNewDiscountToCompanyCustomer TO Accountant
GRANT EXECUTE ON TryAssignNewDiscountToIndividualCustomer TO Accountant

GRANT SELECT ON CustomerDiscounts TO Accountant
GRANT EXECUTE ON CustomerOrdersNumberForValue TO Accountant
GRANT EXECUTE ON CustomerOrdersNumber TO Accountant
GRANT EXECUTE ON CustomerOrderWorth TO Accountant
GRANT EXECUTE ON CustomerMonthOrdersNumber TO Accountant
GRANT EXECUTE ON CustomerQuarterWorth TO Accountant
GRANT EXECUTE ON CustomerHasNOrdersOfGivenValue TO Accountant
GRANT EXECUTE ON CustomerMonthOrdersNumberOfGivenValue TO Accountant
GRANT SELECT ON ReservationReport TO Accountant
GRANT SELECT ON DiscountReport TO Accountant
GRANT SELECT ON CustomerOrders TO Accountant
GRANT SELECT ON GenerateInvoice TO Accountant
GRANT SELECT ON GenerateCollectiveInvoice TO Accountant

GRANT SELECT ON WeekReservationsReport TO Accountant
GRANT SELECT ON MonthReservationsReport TO Accountant
GRANT SELECT ON WeekDiscountsReport TO Accountant
GRANT SELECT ON MonthDiscountsReport TO Accountant
GRANT SELECT ON WeekOrderReport TO Accountant
GRANT SELECT ON MonthOrderReport TO Accountant
```

Administrator bazy danych

Administrator bazy danych ma dostęp do wszystkich zdefiniowanych procedur, funkcji i widoków.

```
CREATE ROLE DataBaseAdmin

GRANT EXECUTE, ALTER ON AddNewCustomer TO DataBaseAdmin
GRANT EXECUTE, ALTER ON AddNewCompanyCustomer TO DataBaseAdmin
GRANT EXECUTE, ALTER ON AddNewIndividualCustomer TO DataBaseAdmin
GRANT EXECUTE, ALTER ON AddNewEmployee TO DataBaseAdmin
GRANT EXECUTE, ALTER ON AddNewOrder TO ODataBaseAdmin
GRANT EXECUTE, ALTER ON AddNewOrderDetails TO DataBaseAdmin
GRANT EXECUTE, ALTER ON AddNewDish TO DataBaseAdmin
```

```
GRANT EXECUTE, ALTER ON AddProductToDish TO DataBaseAdmin
GRANT EXECUTE, ALTER ON AddNewProduct TO DataBaseAdmin
GRANT EXECUTE, ALTER ON AddNewProductCategory TO DataBaseAdmin
GRANT EXECUTE, ALTER ON AddNewDiscount TO DataBaseAdmin
GRANT EXECUTE, ALTER ON AddNewReservation TO DataBaseAdmin
GRANT EXECUTE, ALTER ON AddDishToMenu TO DataBaseAdmin
GRANT EXECUTE, ALTER ON AddNewMenu TO DataBaseAdmin
GRANT EXECUTE, ALTER ON AddNewTable TO DataBaseAdmin
GRANT EXECUTE, ALTER ON ChangeTableCurrentCapacity TO DataBaseAdmin
GRANT EXECUTE, ALTER ON ChangeTablesLimit TO DataBaseAdmin
GRANT EXECUTE, ALTER ON RemoveTablesLimits TO DataBaseAdmin
GRANT EXECUTE, ALTER ON ChangeTableParticipants TO DataBaseAdmin
GRANT EXECUTE, ALTER ON ChangeCustomerData TO DataBaseAdmin
GRANT EXECUTE, ALTER ON ChangeCompanyCustomerData TO DataBaseAdmin
GRANT EXECUTE, ALTER ON ChangeIndividualCustomerData TO DataBaseAdmin
GRANT EXECUTE, ALTER ON ChangeEmployeeData TO DataBaseAdmin
GRANT EXECUTE, ALTER ON RemoveEmployee TO DataBaseAdmin
GRANT EXECUTE, ALTER ON CancelReservation TO DataBaseAdmin
GRANT EXECUTE, ALTER ON CancelOrder TO DataBaseAdmin
GRANT EXECUTE, ALTER ON ChangeDishAvailability TO DataBaseAdmin
GRANT EXECUTE, ALTER ON ChangeProductUnitsInStock TO DataBaseAdmin
GRANT EXECUTE, ALTER ON ChangeMenuDishAvailability TO DataBaseAdmin
GRANT EXECUTE, ALTER ON ChangeDishPrice TO DataBaseAdmin
GRANT EXECUTE, ALTER ON UseOneTimeDiscount TO DataBaseAdmin
GRANT EXECUTE, ALTER ON DeactivateDiscount TO DataBaseAdmin
GRANT EXECUTE, ALTER ON ChangeDiscountValue TO DataBaseAdmin
GRANT EXECUTE, ALTER ON TryAssignNewDiscountToCompanyCustomer TO DataBaseAdmin
GRANT EXECUTE, ALTER ON TryAssignNewDiscountToIndividualCustomer TO DataBaseAdmin

GRANT SELECT, ALTER ON FreeTables TO DataBaseAdmin
GRANT SELECT, ALTER ON CustomerDiscounts TO DataBaseAdmin
GRANT EXECUTE, ALTER ON CustomerOrdersNumberForValue TO DataBaseAdmin
GRANT EXECUTE, ALTER ON CustomerOrdersNumber TO DataBaseAdmin
GRANT EXECUTE, ALTER ON CustomerOrderWorth TO DataBaseAdmin
GRANT EXECUTE, ALTER ON CustomerMonthOrdersNumber TO DataBaseAdmin
GRANT EXECUTE, ALTER ON CustomerQuarterWorth TO DataBaseAdmin
GRANT EXECUTE, ALTER ON CustomerHasNOrdersOfGivenValue TO DataBaseAdmin
GRANT EXECUTE, ALTER ON CustomerMonthOrdersNumberOfGivenValue TO DataBaseAdmin
GRANT SELECT, ALTER ON ReservationReport TO DataBaseAdmin
GRANT SELECT, ALTER ON MenuReport TO DataBaseAdmin
GRANT SELECT, ALTER ON DiscountReport TO DataBaseAdmin
GRANT SELECT, ALTER ON CustomerOrders TO DataBaseAdmin
GRANT SELECT, ALTER ON MonthDishes TO DataBaseAdmin
GRANT EXECUTE, ALTER ON CanMakeReservation TO DataBaseAdmin
GRANT SELECT, ALTER ON GenerateInvoice TO DataBaseAdmin
GRANT SELECT, ALTER ON GenerateCollectiveInvoice TO DataBaseAdmin
```

```
GRANT SELECT, ALTER ON ToDoOrders TO DataBaseAdmin
GRANT SELECT, ALTER ON CurrentMenu TO DataBaseAdmin
GRANT SELECT, ALTER ON UpcomingReservations TO DataBaseAdmin
GRANT SELECT, ALTER ON WeekReservationsReport TO DataBaseAdmin
GRANT SELECT, ALTER ON MonthReservationsReport TO DataBaseAdmin
GRANT SELECT, ALTER ON WeekDiscountsReport TO DataBaseAdmin
GRANT SELECT, ALTER ON MonthDiscountsReport TO DataBaseAdmin
GRANT SELECT, ALTER ON WeekMenuReport TO DataBaseAdmin
GRANT SELECT, ALTER ON MonthMenuReport TO DataBaseAdmin
GRANT SELECT, ALTER ON WeekOrderReport TO DataBaseAdmin
GRANT SELECT, ALTER ON MonthOrderReport TO DataBaseAdmin
GRANT SELECT, ALTER ON OccupiedTablesNow TO DataBaseAdmin
GRANT SELECT, ALTER ON FreeTablesNow TO DataBaseAdmin
GRANT SELECT, ALTER ON UnavailableProducts TO DataBaseAdmin
GRANT SELECT, ALTER ON ExhaustingProducts TO DataBaseAdmin
```

Klient

Klient ma dostęp do następujących procedur:

- CancelReservation
- CancelOrder

Klient ma dostęp do następujących widoków:

- CurrentMenu

```
CREATE ROLE Customer

GRANT EXECUTE ON CancelReservation TO Customer
GRANT EXECUTE ON CancelOrder TO Customer

GRANT SELECT ON CurrentMenu TO Customer
```

Generowanie danych

Przykładowe dane wygenerowano za pomocą generatora danych SQL Data Generator firmy **RedGate**:
https://www.red-gate.com/products/sql-development/sql-data-generator/?fbclid=IwAR1IeAiRBR0nrtpBIBd9fqCLIhTNutnKkp1GzBx-nba9QVlo3hQyJut992s .

Z tych danych powstały tabele, z których dane były łączone losowo w potrzebne w naszej bazie rekordy.