Podstawy Baz Danych
Dokumentacja

Andrzej Szarata, Krzysztof Pęczek, Szymon Bednorz

Role
- Administrator
- Kierownik
- Klient Indywidualny
- Klient firma
- Pracownik Restauracji

Opis funkcji systemu

Tabele z obszaru Products
- Tabela Categories
- Tabela Products
- Tabela Menu

Tabele z obszaru Employees
- Tabela Employees
- Tabela Managers
- Tabela Administrators

Tabele z obszaru Order Management
- Tabela Orders
- Tabela OrderDetails
- Tabela Takeaway
- Tabela TableDetails
- Tabela Tables

Tabele z obszaru Reservations
- Tabela Client
- Tabela Companies
- Tabela CompanyReservations
- Tabela CompanyReservationsDetails
- Tabela IndividualReservations
- Tabela Individuals
- Tabela Reservations
- Tabela Discounts

Tabele z obszaru Parameters
- Tabela DiscountParameters
- Tabela Parameters

Widok ClientStats

Widok CompanyOrders

Widok IndividualOrders

Widok IndividualReservationsHistory

Widok CurrentDiscounts

Widok CurrentlyFreeTables

Widok CurrentMenu

Widok EmployeeStats

Widok GeneralProductStats

Widok MonthlyOrderStats

Widok WeeklyOrderStats

Widok MonthlyProductStats

Widok ProductDetails

Widok ReservationsToCheck

Widok TableReservationDetails

Widok UpcommingTakeawayOrders

Widok UpcommingReservations

Widok CompanyReservationsHistory

Widok MonthlyCompanyOrders

Widok SeaFoodOrders

Widok AverageServiceTime

Widok AverageProductServingTime

Widok OrderHistory

CanOrderSeafood

GetPreviousMonday

HasSeafood

CanMakeReservation

GetCurrentDiscount

VerifyMenu

GetOrderValue

GetOrderDetails

AddCategory

AddProduct

AddToMenu

AddIndividual

AddComapny

AddIndividualReservation

AddNamedCompanyReservation

AddAnonymousCompanyReservation

AcceptReservation

AddEmployee

MakeManager

MakeAdministrator

AddTable

AddTableToReservation

AddOrder

MakeTakeaway

ChangeParameter

UpdateMenu

Trigger RemoveAllOrderDetailsTRIGGER

Trigger RemoveReservationDetailsTRIGGER

Trigger RemoveOrderFromReservationTRIGGER

Indeksy tabeli Client

Indeksy tabeli Individuals

Indeksy tabeli Reservations

Indeksy tabeli CompanyReservations

Indeksy tabeli CompanyReservationsDetails

Indeksy tabeli IndividualReservations

Indeksy tabeli Employees

Indeksy tabeli Orders

Indeksy tabeli OrderDetails

Indeksy tabeli Takeaway

Indeksy tabeli TableDetails

Indeksy tabeli Tables

Indeksy tabeli Menu

Indeksy tabeli Products

Administrator

Customer

IndividualCustomer

CompanyCustomer

Employee

Manager

Role

- Administrator
- Kierownik
- Klient Indywidualny
- Klient firma
- Pracownik Restauracji

Administrator

- Posiada dostęp do danych użytkowników
- Może dowolnie tworzyć, edytować konta klientów indywidualnych, firm oraz pracowników restauracji.
- Może tworzyć, edytować oraz usuwać rezerwacje
- Ma możliwość edycji menu
- Może generować dowolne raporty

Kierownik

- Możliwość zmian i zatwierdzania pozycji w menu
- Posiada dostęp do aktualnych rezerwacji oraz historii rezerwacji.
- Ma możliwość zmian kwot oraz wymogów zniżek
- Ma możliwość generowania dowolnych raportów

Klient Indywidualny

- Podaje dane osobowe podczas rejestracji
- Po rejestracji ma możliwość zmiany niektórych danych osobowych (np. Adres)
- Posiada dostęp tylko do własnych danych
- Klient ma możliwość złożenia rezerwacji stolika na co najmniej dwie osoby jeśli spełnione są warunki:
    - Klient złożył co najmniej **WK** zamówień
    - Wartość zamówienia przy rezerwacji wniosła co najmniej **WZ** zł .
- Podczas tworzenia rezerwacji ma dostęp do informacji o dostępności stolików
- Posiada możliwość odczytania aktualnego menu
- Może przeglądać informacje na temat własnych rezerwacji (nadchodzących oraz historii)
- Może złożyć zamówienie
    - Na miejscu
    - Na wynos na miejscu
    - Na wynos z wykorzystaniem formularza WWW
- Może złożyć zamówienie stałe w formie cateringu i odebrać je w porze lunchu (bez dostaw)
- Może anulować swoje rezerwacje

Klient firma

- Może dokonać rejestracji konta firmy
- Po rejestracji może zmieniać niektóre dane firmy
- Może dokonać rezerwacji na firmę
- Może dokonać rezerwacji na pracownika firmy (bez wymogów dla klienta indywidualnego)
- Może odczytywać aktualne menu
- Może złożyć zamówienie stałe w formie cateringu do odebrania przez pracowników w porze lunchu
- Może przeglądać informacje na temat własnych rezerwacji
- Może anulować rezerwację

Pracownik Restauracji

- Przyjmuje zamówienia klientów na miejscu
- Akceptacja płatności klientów
- Akceptacja informacji o rezerwacji zamówień/stolika
- Posiada dostęp do aktualnych rezerwacji
- Posiada dostęp do informacji o dostępności stolików
- Generowanie faktur dla danego zamówienia lub faktury zbiorczej raz na miesiąc (na życzenie klienta)
- Dostęp do informacji o stanie magazynu

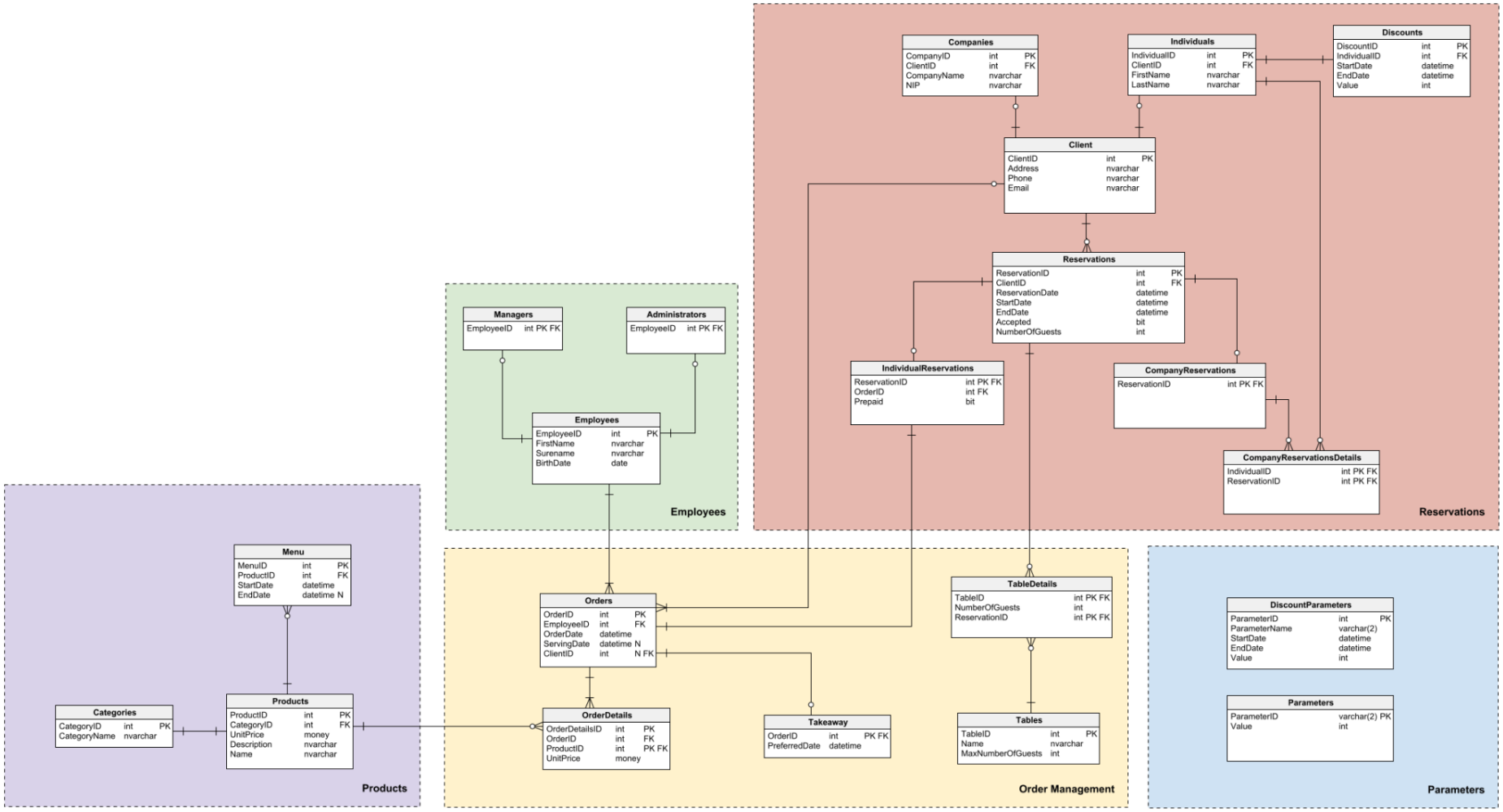
Opis funkcji systemu

- Dodawanie zamówienia do bazy danych na miejscu (przez pracownika)
- Generowanie formularza www umożliwiający złożenie zamówienia przez klienta
- Możliwość rezerwacji wolnego stolika:
    - dla klienta indywidualnego:
        - stolik musi być zarezerwowany dla przynajmniej dwóch osób
        - przy rezerwację stolika należy złożyć zamówienia
        - Istnieje określona minimalna wartość zamówienia wz
        - aby móc złożyć rezerwację klient musiał wcześniej dokonać przynajmniej WK zamówień.
        - rezerwacja stolika musi zostać zatwierdzona przez obsługę
    - dla firmy:
        - rezerwacja stolików może zostać stworzona na firmę
        - rezerwacja stolików dla konkretnych pracowników firmy

- ustalenie menu:
    - Menu jest ustalane co najmniej z dziennym wyprzedzeniem
    - Co najmniej połowa pozycji jest zmieniana raz na dwa tygodnie

    - Owoce morza można zamawiać na czwartki, piątki i soboty z wcześniejszym zamówieniem przed poniedziałkiem.
    - wszystkie decyzje dotyczące menu podejmuje kierownik
- system posiada funkcję rabatów dla klientów indywidualnych
    - po realizacji Z1 zamówień za co najmniej K1 złotych pojawia się R1% zniżki na wszystkie zamówienia
    - po realizacji zamówienia za łączną kwotę K2 zostaje przyznana jednorazowa zniżka R2% na wszystkie zamówienia złożone przez D1 dni od dnia przyznania zniżki
    - zniżki są przyznawane automatycznie przez system
    - wymogi oraz kwoty zniżek ustala kierownik
- generowanie raportów miesięcznych i tygodniowych
    - raporty dotyczące rezerwacji stolików
    - raporty dotyczące rabatów
    - raporty dotyczące menu
    - dla klientów indywidualnych oraz firm raporty dotyczące kwot oraz czasu składania zamówień
    - raporty może generować tylko kierownik i administratorzy.

Schemat


Companies
CompanyID int PK
ClientID int FK
CompanyName nvarchar
NIP nvarchar
Individuals
IndividualID int PK
ClientID int FK
FirstName nvarchar
LastName nvarchar
Discounts
DiscountID int PK
IndividualID int FK
StartDate datetime
EndDate datetime
Value int
Client
ClientID int PK
Address nvarchar
Phone nvarchar
Email nvarchar
Reservations
ReservationID int PK
ClientID int FK
ReservationDate datetime
StartDate datetime
EndDate datetime
Accepted bit
NumberOfGuests int
IndividualReservations
ReservationID int PK FK
OrderID int FK
Prepaid bit
CompanyReservations
ReservationID int PK FK
CompanyReservationsDetails
IndividualID int PK FK
ReservationID int PK FK
Reservations
Managers
EmployeeID int PK FK
Administrators
EmployeeID int PK FK
Employees
EmployeeID int PK
FirstName nvarchar
Surename nvarchar
BirthDate date
Employees
Menu
MenuID int PK
ProductID int FK
StartDate datetime
EndDate datetime N
Categories
CategoryID int PK
CategoryName nvarchar
Products
ProductID int PK
CategoryID int FK
UnitPrice money
Description nvarchar
Name nvarchar
Products
Orders
OrderID int PK
EmployeeID int FK
OrderDate datetime
ServingDate datetime N
ClientID int N FK
OrderDetails
OrderDetailsID int PK
OrderID int FK
ProductID int PK FK
UnitPrice money
Takeaway
OrderID int PK FK
PreferredDate datetime
TableDetails
TableID int PK FK
NumberOfGuests int
ReservationID int PK FK
Tables
TableID int PK
Name nvarchar
MaxNumberOfGuests int
Order Management
DiscountParameters
ParameterID int PK
ParameterName varchar(2)
StartDate datetime
EndDate datetime
Value int
Parameters
ParameterID varchar(2) PK
Value int
Parameters

Tworzenie tabel

Tabele z obszaru Products

Tabela **Categories**

Klucz główny: CategoryID
Pola:

- CategoryName (nvachar, not null) - nazwa kategorii

```
CREATE TABLE Categories
(
   CategoryID int NOT NULL,
   CategoryName nvarchar NOT NULL,
   CONSTRAINT Categories_pk PRIMARY KEY  (CategoryID)
);
```

Tabela **Products**

Klucz główny: ProductID
Klucz obcy: CategoryID
Pola:

- UnitPrice (money, not null) - cena jednostkowa dania .
- Description (nvarchar, not null) - opis produktu. Domyślnie puste pole.
- Name (nvarchar, not null) - nazwa produktu.

Warunki integralności:

- UnitPrice większe od zera

```
CREATE TABLE Products
(
   ProductID int NOT NULL,
   CategoryID int NOT NULL,
   UnitPrice money NOT NULL,
   Name nvarchar NOT NULL,
   Description nvarchar NOT NULL DEFAULT '',
   CONSTRAINT ValidUnitPrice CHECK (UnitPrice > 0),
   CONSTRAINT Products_pk PRIMARY KEY  (ProductID)
);

ALTER TABLE Products ADD CONSTRAINT Products_Categories
FOREIGN KEY (CategoryID)
REFERENCES Categories (CategoryID);
```

Tabela **Menu**

Klucz Główny: MenuID
Klucz obcy: ProductID
Pola:

- StartDate (date, not null) data pojawienia się potrawy w menu. Domyślnie data wprowadzenia.
- EndDate (date, null) data zniknięcia potrawy z menu.

Warunki integralności:

- ValidDate - zapewnia, że StartDate jest wcześniejsze od EndDate.

```
CREATE TABLE Menu
(
   MenuID int NOT NULL,
   ProductID int NOT NULL,
```

```
   StartDate date NOT NULL DEFAULT GETDATE(),
   EndDate date NULL,
   CONSTRAINT Menu_pk PRIMARY KEY  (MenuID)
   CONSTRAINT ValidDate CHECK (StartDate < EndDate OR EndDate IS NULL),
);

ALTER TABLE Menu ADD CONSTRAINT Products_Menu
FOREIGN KEY (ProductID)
REFERENCES Products (ProductID);
```

Tabele z obszaru Employees

Tabela **Employees**

Klucz główny: EmployeeID
Kolumny:

- FirstName (nvarchar, not null) - Imie pracownika
- Surname (nvarchar, not null) - Nazwisko pracownika
- BirthDate (date, not null) - Data urodzin pracownika

Warunki integralnościowe:

- ValidBirthDate - Zapewnia, że BirthDate jest mniejszy niż obecna data

```
CREATE TABLE Employees
(
   EmployeeID int NOT NULL,
   FirstName nvarchar NOT NULL,
   Surename nvarchar NOT NULL,
   BirthDate date NOT NULL,
   CONSTRAINT ValidBirthDate CHECK (BirthDate < GETDATE()),
   CONSTRAINT Employees_pk PRIMARY KEY  (EmployeeID)
);
```

Tabela **Managers**

Lista managerow restauracji
Klucz główny: EmployeeID
Kolumny: -
Warunki integralnościowe: -

```
CREATE TABLE Managers
(
   EmployeeID int NOT NULL,
   CONSTRAINT Managers_pk PRIMARY KEY  (EmployeeID)
);

ALTER TABLE Managers ADD CONSTRAINT Employees_Managers
FOREIGN KEY (EmployeeID)
REFERENCES Employees (EmployeeID);
```

Tabela **Administrators**

Lista administratorów
Klucz główny: EmployeeID
Kolumny: -
Warunki integralnościowe: -

```
CREATE TABLE Administrators
(
   EmployeeID int NOT NULL,
   CONSTRAINT Administrators_pk PRIMARY KEY  (EmployeeID)
);
```

```
ALTER TABLE Administrators ADD CONSTRAINT Employees_Administrators
FOREIGN KEY (EmployeeID)
REFERENCES Employees (EmployeeID);
```

Tabele z obszaru Order Management

Tabela **Orders**

Klucz główny: OrderID
Klucz obcy:

- EmployeeID (int, not null) - ID pracownika, który obsługuje zamówienie
- ClientID (int, null) - ID klienta składającego zamówienie

Pola:

- OrderDate (date, not null) - data złożenia zamówienia. Domyślnie data wprowadzenia.
- ServingDate (date, not null) - data realizacji zamówienia

```
CREATE TABLE Orders
(
   OrderID int NOT NULL,
   EmployeeID int NOT NULL,
   OrderDate datetime NOT NULL DEFAULT GETDATE(),
   ServingDate datetime NULL,
   ClientID int NULL,
   CONSTRAINT OrderID PRIMARY KEY  (OrderID)
   CONSTRAINT ValidDate CHECK (OrderDate < ServingDate OR ServingDate IS NULL),
);
ALTER TABLE Orders ADD CONSTRAINT Client_Orders
FOREIGN KEY (ClientID)
REFERENCES Client (ClientID);

ALTER TABLE Orders ADD CONSTRAINT Orders_Employees
FOREIGN KEY (EmployeeID)
```

```
REFERENCES Employees (EmployeeID);
```

Tabela **OrderDetails**

Klucz główny: (OrderDetailsID,ProductID)
Klucz obcy: OrderID - ID Zamówienia, do którego należy zakup.
Klucz obcy: ProductID - ID produktu, którego dotyczy zakup.
Kolumny:

- UnitPrice (money, not null) - cena za jaką został zakupiony produkt .

Warunki integralnościowe:

- ValidUnitPrice - zapewnia, że cena jest większa od 0.

```
CREATE TABLE OrderDetails
(
   OrderDetailsID int NOT NULL IDENTITY(1, 1),
   OrderID int NOT NULL,
   ProductID int NOT NULL,
   UnitPrice money NOT NULL,
   CONSTRAINT ValidUnitPrice CHECK (UnitPrice > 0),
   CONSTRAINT OrderDetails_pk PRIMARY KEY  (OrderDetailsID)
);
ALTER TABLE OrderDetails ADD CONSTRAINT Order_OrdersDetails
FOREIGN KEY (OrderID)
REFERENCES Orders (OrderID);

ALTER TABLE OrderDetails ADD CONSTRAINT Products_OrderDetails
FOREIGN KEY (ProductID)
REFERENCES Products (ProductID);
```

Tabela **Takeaway**

Klucz główny: OrderID
Klucz obcy: OrderID (int, not null)
Pola:

- PreferredDate (datetime, not null) - preferowana data odbioru.

Warunki integralnościowe:

- ValidPreferredDate - zapewnia, że PreferredDate nie jest późniejsza od aktualnej.

```
CREATE TABLE Takeaway
(
   OrderID int NOT NULL,
   PreferredDate datetime NOT NULL,
   CONSTRAINT ValidPreferredDate CHECK (PreferredDate > GETDATE()),
   CONSTRAINT Takeaway_pk PRIMARY KEY  (OrderID)
);

ALTER TABLE Takeaway ADD CONSTRAINT Takeaway_Orders
FOREIGN KEY (OrderID)
REFERENCES Orders (OrderID);
```

Tabela **TableDetails**

Klucz główny: TableID, ReservationID
Kolumny:

- NumberOfGuests (money, not null) - liczba gości przydzielonych do stolika na podstawie rezerwacji.

Warunki integralnościowe:

- ValidNumberOfGuests - zapewnia, że liczba gości jest większa lub równa 2

```
CREATE TABLE TableDetails
```

```
(
   TableID int NOT NULL,
   NumberOfGuests int NOT NULL,
   ReservationID int NOT NULL,
   CONSTRAINT ValidNumberOfGuests CHECK (NumberOfGuests >= 2),
   CONSTRAINT TableDetails_pk PRIMARY KEY  (TableID, ReservationID)
);
ALTER TABLE TableDetails ADD CONSTRAINT TableDetails_Reservations
FOREIGN KEY (ReservationID)
REFERENCES Reservations (ReservationID);

ALTER TABLE TableDetails ADD CONSTRAINT Tables_TableDetails
FOREIGN KEY (TableID)
REFERENCES Tables (TableID);
```

Tabela **Tables**

Klucz główny: Table
Pola:

- Name (nvarchar, not null) - unikalna nazwa stolika
- MaxNumberOfGuests (int null) - ilość miejsc przy stole

Warunki integralnościowe:

- ValidNumberOfGuests - ilość miejsc musi być większa od zera
- UniqueName - nazwa stolika musi byc unikalna

```
CREATE TABLE Tables
(
   TableID int NOT NULL,
```

```
    Name nvarchar NOT NULL,
    MaxNumberOfGuests int NOT NULL,
    CONSTRAINT UniqueName UNIQUE (Name),
    CONSTRAINT ValidMaxNumberOfGuests CHECK (MaxNumberOfGuests > 0),
    CONSTRAINT Tables_pk PRIMARY KEY  (TableID)
);
```

Tabele z obszaru Reservations

Tabela **Client**

Klucz główny: ClientID
Kolumny:

- Address - (nvarchar, not null) - Adres klienta
- Phone - (nvarchar, not null) - Telefon klienta
- Email - (nvarchar, not null) - Adres email klienta

Warunki integralnościowe:

- ValidPhone - Zapewnia poprawny format numeru telefonu - Rozpoczyna się znakiem + oraz ma 11 cyfr
- UniquePhone - Zapewnia unikalność numeru telefonu
- ValidEmail - Zapewnia poprawny format adresu email
- UniqueEmail - Zapewnia unikalność adresu email

```
CREATE TABLE Client
(
   ClientID int NOT NULL,
   Address nvarchar NOT NULL,
   Phone nvarchar NOT NULL,
   Email nvarchar NOT NULL,
   CONSTRAINT ValidPhone CHECK (Phone LIKE '+[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
   CONSTRAINT UniquePhone UNIQUE (Phone),
   CONSTRAINT ValidEmail CHECK (Email LIKE '%@%'),
   CONSTRAINT UniqueEmail UNIQUE (Email)
   CONSTRAINT Client_pk PRIMARY KEY (ClientID)
);
```

Tabela **Companies**

Klucz główny: CompanyID
Klucze obce:

- ClientID

Kolumny:

- CompanyName - (nvarchar, not null) - Nazwa firmy
- NIP - (nvarchar, not null) - Numer NIP firmy

Warunki integralnościowe:

- ValidNIP - Zapewnia poprawność formatu numernu NIP - 10 cyfr
- UniqueNip - Zapewnia unikalność numeru NIP

```
CREATE TABLE Companies
(
   CompanyID int NOT NULL,
   ClientID int NOT NULL,
   CompanyName nvarchar NOT NULL,
   NIP nvarchar NOT NULL,
   CONSTRAINT ValidNIP CHECK (NIP LIKE '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
   CONSTRAINT UniqueNIP UNIQUE (NIP),
   CONSTRAINT Companies_pk PRIMARY KEY  (CompanyID)
);

ALTER TABLE Companies ADD CONSTRAINT Client_Companies
FOREIGN KEY (ClientID)
REFERENCES Client (ClientID);
```

Tabela **CompanyReservations**

Klucz główny: ReservationID
Klucze obce:

- ReservationID

Kolumny:

- NumberOfGuests - (int, not null) - Na ile osób jest zrobiona rezerwacja

Warunki integralnościowe:

- ValidNumberOfGuests - Sprawdza czy ilosć osób jest większa lub równa 2

```
CREATE TABLE CompanyReservations
(
   ReservationID int NOT NULL,
   CONSTRAINT CompanyReservations_pk PRIMARY KEY  (ReservationID)
);

ALTER TABLE CompanyReservations ADD CONSTRAINT Reservations_CompanyReservations
FOREIGN KEY (ReservationID)
REFERENCES Reservations (ReservationID);
```

Tabela **CompanyReservationsDetails**

Klucze główne: ReservationID, IndividualID

```
CREATE TABLE CompanyReservationsDetails
(
   IndividualID int NOT NULL,
   ReservationID int NOT NULL,
   CONSTRAINT CompanyReservationsDetails_pk PRIMARY KEY  (ReservationID, IndividualID)
);

ALTER TABLE CompanyReservationsDetails ADD CONSTRAINT CompanyReservationsDetails_Individuals
FOREIGN KEY (IndividualID)
REFERENCES Individuals (IndividualID);

ALTER TABLE CompanyReservationsDetails ADD CONSTRAINT ReservationCompanyDetails_ReservationCompany
FOREIGN KEY (ReservationID)
REFERENCES CompanyReservations (ReservationID);
```

Tabela **IndividualReservations**

Klucz główny: ReservationID
Klucze obce:

- ReservationID - ID rezerwacji.
- OrderID - ID złożonego zamówienia.

Kolumny:

- Prepaid (bit, not null) - czy zamówienie zostało opłacone.

```
CREATE TABLE IndividualReservations
(
   ReservationID int NOT NULL,
   OrderID int NOT NULL,
   Prepaid bit NOT NULL,
   CONSTRAINT IndividualReservations_pk PRIMARY KEY  (ReservationID)
);

ALTER TABLE IndividualReservations ADD CONSTRAINT IndividualReservations_Orders
FOREIGN KEY (OrderID)
REFERENCES Orders (OrderID);

ALTER TABLE IndividualReservations ADD CONSTRAINT Reservations_IndividualReservations
FOREIGN KEY (ReservationID)
REFERENCES Reservations (ReservationID);
```

Tabela **Individuals**

Klucz główny: IndividualID
Klucze obce: ClientID
Kolumny:

- FirstName (nvarchar, not null) - Pierwsze imię klienta.
- LastName (nvarchar, not null) - Nazwisko klienta.

```
CREATE TABLE Individuals
(
   IndividualID int NOT NULL,
   ClientID int NOT NULL,
   FirstName nvarchar NOT NULL,
   LastName nvarchar NOT NULL,
   CONSTRAINT Individuals_pk PRIMARY KEY  (IndividualID)
);

ALTER TABLE Individuals ADD CONSTRAINT Client_Individuals
FOREIGN KEY (ClientID)
REFERENCES Client (ClientID);
```

Tabela **Reservations**

Klucz główny: ReservationID
Klucze obce:

- ClientID

Kolumny:

- **ReservationDate** - (datetime, not null) - Data i czas, momentu stworzenia rezerwacji. Domyślnie obecna data i czas.
- **StartDate** - (datetime, not null) - Data i czas, momentu kiedy rozpoczyna się rezerwacja
- **EndDate** - (datetime, not null) - Data i czas kiedy kończy się rezerwacja
- **Accepted -** (bit, not null) - Akceptacja rezerwacji przez pracowników. Domyślnie 0.

Warunki integralnościowe:

- ValidDate - Zapewnia, że rezerwacja została dokonana przed rozpoczęciem oraz że rezerwacja rozpoczyna się przed jej końcem.
- ValidNumberOfGuests - Zapewnia warunek “przynajmniej dwie osoby potrzebne są do rezerwacji”

```
CREATE TABLE Reservations
(
   ReservationID int NOT NULL,
   ClientID int NOT NULL,
   ReservationDate datetime NOT NULL DEFAULT GETDATE(),
   StartDate datetime NOT NULL,
   EndDate datetime NOT NULL,
   Accepted bit NOT NULL DEFAULT 0,
   NumberOfGuests int NOT NULL,
   CONSTRAINT ValidNumberOfGuests CHECK (NumberOfGuests >= 2),
   CONSTRAINT ValidDate CHECK (ReservationDate < StartDate AND StartDate < EndDate),
   CONSTRAINT Reservations_pk PRIMARY KEY  (ReservationID)
);

ALTER TABLE Reservations ADD CONSTRAINT Reservations_Client
FOREIGN KEY (ClientID)
```

```
REFERENCES Client (ClientID);
```

Tabela **Discounts**

Klucz główny: IndividualID
Klucze obce: IndividualID
Kolumny:

- StartDate (datetime, not null) - data rozpoczęcia rabatu.
- EndDate (datetime, not null) - data utraty ważności rabatu.
- Value (int, not null) - wartość rabatu w %.

Warunki integralnościowe:

- ValidDate - zapewnia, że EndDate jest późniejsze od StartDate.
- ValidValue - zapewnie, że wartość jest prawidłową liczbą

```
CREATE TABLE Discounts
(
   DiscountID INT NOT NULL IDENTITY(1, 1),
   IndividualID int NOT NULL,
   StartDate datetime NOT NULL DEFAULT GETDATE(),
   EndDate datetime NOT NULL,
   Value int NOT NULL,
   CONSTRAINT ValidDate CHECK (EndDate > StartDate),
   CONSTRAINT ValidValue CHECK (Value > 0 AND Value <= 100),
   CONSTRAINT Discounts_pk PRIMARY KEY  (DiscountID)
);
ALTER TABLE Discounts ADD CONSTRAINT Discounts_Individuals
FOREIGN KEY (IndividualID)
REFERENCES Individuals (IndividualID);
```

Tabele z obszaru Parameters

Tabela **DiscountParameters**

Klucz główny: ParameterID
Klucze obce: -
Kolumny:

- ParameterName (varchar(2), not null) - kod parametru rabatu.
- StartDate (datetime, not null) - data rozpoczęcia ważności parametru rabatu. Domyślnie bieżąca data.
- EndDate (datetime, not null) - data utraty ważności parametru rabatu. Domyślnie null - bezterminowa ważność.
- Value (int, not null) - wartość rabatu.

Warunki integralnościowe:

- ValidDate - zapewnia, że EndDate jest późniejsze od StartDate.
- ValidValue - zapewnie, że wartość jest prawidłową wartością rabatu

```
CREATE TABLE DiscountParameters
(
   ParameterID int NOT NULL,
   ParameterName varchar(2) NOT NULL,
   StartDate datetime NOT NULL DEFAULT GETDATE(),
   EndDate datetime NULL DEFAULT NULL,
   Value int NOT NULL,
   CONSTRAINT ValidDate CHECK (EndDate > StartDate),
   CONSTRAINT ValidValue CHECK (Value > 0),
   CONSTRAINT DiscountParameters_pk PRIMARY KEY  (ParameterID)
);
```

Tabela **Parameters**

Klucz główny: ParameterID
Klucze obce: -
Kolumny:

- Value (int, not null) - wartość parametru.

Warunki integralnościowe:

- ValidValue - zapewnie, że wartość jest prawidłową parametru >= 0.

```
CREATE TABLE Parameters
(
   ParameterID varchar(2) NOT NULL,
   Value int NOT NULL ,
   CONSTRAINT ValidValue CHECK (Value >= 0),
   CONSTRAINT Parameters_pk PRIMARY KEY  (ParameterID)
);
```

Widoki

Widok ClientStats

Przedstawia ilość oraz wartość zamówień każdego nie anonimowego klienta.

```
SELECT dbo.Client.ClientID, COUNT(dbo.Orders.OrderID) AS NumberOfOrders, SUM(dbo.OrderDetails.UnitPrice) AS
ValueOfOrders
FROM  dbo.Orders INNER JOIN
                  dbo.Client ON dbo.Orders.ClientID = dbo.Client.ClientID INNER JOIN
                  dbo.OrderDetails ON dbo.Orders.OrderID = dbo.OrderDetails.OrderID
GROUP BY dbo.Client.ClientID
```

Widok CompanyOrders

Przedstawia historię zamówień dla każdej firmy wraz z ich wartością, datą zamówienia i datą podania

```
SELECT Companies_1.CompanyName, dbo.Orders.OrderID, dbo.Orders.OrderDate, dbo.Orders.ServingDate,
SUM(dbo.OrderDetails.UnitPrice) AS OrderValue
FROM     dbo.Companies AS Companies_1 INNER JOIN
                  dbo.Client ON Companies_1.ClientID = dbo.Client.ClientID INNER JOIN
                  dbo.Orders INNER JOIN
                  dbo.OrderDetails ON dbo.Orders.OrderID = dbo.OrderDetails.OrderID ON dbo.Client.ClientID =
dbo.Orders.ClientID
GROUP BY Companies_1.CompanyName, dbo.Orders.OrderID, dbo.Orders.OrderDate, dbo.Orders.ServingDate
```

Widok IndividualOrders

Przedstawia historię zamówień każdego indywidualnego klienta w systemie, oraz wartości każdego z osobna

```
SELECT dbo.Individuals.FirstName, dbo.Individuals.LastName, dbo.Orders.OrderID, dbo.Orders.OrderDate,
dbo.Orders.ServingDate, SUM(dbo.OrderDetails.UnitPrice) AS OrderValue
FROM     dbo.Orders INNER JOIN
                  dbo.OrderDetails ON dbo.Orders.OrderID = dbo.OrderDetails.OrderID INNER JOIN
                  dbo.Client ON dbo.Orders.ClientID = dbo.Client.ClientID INNER JOIN
                  dbo.Individuals ON dbo.Client.ClientID = dbo.Individuals.ClientID
GROUP BY dbo.Individuals.FirstName, dbo.Individuals.LastName, dbo.Orders.OrderID, dbo.Orders.OrderDate,
dbo.Orders.ServingDate
```

Widok IndividualReservationsHistory

Przedstawia historię rezerwacji klientów indywidualnych, ich daty rezerwacji oraz początku i konca

```
SELECT dbo.Individuals.FirstName, dbo.Individuals.LastName, dbo.Reservations.ReservationDate,
dbo.Reservations.StartDate, dbo.Reservations.EndDate
FROM     dbo.Client INNER JOIN
                  dbo.Individuals ON dbo.Client.ClientID = dbo.Individuals.ClientID INNER JOIN
                  dbo.Reservations ON dbo.Client.ClientID = dbo.Reservations.ClientID
```

Widok CurrentDiscounts

Przedstawia aktualne zniżki dla klientów indywidualnych wraz ich danymi oraz datą ważności

```
SELECT dbo.Individuals.ClientID, dbo.Individuals.FirstName, dbo.Individuals.LastName, dbo.Discounts.EndDate,
dbo.Discounts.Value
FROM     dbo.Discounts INNER JOIN
                  dbo.Individuals ON dbo.Discounts.IndividualID = dbo.Individuals.IndividualID
WHERE  (GETDATE() BETWEEN dbo.Discounts.StartDate AND dbo.Discounts.EndDate)
```

Widok CurrentlyFreeTables

Przedstawia listę stolików, które nie są aktualnie zarezerwowane

```
SELECT dbo.Tables.Name, dbo.Tables.MaxNumberOfGuests
FROM     dbo.Reservations INNER JOIN
                  dbo.TableDetails ON dbo.Reservations.ReservationID = dbo.TableDetails.ReservationID INNER JOIN
                  dbo.Tables ON dbo.TableDetails.TableID = dbo.Tables.TableID
WHERE  (NOT (GETDATE() BETWEEN dbo.Reservations.StartDate AND dbo.Reservations.EndDate))
```

Widok CurrentMenu

Przedstawia aktualne menu

```
SELECT dbo.Products.Name, dbo.Products.UnitPrice
FROM     dbo.Menu INNER JOIN
                  dbo.Products ON dbo.Menu.ProductID = dbo.Products.ProductID
WHERE  (GETDATE() BETWEEN dbo.Menu.StartDate AND dbo.Menu.EndDate) OR (dbo.Menu.EndDate IS NULL AND GETDATE() >
dbo.Menu.StartDate)
```

Widok EmployeeStats

Przedstawia dla każdego pracownika ilość oraz wartość obsłużonych zamówień

```
SELECT dbo.Employees.EmployeeID, dbo.Employees.FirstName, dbo.Employees.Surename, COUNT(dbo.Orders.OrderID) AS
OrdersNumber, SUM(dbo.OrderDetails.UnitPrice) AS OrdersValue
FROM     dbo.Employees INNER JOIN
                  dbo.Orders ON dbo.Employees.EmployeeID = dbo.Orders.EmployeeID INNER JOIN
                  dbo.OrderDetails ON dbo.Orders.OrderID = dbo.OrderDetails.OrderID
GROUP BY dbo.Employees.EmployeeID, dbo.Employees.FirstName, dbo.Employees.Surename
```

Widok GeneralProductStats

Przedstawia ID produktu, jego nazwę, jego menuID i daty dostępności w nim, oraz ilość jego sprzedażyw tym okresie

```
SELECT dbo.Products.ProductID, dbo.Products.Name AS ProductName, dbo.Menu.StartDate, dbo.Menu.EndDate,
dbo.Menu.MenuID, COUNT(dbo.OrderDetails.OrderDetailsID) AS NumberOfOrders
FROM     dbo.Products INNER JOIN
                  dbo.OrderDetails ON dbo.Products.ProductID = dbo.OrderDetails.ProductID INNER JOIN
                  dbo.Menu ON dbo.Products.ProductID = dbo.Menu.ProductID
GROUP BY dbo.Products.ProductID, dbo.Products.Name, dbo.Menu.StartDate, dbo.Menu.EndDate, dbo.Menu.MenuID
```

Widok MonthlyOrderStats

Przedstawia statystyki zamówień z podziałem na lata i miesiące

```
SELECT YEAR(dbo.Orders.ServingDate) AS Year, MONTH(dbo.Orders.ServingDate) AS Month, COUNT(dbo.Orders.OrderID) AS
OrdersNumber SUM(dbo.OrderDetails.UnitPrice) AS OrdersValue
FROM     dbo.Orders INNER JOIN
                  dbo.OrderDetails ON dbo.Orders.OrderID = dbo.OrderDetails.OrderID
GROUP BY YEAR(dbo.Orders.ServingDate), MONTH(dbo.Orders.ServingDate) WITH ROLLUP
```

Widok WeeklyOrderStats

Przedstawia statystyki zamówień z podziałem na lata i tygodnie

```
SELECT YEAR(dbo.Orders.ServingDate) AS Year, { fn WEEK(dbo.Orders.ServingDate) } AS Week,
COUNT(dbo.Orders.OrderID) AS OrdersNumber, SUM(dbo.OrderDetails.UnitPrice) AS OrdersValue
FROM     dbo.Orders INNER JOIN
                  dbo.OrderDetails ON dbo.Orders.OrderID = dbo.OrderDetails.OrderID
GROUP BY YEAR(dbo.Orders.ServingDate), { fn WEEK(dbo.Orders.ServingDate) } WITH ROLLUP
```

Widok MonthlyProductStats

Przedstawia ilość sprzedanych produktów z podziałem na lata i miesiące

```
SELECT dbo.Products.Name, YEAR(dbo.Orders.OrderDate) AS year, MONTH(dbo.Orders.OrderDate) AS month,
COUNT(dbo.OrderDetails.OrderID) AS SoldProducts
FROM     dbo.OrderDetails INNER JOIN
                  dbo.Products ON dbo.OrderDetails.ProductID = dbo.Products.ProductID INNER JOIN
                  dbo.Orders ON dbo.OrderDetails.OrderID = dbo.Orders.OrderID
GROUP BY dbo.Products.Name, YEAR(dbo.Orders.OrderDate), MONTH(dbo.Orders.OrderDate) WITH ROLLUP
```

Widok ProductDetails

Przedstawia informacje o kategorii produktu oraz jego opis

```
SELECT dbo.Products.Name, dbo.Categories.CategoryName, dbo.Products.Description
FROM     dbo.Categories INNER JOIN
                  dbo.Products ON dbo.Categories.CategoryID = dbo.Products.CategoryID
```

Widok ReservationsToCheck

Przedstawia listę nadchodzących rezerwacji nie zaakceptowanych przez pracownika oraz ID klienta, który je złożył

```
SELECT ReservationID, ClientID
FROM     dbo.Reservations
WHERE  (Accepted = 0) AND (GETDATE() < StartDate)
```

Widok TableReservationDetails

Przedstawia nazwę, ID stolika oraz ile razy był on rezerwowany

```
SELECT dbo.TableDetails.TableID, dbo.Tables.Name, COUNT(dbo.TableDetails.ReservationID) AS NumberOfReservations
FROM     dbo.TableDetails INNER JOIN
                  dbo.Tables ON dbo.TableDetails.TableID = dbo.Tables.TableID
GROUP BY dbo.TableDetails.TableID, dbo.Tables.Name
```

Widok UpcommingTakeawayOrders

Przedstawia listę zamówień na wynos do przygotowania w przyszłości, oraz dane kontaktowe z klientem, który je odbiera

```
SELECT dbo.Client.ClientID, dbo.Client.Phone, dbo.Takeaway.PreferredDate
FROM     dbo.Takeaway CROSS JOIN
                  dbo.Client
WHERE  (dbo.Takeaway.PreferredDate > GETDATE())
```

Widok UpcommingReservations

Przedstawia listę zbliżających się rezerwacji z informacjami o zarezerwowanym stoliku, datą początku i końca oraz danymi kontaktowymi rezerwującego klienta

```
SELECT dbo.TableDetails.ReservationID, dbo.Tables.Name AS TableName, dbo.Client.ClientID, dbo.Client.Phone,
dbo.Client.Email, dbo.Reservations.StartDate, dbo.Reservations.EndDate, dbo.Reservations.Accepted
FROM     dbo.Reservations INNER JOIN
                  dbo.TableDetails ON dbo.Reservations.ReservationID = dbo.TableDetails.ReservationID INNER JOIN
                  dbo.Tables ON dbo.TableDetails.TableID = dbo.Tables.TableID INNER JOIN
                  dbo.Client ON dbo.Reservations.ClientID = dbo.Client.ClientID
WHERE  (GETDATE() < dbo.Reservations.StartDate)
```

Widok CompanyReservationsHistory

Przedstawia historię rezerwacji firm

```
SELECT dbo.Companies.CompanyName, dbo.Reservations.StartDate, dbo.Reservations.EndDate
FROM     dbo.Client INNER JOIN
                  dbo.Companies ON dbo.Client.ClientID = dbo.Companies.ClientID INNER JOIN
                  dbo.Reservations ON dbo.Client.ClientID = dbo.Reservations.ClientID
```

Widok MonthlyCompanyOrders

Przedstawia wartość zamówień firm z podziałem na miesiące i lata w celu wystawienia faktury za dany miesiąc.

```
SELECT CompanyName, MONTH(ServingDate) AS Month, YEAR(ServingDate) AS Year, SUM(OrderValue) AS OrderValue
FROM     (SELECT Companies_1.CompanyName, dbo.Orders.OrderID, dbo.Orders.OrderDate, dbo.Orders.ServingDate,
SUM(dbo.OrderDetails.UnitPrice) AS OrderValue
                  FROM      dbo.Companies AS Companies_1 INNER JOIN
                                    dbo.Client ON Companies_1.ClientID = dbo.Client.ClientID INNER JOIN
                                    dbo.Orders INNER JOIN
                                    dbo.OrderDetails ON dbo.Orders.OrderID = dbo.OrderDetails.OrderID ON
dbo.Client.ClientID = dbo.Orders.ClientID
                  GROUP BY Companies_1.CompanyName, dbo.Orders.OrderID, dbo.Orders.OrderDate,
dbo.Orders.ServingDate) AS derivedtbl_1
GROUP BY CompanyName, MONTH(ServingDate), YEAR(ServingDate)
```

Widok SeaFoodOrders

Przedstawia listę wszystkich zamówień na owoce morza

```
SELECT dbo.Orders.ClientID, dbo.Orders.OrderDate, dbo.Orders.ServingDate
FROM     dbo.Categories INNER JOIN
                  dbo.Products ON dbo.Categories.CategoryID = dbo.Products.CategoryID INNER JOIN
                  dbo.OrderDetails ON dbo.Products.ProductID = dbo.OrderDetails.ProductID INNER JOIN
                  dbo.Orders ON dbo.OrderDetails.OrderID = dbo.Orders.OrderID
WHERE  (dbo.Categories.CategoryName = 'Owoce morza')
```

Widok AverageServiceTime

Przedstawia średni czas obsługi zamówienia przez każdego pracownika w minutach.

```
SELECT dbo.Employees.EmployeeID, dbo.Employees.FirstName, dbo.Employees.Surename,
      AVG(DATEDIFF(MINUTE, dbo.Orders.OrderDate, dbo.Orders.ServingDate)) AS AverageServiceTime,
      COUNT(dbo.Orders.OrderID) AS NumberOfServicedOrders
FROM     dbo.Employees INNER JOIN
                  dbo.Orders ON dbo.Employees.EmployeeID = dbo.Orders.EmployeeID
WHERE  (dbo.Orders.OrderID NOT IN
                     (SELECT OrderID
                      FROM      dbo.Takeaway AS Takeaway_1))
GROUP BY dbo.Employees.EmployeeID, dbo.Employees.FirstName, dbo.Employees.Surename
```

Widok AverageProductServingTime

Przedstawia średni czas przygotowania każdej z dostępnych potraw w minutach, kategorię potrawy oraz całkowitą ilość zamówień

```
SELECT dbo.Products.Name, dbo.Products.CategoryID, AVG(DATEDIFF(MINUTE, dbo.Orders.OrderDate,
dbo.Orders.ServingDate)) AS AverageServiceTime, COUNT(dbo.Orders.OrderID) AS NumberOfOrders
FROM     dbo.Products INNER JOIN
                  dbo.OrderDetails ON dbo.Products.ProductID = dbo.OrderDetails.ProductID INNER JOIN
                  dbo.Orders ON dbo.OrderDetails.OrderID = dbo.Orders.OrderID
WHERE  (dbo.Orders.OrderID NOT IN
                      (SELECT OrderID
                       FROM      dbo.Takeaway AS Takeaway_1))
GROUP BY dbo.Products.Name, dbo.Products.CategoryID
```

Widok OrderHistory

Przedstawia historię zamówień, czyli wartość każdego z nich, zamawiającego klienta, datę oraz ilość produktów

```
SELECT dbo.Orders.OrderID, dbo.Client.ClientID, dbo.Orders.OrderDate, FORMAT(SUM(dbo.OrderDetails.UnitPrice),
'N2') AS OrderValue, COUNT(*) AS NumberOfProducts
FROM     dbo.Client INNER JOIN
                  dbo.Orders ON dbo.Client.ClientID = dbo.Orders.ClientID INNER JOIN
                  dbo.OrderDetails ON dbo.Orders.OrderID = dbo.OrderDetails.OrderID
GROUP BY dbo.Orders.OrderID, dbo.Client.ClientID, dbo.Orders.OrderDate
```

Zdefiniowane typy

Typy Table-Valued Parameters - Używane do przekazywania list do procedur i funkcji

```
CREATE TYPE ProductList AS TABLE (ProductID INT NOT NULL)
```

```
CREATE TYPE IndividualList AS TABLE (IndividualID INT NOT NULL)
```

```
CREATE TYPE TablesList AS TABLE (TableID INT NOT NULL, NumberOfGuests INT NOT NULL)
```

Funkcje

CanOrderSeafood

Sprawdza czy można w określonych datach zamówić owoce morza. Przyjmuje:

- OrderDate - Data zamówienia
- ServingDate - Data którego dnia zamówienie ma być zrealizowane

```
CREATE FUNCTION CanOrderSeafood(@OrderDate DATETIME, @ServingDate DATETIME)
RETURNS BIT
AS
BEGIN
   IF @OrderDate <= dbo.GetPreviousMonday(@ServingDate)
       RETURN 1

   RETURN 0
END
GO
```

GetPreviousMonday

Zwraca ostatni poniedziałek względem podanej daty. Używane w CanOrderSeafood

```
CREATE FUNCTION GetPreviousMonday(@Date DATETIME)
RETURNS DATETIME
AS
BEGIN
   RETURN DATEADD(wk, DATEDIFF(wk, 6, @ServingDate), 7)
END
GO


CREATE FUNCTION HasSeafood(@ProductIDs ProductList READONLY)
RETURNS BIT
AS
BEGIN
   IF EXISTS (
   SELECT *
   FROM @ProductIDs as L
       INNER JOIN Products AS P ON L.ProductID = P.ProductID
       INNER JOIN Categories C on P.CategoryID = C.CategoryID
   WHERE C.CategoryName LIKE N'Owoce morza')
       RETURN 1

   RETURN 0
END
GO
```

HasSeafood

Sprawdza czy lista produktów ma w sobie produkt z kategorii owoce morza.

```
CREATE FUNCTION HasSeafood(@ProductIDs ProductList READONLY)
RETURNS BIT
AS
BEGIN
   IF EXISTS (
   SELECT *
   FROM @ProductIDs as L
       INNER JOIN Products AS P ON L.ProductID = P.ProductID
       INNER JOIN Categories C on P.CategoryID = C.CategoryID
   WHERE C.CategoryName LIKE N'Owoce morza')
       RETURN 1

   RETURN 0
END
GO
```

CanMakeReservation

Sprawdza czy określony Individual może zrobić rezerwację - Sprawdza warunki WK i WZ

```
CREATE FUNCTION CanMakeReservation(@IndividualID INT, @ProductIDs ProductList READONLY)
RETURNS BIT
AS
BEGIN
   DECLARE @WZ INT
   SELECT @WZ = Value
   FROM Parameters
   WHERE ParameterID LIKE 'WZ'

   DECLARE @WK INT
   SELECT @WK = Value
   FROM Parameters
   WHERE ParameterID LIKE 'WK'

   DECLARE @ClientID INT
   SELECT @ClientID = ClientID
   FROM Individuals
   WHERE IndividualID = @IndividualID

   DECLARE @OrdersTotal INT
   SET @OrdersTotal = (SELECT COUNT(*)
   FROM Orders
   WHERE ClientID = @ClientID)

   DECLARE @PriceTotal money
```

```
    SET @PriceTotal = (SELECT SUM(P.UnitPrice)
    FROM @ProductIDs AS IDs INNER JOIN Products AS P ON P.ProductID
    = IDs.ProductID)

    IF @PriceTotal >= @WZ AND @OrdersTotal >= @WK BEGIN
        RETURN 1
    END
    RETURN 0
END
GO
```

GetCurrentDiscount

Zwraca zniżkę podanego Individual dnia podanego w @Date

```
CREATE FUNCTION GetCurrentDiscount(@IndividualID INT, @Date DATETIME)
RETURNS INT
AS
BEGIN
   DECLARE @result INT
   SET @result = (SELECT D.Value
   FROM Individuals AS I
       INNER JOIN Discounts D on I.IndividualID = D.IndividualID
   WHERE I.IndividualID = @IndividualID AND @Date BETWEEN D.StartDate AND D.EndDate)
   IF @result IS NULL BEGIN
       RETURN 0
   END
   RETURN @result
END
```

VerifyMenu

Weryfikuje czy aktualne menu spełnia warunki o co dwutygodniowej zmianie co najmniej połowy pozycji. Kategoria “Owoce morza” jest traktowana wyjątkowo i pomijamy produkty do niej należące

```
CREATE FUNCTION VerifyMenu()
RETURNS BIT
AS
BEGIN
    DECLARE @Count INT
    SET @Count = (
        SELECT COUNT(*)
        FROM menu as M
        INNER JOIN Products P on P.ProductID = M.ProductID
        INNER JOIN Categories C on C.CategoryID = P.CategoryID AND C.CategoryName <> 'Owoce morza'
        WHERE EndDate IS NULL OR EndDate > GETDATE())

    DECLARE @CountBad INT
    SET @CountBad = (
        SELECT COUNT(*)
        FROM menu as M
        INNER JOIN Products P on P.ProductID = M.ProductID
        INNER JOIN Categories C on C.CategoryID = P.CategoryID AND C.CategoryName <> 'Owoce morza'
        WHERE (EndDate IS NULL OR EndDate > GETDATE()) AND DATEDIFF(DAY, StartDate, GETDATE()) >= 14
    )

    IF 2 * @CountBad >= @Count
```

```
        RETURN 0

    RETURN 1
END
go
```

GetOrderValue

Zwraca wartość zamówienia o zadanym OrderID, ID klienta oraz ilość zamówionych produktów

```
CREATE FUNTION GetOrderValue(@OrderID)
RETURNS TABLE
AS
RETURN
(
      SELECT *
      FROM dbo.OrderHistory
      WHERE OrderID = @OrderID
)
GO
```

GetOrderDetails

Funkcja przyjmująca id zamówienia i pokazująca zamówione produkty

```
CREATE FUNCTION GetOrderDetails(@OrderID INT)
RETURNS TABLE
AS
RETURN(
      SELECT P.Name, C.CategoryName, P.UnitPrice AS PriceWithNoDiscount, OD.UnitPrice AS PriceWithDiscount
      FROM OrderDetails AS OD
      INNER JOIN Products AS P ON P.ProductID = OD.ProductID
      INNER JOIN Categories AS C ON C.CategoryID = P.CategoryID
      WHERE OD.OrderID = @OrderID
      )
GO
```

Procedury

AddCategory

Dodaje kategorię do tablicy Categories.
Przyjmuje:

- CategoryName nvarchar(255) - Nazwa nowej kategorii

```
CREATE PROCEDURE AddCategory
   @CategoryName nvarchar(255)
AS
```

```
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
   IF EXISTS(SELECT *
   FROM Categories
   WHERE @CategoryName = CategoryName)
   BEGIN;
       THROW 52000, N'Category already in database', 1
   END

   DECLARE @CategoryID INT
   SELECT @CategoryID = ISNULL(MAX(CategoryID), 0) + 1
   FROM Categories

   INSERT INTO Categories
       (CategoryID, CategoryName)
   VALUES
       (@CategoryID, @CategoryName);
END TRY
BEGIN CATCH
   DECLARE @msg nvarchar(1024) = N'An error occurred while adding a category: ' + ERROR_MESSAGE();
   THROW 52000, @msg, 1;
END CATCH
END
GO
```

Przykład użycia:

```
EXEC AddCategory N'Zupy'
```

AddProduct

Dodaje nowy produkt
Przyjmuje:

- Name - Nazwa nowego produktu
- CategoryName - Nazwa kategorii produktu
- UnitPrice - Cena
- Description - Opis produktu

```
CREATE PROCEDURE AddProduct
   @Name nvarchar(255),
   @CategoryName nvarchar(255),
   @UnitPrice money,
   @Description nvarchar(1024)
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY

   IF EXISTS(
   SELECT *
   FROM Products
   WHERE Name = @Name
   )
   BEGIN;
       THROW 52000, N'Product already in database', 1
   END
```

```
    IF NOT EXISTS(
    SELECT *
    FROM Categories
    WHERE CategoryName = @CategoryName
    )
    BEGIN;
        THROW 52000, N'Could not find category in database', 1
    END

    DECLARE @CategoryID INT
    SELECT @CategoryID = CategoryID
    FROM Categories
    WHERE CategoryName = @CategoryName

    DECLARE @ProductID INT
    SELECT @ProductID = ISNULL(MAX(ProductID), 0) + 1
    FROM Products

    INSERT INTO Products
        (ProductID, CategoryID, UnitPrice, Description, Name)
    VALUES
        (@ProductID, @CategoryID, @UnitPrice, @Description, @Name);

END TRY
BEGIN CATCH
    DECLARE @msg nvarchar(1024) = N'An error occurred while adding a product: ' + ERROR_MESSAGE();
    THROW 52000, @msg, 1;
```

```
END CATCH
END
GO
```

Przykłąd użycia:

```
EXEC AddProduct N'Naleśniki', N'Dania Główne', 30, N'Lekkie lecz pożywne'
```

AddToMenu

Dodaje produkt do menu. Przyjmuje

- ProductName - Nazwa produktu
- StartDate - Początek pobytu danego produktu w menu
- EndDate (DEFAUTL NULL) - Koniec pobytu danego produktu w menu

```
CREATE PROCEDURE AddToMenu
   @ProductName nvarchar(255),
   @StartDate datetime = NULL,
   @EndDate datetime = NULL
AS
BEGIN
   BEGIN TRY
       IF NOT EXISTS (SELECT *
   FROM Products
   WHERE Name = @ProductName) BEGIN;
       THROW 52000, N'Could not a product with such a name', 1
   END

   DECLARE @ProductID INT
   SELECT @ProductID = ProductID
   FROM Products
```

```
    WHERE Name = @ProductName

    DECLARE @MenuID INT
    SELECT @MenuID = ISNULL(MAX(MenuID), 0) + 1
    FROM Menu

    IF @StartDate IS NULL BEGIN;
        -- Clever yet unreadable. Sets @StartDate as midnight of next day.
        SET @StartDate = (Select DATEADD(d, 0, DATEDIFF(d, 0, (SELECT DATEADD(day, 1, GETDATE()))))))
    END

    INSERT INTO Menu
        (MenuID, ProductID, StartDate, EndDate)
    VALUES
        (@MenuID, @ProductID, @StartDate, @EndDate)
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(1024) = N'An error occurred while adding a product to the menu: ' +
ERROR_MESSAGE();
        THROW 52000, @msg, 1;
    END CATCH
END
GO

GO
```

Przykład użycia:

```
DECLARE @yesterday DATETIME
```

```
SELECT @yesterday = GETDATE() - 1
EXEC AddToMenu N'Szaszłyk z kurczaka', @yesterday
```

Wykonanie:

```
EXEC AddToMenu N'Woda'
```

Spowoduje dodanie produktu do menu ale nie wyświetli się on w widoku CurrentMenu ponieważ StartDate jest ustawiony na północ kolejnego dnia.

AddIndividual

Dodaje klienta indywidualnego do bazy. Przymuje

- FirstName - Imie
- LastName - Nazwisko
- Address - Adres
- Phone - Numer telefoniczny
- Email - Adres email

```
CREATE PROCEDURE AddIndividual
   @FirstName nvarchar(255),
   @LastName nvarchar(255),
   @Address nvarchar(255),
   @Phone nvarchar(15),
   @Email nvarchar(255)
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY

   IF EXISTS (SELECT *
```

```
FROM Client
WHERE Address = @Address)
BEGIN;
    THROW 52000, N'Given address is already in the database', 1
END

IF EXISTS (SELECT *
FROM Client
WHERE Phone = @Phone)
BEGIN;
    THROW 52000, N'Given phone is already in database', 1
END

IF EXISTS (SELECT *
FROM Client
WHERE Email = @Email)
BEGIN;
    THROW 52000, N'Given email is already in database', 1
END

DECLARE @ClientID INT
SELECT @ClientID = ISNULL(MAX(ClientID), 0) + 1
FROM Client

DECLARE @IndividualID INT
SELECT @IndividualID = ISNULL(MAX(IndividualID), 0) + 1
FROM Individuals
```

```
    INSERT INTO Client
        (ClientID, Address, Phone, Email)
    VALUES
        (@ClientID, @Address, @Phone, @Email)

    INSERT INTO Individuals
        (IndividualID, ClientID, FirstName, LastName)
    VALUES
        (@IndividualID, @ClientID, @FirstName, @LastName)

    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(1024)
        =N'An error occurred while adding an individual client: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1;
    END CATCH
END
GO
```

Przykłąd użycia:

```
EXEC AddIndividual N'John', N'Carmack', N'ul. Doom 42', '+22573923409', 'thecooldude@gmail.com'
```

AddComapny

Dodaje firmę. Przyjmuje

- CompanyName - Nazwa firmy
- NIP - Numer NIP
- Address - Adres firmy

- Phone - Telefon do firmy
- Email - Email firmy

```
CREATE PROCEDURE AddCompany
   @CompanyName nvarchar(255),
   @NIP nvarchar(255),
   @Address nvarchar(255),
   @Phone nvarchar(15),
   @Email nvarchar(255)
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY

   IF EXISTS (SELECT *
   FROM Companies
   WHERE CompanyName = @CompanyName)
   BEGIN;
       THROW 52000, N'Given company name is already registered in the database', 1
   END

   IF EXISTS (SELECT *
   FROM Companies
   WHERE NIP = @NIP)
   BEGIN;
       THROW 52000, N'Given NIP is already registered in the database', 1
   END

   IF EXISTS (SELECT *
```

```
FROM Client
WHERE Address = @Address)
BEGIN;
    THROW 52000, N'Given address is already in the database', 1
END

IF EXISTS (SELECT *
FROM Client
WHERE Phone = @Phone)
BEGIN;
    THROW 52000, N'Given phone is already in database', 1
END

IF EXISTS (SELECT *
FROM Client
WHERE Email = @Email)
BEGIN;
    THROW 52000, N'Given email is already in database', 1
END

DECLARE @ClientID INT
SELECT @ClientID = ISNULL(MAX(ClientID), 0) + 1
FROM Client

DECLARE @CompanyID INT
SELECT @CompanyID = ISNULL(MAX(CompanyID), 0) + 1
FROM Companies
```

```
    INSERT INTO Client
        (ClientID, Address, Phone, Email)
    VALUES
        (@ClientID, @Address, @Phone, @Email)

    INSERT INTO Companies
        (CompanyID, ClientID, CompanyName, NIP)
    VALUES
        (@CompanyID, @ClientID, @CompanyName, @NIP)

    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(1024)
        =N'An error occurred while adding a company: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1;
    END CATCH
END
GO
```

Przykład użycia:

```
EXEC AddCompany 'ID Software', '6462933516', 'Richardson, Teksas','+22696902523', 'contact@idsoftware.com'
```

AddIndividualReservation

Dodaje rezjrwację do tabel IndividualReservations, Reservations. Dodaje zamówienie do tablicy Orders oraz dodaje wszystkie produkty wyliczone w @ProductIDs do OrderDetails. Przyjmuje:

- IndividualID - Id osoby na którą jest robiona rezerwacja
- StartDate - Czas rozpoczęcia rezerwacji
- EndDate - Czas zakończenia rezerwacji
- Prepaid - Bit zawierający informację o tym, czy order podpięty do rezerwacji jest już opłacony
- EmployeeID - ID pracownika, który obsługuje zamówienie
- OrderDate - Data kiedy zamówienie zostało złożone
- NumberOfGuests - Ile osób obejmuje rezerwacja
- ProductIDs - Table-Valued Parameters - Lista ID produktów zamawianych
- ServingDate - Data kiedy dokonane będzie zamówienie

```
CREATE PROCEDURE AddIndividualReservation
   @IndividualID int,
   @StartDate datetime,
   @EndDate datetime,
   @Prepaid bit,
   @EmployeeID int,
   @OrderDate datetime,
   @NumberOfGuests int,
   @ProductIDs ProductList READONLY,
   @ServingDate datetime = NULL
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY

   IF NOT EXISTS(SELECT *
```

```
FROM Individuals
WHERE IndividualID = @IndividualID)
BEGIN;
    THROW 52000,'Individual is not registered in database', 1
END

IF dbo.CanMakeReservation(@IndividualID, @ProductIDs) = 0 BEGIN;
    THROW 52000,'This individual has not yet made enough orders to make a reservation', 1
END

IF dbo.HasSeafood(@ProductIDs) = 1 BEGIN
    IF @ServingDate IS NULL BEGIN;
        THROW 52000, 'Cannot order seafood without specifying when it will be served', 1
    END
    IF NOT ((SELECT DATEPART(WEEKDAY, @ServingDate) - 1) BETWEEN 4 AND 6) BEGIN;
        THROW 52000, 'Trying to order seafood for days other than Thursday, Friday, Saturday', 1
    END
    IF dbo.CanOrderSeafood(@OrderDate, @ServingDate) = 0 BEGIN;
        THROW 52000, 'Must order seafood before tuesday preceding serving date', 1
    END
END


    DECLARE @ClientID INT
    SELECT @ClientID = ClientID
FROM Individuals
WHERE IndividualID = @IndividualID
```

```
    DECLARE @OrderID INT
    SELECT @OrderID = ISNULL(MAX(OrderID), 0) + 1
FROM Orders

    DECLARE @ReservationID INT
    SELECT @ReservationID = ISNULL(MAX(ReservationID), 0) + 1
FROM Reservations

    INSERT INTO Reservations
    (ReservationID, ClientID, StartDate, EndDate, Accepted, NumberOfGuests)
VALUES
    (@ReservationID, @ClientID, @StartDate, @EndDate, 0, @NumberOfGuests)

INSERT INTO Orders
    (OrderID, EmployeeID, OrderDate, ServingDate, ClientID)
VALUES
    (@OrderID, @EmployeeID, @OrderDate, @ServingDate, @ClientID)

INSERT INTO IndividualReservations
    (ReservationID, OrderID, Prepaid)
VALUES
    (@ReservationID, @OrderID, @Prepaid)

DECLARE @Discount AS DECIMAL(10, 2) = (100 - dbo.GetCurrentDiscount(@IndividualID, @OrderDate)) / 100

INSERT INTO OrderDetails
    (OrderID, ProductID, UnitPrice)
SELECT @OrderID, list.ProductID, (SELECT P.UnitPrice * @Discount
```

```
        FROM Products AS P
        WHERE list.ProductID = P.ProductID)
    FROM @ProductIDs as list

        END
    TRY
    BEGIN CATCH
    DECLARE @errorMsg nvarchar(1024) = N'An error occurred when adding an individual reservation: ' +
ERROR_MESSAGE();
    THROW 52000, @errorMsg, 1
    END CATCH
END
GO
```

Przykład użycia:

```
DECLARE @ProductIDs ProductList
INSERT @ProductIDs VALUES (2), (3), (4)
DECLARE @OrderDate DATETIME
SET @OrderDate = GETDATE()
EXEC AddIndividualReservation 1, '2023-03-01 12:00', '2023-03-01 14:00', 0, 1, @OrderDate, 4, @ProductIDs
```

Przykład kodu wyrzucający błąd

```
DECLARE @ProductIDs ProductList
INSERT @ProductIDs VALUES (2), (3), (4)
DECLARE @OrderDate DATETIME
SET @OrderDate = GETDATE()
EXEC AddIndividualReservation 1, '2023-03-01 12:00', '2023-03-01 14:00', 0, 1, @OrderDate, 1, @ProductIDs
```

AddNamedCompanyReservation

Dodaje rezerwację przyjmując listę Individuals. Przyjmuje

- CompanyID - ID firmy, która tworzy rezerwację
- ReservationDate - Data kiedy wykonana została rezerwacja
- StartDate - Początek rezerwacji
- EndDate - Koniec rezerwacji
- IndividualsIDs - Table-Valued Parameters wszystkich klientów wpisanych pod rezerwacją

```
CREATE PROCEDURE AddNamedCompanyReservation
   @CompanyID int,
   @ReservationDate datetime,
   @StartDate datetime,
   @EndDate datetime,
   @IndividualsIDs IndividualsList READONLY
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY

   IF NOT EXISTS(SELECT *
   FROM Companies
   WHERE CompanyID = @CompanyID)
   BEGIN;
       THROW 52000,'Company is not registered in database', 1
   END

   IF EXISTS (SELECT COUNT(*)
   FROM @IndividualsIDs
```

```
GROUP BY IndividualID
HAVING COUNT(*) > 1) BEGIN;
    THROW 52000, 'Can not add the same individual to the reservation twice.', 1
END

DECLARE @NumberOfGuests INT
SET @NumberOfGuests = (SELECT COUNT(*)
FROM @IndividualsIDs)

DECLARE @ClientID INT
SELECT @ClientID = ClientID
FROM Companies
WHERE CompanyID = @CompanyID

DECLARE @ReservationID INT
SELECT @ReservationID = ISNULL(MAX(ReservationID), 0) + 1
FROM Reservations


INSERT INTO Reservations
    (ReservationID, ClientID, ReservationDate, StartDate, EndDate, Accepted, NumberOfGuests)
VALUES
    (@ReservationID, @ClientID, @ReservationDate, @StartDate, @EndDate, 0, @NumberOfGuests)

INSERT INTO CompanyReservations
    (ReservationID)
VALUES
    (@ReservationID)
```

```
    INSERT INTO CompanyReservationsDetails
        (IndividualID, ReservationID)
    SELECT list.IndividualID, @ReservationID
    FROM @IndividualsIDs as list

    END TRY
    BEGIN CATCH
        DECLARE @errorMsg nvarchar(1024) = N'An error occurred when adding a named company reservation: ' +
ERROR_MESSAGE();
        THROW 52000, @errorMsg, 1
    END CATCH
END
GO
```

Przykład działania

```
DECLARE @IndividualIDs IndividualsList
INSERT @IndividualIDs VALUES (1), (2), (3)
DECLARE @ReservationDate DATETIME
SET @ReservationDate = GETDATE()
EXEC AddNamedCompanyReservation 6, @ReservationDate, '2023-04-04 17:00', '2023-04-04 18:00', @IndividualIDs
```

AddAnonymousCompanyReservation

Dodaje rezerwację bez określonych osób podpisanych pod rezerwacją. Przyjmuje

- CompanyID - ID firmy, która tworzy rezerwację
- ReservationDate - Data kiedy wykonana została rezerwacja
- StartDate - Początek rezerwacji
- EndDate - Koniec rezerwacji
- NumberOfGuests - Liczba osób na którą jest robiona rezerwacja

```
CREATE PROCEDURE AddAnonymousCompanyReservation
   @CompanyID int,
   @ReservationDate datetime,
   @StartDate datetime,
   @EndDate datetime,
   @NumberOfGuests int
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY

   IF NOT EXISTS(SELECT *
   FROM Companies
   WHERE CompanyID = @CompanyID)
   BEGIN;
       THROW 52000,'Company is not registered in database', 1
   END

   DECLARE @ClientID INT
   SELECT @ClientID = ClientID
```

```
    FROM Companies
    WHERE CompanyID = @CompanyID

    DECLARE @ReservationID INT
    SELECT @ReservationID = ISNULL(MAX(ReservationID), 0) + 1
    FROM Reservations


    INSERT INTO Reservations
        (ReservationID, ClientID, ReservationDate, StartDate, EndDate, Accepted, NumberOfGuests)
    VALUES
        (@ReservationID, @ClientID, @ReservationDate, @StartDate, @EndDate, 0, @NumberOfGuests)

    INSERT INTO CompanyReservations
        (ReservationID)
    VALUES
        (@ReservationID)

    END TRY
    BEGIN CATCH
        DECLARE @errorMsg nvarchar(1024) = N'An error occurred when adding an anonymous company reservation: '
+ ERROR_MESSAGE();
        THROW 52000, @errorMsg, 1
    END CATCH
END
GO
```

Przykład użycia:

```
DECLARE @IndividualIDs IndividualsList
INSERT @IndividualIDs
VALUES (1), (2), (3)
DECLARE @ReservationDate DATETIME
SET @ReservationDate = GETDATE()
EXEC AddNamedCompanyReservation 6, @ReservationDate, '2023-04-04 18:00', '2023-04-04 18:00', @IndividualIDs
```

AcceptReservation

Procedura akceptująca rezerwację oraz dopisująca stoliki do rezerwacji. Używa procedury AddTableToReservation. Przyjmuje

- ReservationID - ID rezerwacji, która ma być zaakceptowana
- TableIDs - Table-Valued Parameters z id oraz liczbą gości przy każdym stole

```
CREATE PROCEDURE AcceptReservation
   @ReservationID INT,
   @TableIDs TablesList READONLY
AS
BEGIN
   SET NOCOUNT ON

   DECLARE @TableID INT = 0
   DECLARE @NumberOfGuests INT = 0
   WHILE (1 = 1)
   BEGIN
       SELECT TOP 1
           @TableID = TableID,
           @NumberOfGuests = NumberOfGuests
       FROM @TableIDs
       WHERE TableID > @TableID
       ORDER BY TableID

       IF @@ROWCOUNT = 0 BREAK;

       EXEC dbo.AddTableToReservation @ReservationID, @NumberOfGuests, @TableID
   END
```

```
    UPDATE Reservations
    SET Accepted = 1
    WHERE ReservationID = @ReservationID
END
GO
```

Przykład użycia

```
DECLARE @TableIDs TablesList
INSERT @TableIDs VALUES (5, 3) -- (ID Stolika, liczba osób przy tym stoliku)
EXEC AcceptReservation 10, @TableIDs
```

AddEmployee

Dodaje pracownika do systemu. Przyjnuje:

- FirstName nvarchar(255) - Imie
- SureName nvarchar(255) - Nazwsko
- BirthDate date - Datę urodzenia

```
CREATE PROCEDURE AddEmployee
    @FirstName nvarchar(255),
    @SureName nvarchar(255),
    @BirthDate date
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY

    DECLARE @EmployeeID INT
```

```
    SELECT @EmployeeID = ISNULL(MAX(EmployeeID), 0) + 1
    FROM Employees

    INSERT INTO Employees
        (EmployeeID, FirstName, SureName, BirthDate)
    VALUES
        (@EmployeeID, @FirstName, @SureName, @BirthDate)

    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(1024)
        =N'An error occurred while adding an employee: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1;
    END CATCH
END
GO
```

MakeManager

Awansuje pracownika na stanowisko managera. Przyjmuje ID tego pracownika.

```
CREATE PROCEDURE MakeManager
    @EmployeeID int
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
```

```
IF NOT EXISTS (SELECT *
FROM Employees
WHERE EmployeeID = @EmployeeID)
BEGIN;
    THROW 52000, N'Employee with given EmployeeID number is not registered in the database', 1
END

IF EXISTS (SELECT *
FROM Administrators
WHERE EmployeeID = @EmployeeID)
BEGIN;
    THROW 52000, N'Employee with given EmployeeID is already an administrator', 1
END

IF EXISTS (SELECT *
FROM Managers
WHERE EmployeeID = @EmployeeID)
BEGIN;
    THROW 52000, N'Employee with given EmployeeID is already a manager', 1
END

INSERT INTO Managers
    (EmployeeID)
VALUES
    (@EmployeeID)

END TRY
BEGIN CATCH
```

```
        DECLARE @msg nvarchar(1024)
        =N'An error occurred while making employee a manager: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1;
    END CATCH
END
GO

        =N'An error occurred while making employee a manager: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1;
    END CATCH
END
GO
```

MakeAdministrator

Awansuje pracownika na stanowisko administratora. Przyjmuje ID tego pracownika.

```
CREATE PROCEDURE MakeAdministrator
    @EmployeeID int
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY

    IF NOT EXISTS (SELECT *
    FROM Employees
    WHERE EmployeeID = @EmployeeID)
    BEGIN;
```

```
    THROW 52000, N'Employee with given EmployeeID number is not registered in the database', 1
END

IF EXISTS (SELECT *
FROM Managers
WHERE EmployeeID = @EmployeeID)
BEGIN;
    THROW 52000, N'Employee with given EmployeeID is already a manager', 1
END

IF EXISTS (SELECT *
FROM Administrators
WHERE EmployeeID = @EmployeeID)
BEGIN;
    THROW 52000, N'Employee with given EmployeeID is already an administrator', 1
END

INSERT INTO Administrators
    (EmployeeID)
VALUES
    (@EmployeeID)

END TRY
BEGIN CATCH
    DECLARE @msg nvarchar(1024)
    =N'An error occurred while making employee an administrator: ' + ERROR_MESSAGE();
    THROW 52000, @msg, 1;
END CATCH
```

```
END
```

AddTable

Dodaje nowy stolik do systemu. Przyjmuje:

- Name nvarchar(255) - Nazwa stolika
- MaxNumberOfGuests int - Maksymalna ilość osób, która może siedzieć przy stoliku

```
CREATE PROCEDURE AddTable
   @Name nvarchar(255),
   @MaxNumberOfGuests INT

AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY

   IF EXISTS(SELECT *
   FROM Tables
   WHERE Name = @Name)
   BEGIN;
       THROW 52000,'Table with this name is already registered', 1
   END

   DECLARE @TableID INT
   SELECt @TableID = ISNULL(MAX(TableID), 0) + 1
   FROM Tables

   INSERT INTO Tables
```

```
        (TableID, Name, MaxNumberOfGuests)
    VALUES
        (@TableID, @Name, @MaxNumberOfGuests)

    END TRY
    BEGIN CATCH
        DECLARE @errorMsg nvarchar(1024) = N'An error occurred when adding a table: ' + ERROR_MESSAGE();
        THROW 52000, @errorMsg, 1
    END CATCH
END
GO
```

AddTableToReservation

Przypisuje rezerwacji stolik. Przyjmuje:

- ReservationID int - ID Rezerwacji
- NumberOfGuests int - Ilość gości
- TableID int - ID stolika podpinanego do rezerwacji

```
CREATE PROCEDURE AddTableToReservation
    @ReservationID int,
    @NumberOfGuests int,
    @TableID int
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
```

```
    IF NOT EXISTS(
    SELECT *
    FROM Tables
    WHERE TableID = @TableID
    )
    BEGIN;
        THROW 52000, 'Table was not registered in the database', 1
    END

    IF NOT EXISTS(
    SELECT *
    FROM Reservations
    WHERE ReservationID = @ReservationID
    )
    BEGIN;
        THROW 52000, 'No reservation with this ID was made', 1
    END

    -- Does not check if table is already used by another reservations
    INSERT INTO TableDetails
        (TableID, NumberOfGuests, ReservationID)
    VALUES
        (@TableID, @NumberOfGuests, @ReservationID)
END TRY
BEGIN CATCH
    DECLARE @msg nvarchar(1024) = N'An error occurred while adding a table to a reservation: ' +
ERROR_MESSAGE();
    THROW 52000, @msg, 1
```

```
END CATCH
END
GO
```

AddOrder

Procedura dodaje nowe zamówienie do tabeli Orders oraz wszystkie procedury związane z zamówieniem do OrderDetails. Przyjmuje argumenty:

- EmployeeID int - ID pracownika który obsługuje zamówienie
- OrderDate datetime - Moment dokonania zamówienia
- ServingDate datetime (DEFAUL NULL) - Czas kiedy serwowane jest zamówienie
- ClientID int - ID klienta, który wykonuje zamówienie
- ProductIDs - Table-Valued Parameters z listą produktów, które podpinane są pod zamówienie

```
CREATE PROCEDURE AddOrder
   @EmployeeID int,
   @OrderDate datetime,
   @ServingDate datetime = NULL,
   @ClientID int = NULL,
   @ProductIDs ProductList READONLY
AS
BEGIN
   BEGIN TRY
   IF NOT EXISTS (SELECT *
   FROM Employees
   WHERE EmployeeID = @EmployeeID) BEGIN;
       THROW 52000, 'Employee with given ID was not registered in the database', 1
   END
```

```
IF dbo.HasSeafood(@ProductIDs) = 1 BEGIN
    IF @ServingDate IS NULL BEGIN;
        THROW 52000, 'Cannot order seafood without specifying when it will be served', 1
    END
    IF NOT ((SELECT DATEPART(WEEKDAY, @ServingDate) - 1) BETWEEN 4 AND 6) BEGIN;
        THROW 52000, 'Trying to order seafood for days other than Thursday, Friday, Saturday', 1
    END
    IF dbo.CanOrderSeafood(@OrderDate, @ServingDate) = 0 BEGIN;
        THROW 52000, 'Must order seafood before tuesday preceding serving date', 1
    END
END

DECLARE @OrderID INT
SELECT @OrderID = ISNULL(MAX(OrderID), 0) + 1
FROM Orders

INSERT INTO Orders
    (OrderID, EmployeeID, OrderDate, ServingDate, ClientID)
VALUES
    (@OrderID, @EmployeeID, @OrderDate, @ServingDate, @ClientID)

DECLARE @Discount AS DECIMAL(10, 2) = 1
IF EXISTS(SELECT *
FROM Individuals
WHERE ClientID = @ClientID) BEGIN
    DECLARE @IndividualID INT
    SELECT @IndividualID = IndividualID
    FROM Individuals
```

```
        WHERE ClientID = @ClientID
        SET @Discount = (100 - dbo.GetCurrentDiscount(@IndividualID, @OrderDate)) / 100
    END

    INSERT INTO OrderDetails
        (OrderID, ProductID, UnitPrice)
    SELECT @OrderID, list.ProductID, (SELECT UnitPrice * @Discount
        FROM Products AS P
        WHERE list.ProductID = P.ProductID)
    FROM @ProductIDs as list

END
    TRY
BEGIN CATCH
    DECLARE @msg nvarchar(1024) = N'An error occurred while adding an order: ' + ERROR_MESSAGE();
    THROW 52000, @msg, 1
    END CATCH
END
GO
```

Przykład użycia:

```
DECLARE @OrderDate DATETIME
SET @OrderDate = GETDATE()
DECLARE @ProductIDs ProductList
INSERT @ProductIDs VALUES (2), (2), (3)
EXEC AddOrder 2, @OrderDate, NULL, 0, @ProductIDs
```

Przykład wyrzucający błąd ponieważ nie zadeklarowano daty serwowania podczas zamówienia owoców morza

```
DECLARE @OrderDate DATETIME
SET @OrderDate = GETDATE()
DECLARE @ProductIDs ProductList
INSERT @ProductIDs VALUES (2), (21), (3)
EXEC AddOrder 2, @OrderDate, NULL, 0, @ProductIDs
```

Przykład wyrzucający błąd ponieważ staramy się zamówić owoce morza na dzień inny niż czwartek, piątek, sobota

```
DECLARE @OrderDate DATETIME
SET @OrderDate = GETDATE()
DECLARE @ProductIDs ProductList
INSERT @ProductIDs VALUES (2), (21), (3)
EXEC AddOrder 2, @OrderDate, '2023-01-18 12:00', 0, @ProductIDs
```

Przykład wyrzucający błąd ponieważ staramy sie zamówić owoce morza za późno.

```
DECLARE @ProductIDs ProductList
INSERT @ProductIDs VALUES (2), (21), (3)
EXEC AddOrder 2, '2023-01-17 12:00', '2023-01-19 12:00', 0, @ProductIDs
```

MakeTakeaway

Procedura ustawia status zamówienia jako “na wynos”. Przyjmuje

- OrderID int - ID zamówienia które ma zmienić status
- PreferredDate datetime - Preferowany czas odbioru

```
CREATE PROCEDURE MakeTakeaway
   @OrderID int,
   @PreferredDate datetime
AS
BEGIN
   BEGIN TRY

   IF NOT EXISTS (SELECT *
   FROM Orders
   WHERE OrderID = @OrderID) BEGIN;
       THROW 52000, N'An order with given ID is not registered in the database', 1
   END

   IF EXISTS (SELECT *
   FROM Takeaway
   WHERE OrderID = @OrderID) BEGIN;
       THROW 52000, N'The order with given ID is already a takeaway', 1
   END

   INSERT INTO Takeaway
       (OrderID, PreferredDate)
   VALUES
       (@OrderID, @PreferredDate)
```

```
   END TRY
   BEGIN CATCH
       DECLARE @msg nvarchar(1024) = N'An error occurred while making an order a takeaway order: ' +
ERROR_MESSAGE();
       THROW 52000, @msg, 1
       END CATCH
END
GO
```

ChangeParameter

Procedura zmieniająca wartość parametru. Przyjmuje argumenty

- ParameterID varchar(2) - Nazwa parametru
- Value int - nowa wartość

```
CREATE PROCEDURE ChangeParameter
   @ParameterID varchar(2),
   @Value int
AS
BEGIN
   BEGIN TRY

   IF NOT EXISTS (SELECT *
   FROM Parameters
   WHERE ParameterID = @ParameterID) BEGIN;
```

```
        THROW 52000, N'An parameter with given name is not registered in the database', 1
    END
    UPDATE Parameters SET Value = @Value WHERE ParameterID = @ParameterID

END TRY
BEGIN CATCH
    DECLARE @msg nvarchar(1024) = N'An error occurred while changing a value of a parameter: ' +
ERROR_MESSAGE();
    THROW 52000, @msg, 1
END CATCH
END
GO
```

UpdateMenu

Weryfikuje czy bieżące menu jest poprawne. Jeśli tak, nie robi nic. Jeśli nie to

- usuwa produkty będące w nim dłużej niż 2 tygodnie względem najniższej sprzedaży,
- dodaje produkty nie będące aktualnie w menu o najlepszej sprzedaży.

```
-- UpdateMenu
CREATE PROCEDURE UpdateMenu
AS
BEGIN
    DECLARE @Count INT
    SET @Count = (
        SELECT COUNT(*)
        FROM menu as M
        INNER JOIN Products P on P.ProductID = M.ProductID
        INNER JOIN Categories C on C.CategoryID = P.CategoryID AND C.CategoryName <> 'Owoce morza'
        WHERE EndDate IS NULL OR EndDate > GETDATE())

    DECLARE @CountBad INT
    SET @CountBad = (
        SELECT COUNT(*)
        FROM menu as M
        INNER JOIN Products P on P.ProductID = M.ProductID
        INNER JOIN Categories C on C.CategoryID = P.CategoryID AND C.CategoryName <> 'Owoce morza'
        WHERE (EndDate IS NULL OR EndDate > GETDATE()) AND DATEDIFF(DAY, StartDate, GETDATE()) >= 14
    )

    IF 2 * @CountBad < @Count
        RETURN
```

```
DECLARE @CurrentProducts TABLE (MenuID INT);
INSERT INTO @CurrentProducts
    SELECT P.ProductID
    FROM menu as M
    INNER JOIN Products P on P.ProductID = M.ProductID
    INNER JOIN Categories C on C.CategoryID = P.CategoryID AND C.CategoryName <> 'Owoce morza'
    WHERE EndDate IS NULL OR EndDate > GETDATE()

DECLARE @CountChange INT
IF @CountBad = @Count / 2
    SET @CountChange = 1
ELSE
    SET @CountChange = @CountBad - (@Count / 2)

WHILE @CountChange > 0
BEGIN
    SET @CountChange = @CountChange - 1

    DECLARE @MenuID INT
    SET @MenuID = (SELECT A.MenuID FROM (
        SELECT TOP 1 M.MenuID, COUNT(*) as 'Sold'
        FROM Menu as M
        INNER JOIN Products P on P.ProductID = M.ProductID
        INNER JOIN Categories C on C.CategoryID = P.CategoryID AND C.CategoryName <> 'Owoce morza'
        INNER JOIN OrderDetails OD on P.ProductID = OD.ProductID
        WHERE (EndDate IS NULL OR EndDate > GETDATE()) AND DATEDIFF(DAY, M.StartDate, GETDATE()) >= 14
        GROUP BY M.MenuID
        ORDER BY 2
```

```
        ) AS A)

        DECLARE @NewProductID INT
        SET @NewProductID = (
            SELECT A.ProductID FROM (
                SELECT TOP 1 P.ProductID, COUNT(*) as 'Sold'
                FROM Products as P
                INNER JOIN Categories C on C.CategoryID = P.CategoryID AND C.CategoryName <> 'Owoce morza'
                INNER JOIN OrderDetails OD on P.ProductID = OD.ProductID
                WHERE P.ProductID NOT IN (SELECT * FROM @CurrentProducts)
                GROUP BY P.ProductID
                ORDER BY 2 DESC) AS A)

        DECLARE @NewMenuID INT
            SELECT @NewMenuID = ISNULL(MAX(MenuID), 0) + 1
            FROM Menu

        INSERT INTO Menu
            (MenuID, ProductID, StartDate, EndDate)
        VALUES
            (@NewMenuID, @NewProductID, GETDATE(), NULL);

        UPDATE Menu SET EndDate = GETDATE() - 1 WHERE MenuID = @MenuID
    END
END
go
```

Triggery

Trigger RemoveAllOrderDetailsTRIGGER

Po usunięciu zamówienia z listy zamówień usuwa wszystkie związane z nim detale z tabeli OrderDetails oraz Takeaway

```
CREATE TRIGGER RemoveAllOrderDetailsTRIGGER
ON Orders
ALTER TRIGGER [dbo].[RemoveAllOrderDetailsTRIGGER]
ON [dbo].[Orders]
FOR DELETE
AS
BEGIN
      DECLARE @orderID int = (Select OrderID from deleted);
      DELETE from OrderDetails where OrderID=@orderID;
      DELETE from Takeaway where OrderID = @orderID;
END
```

Trigger RemoveReservationDetailsTRIGGER

Kiedy anulowana jest jakaś rezerwacja, usuwane są powiązane z nią detale w tabelach IndividualReservations, CompanyReservations i CompanyReservationsDetails

```
CREATE TRIGGER RemoveReservationDetailsTRIGGER
ON Reservations
FOR DELETE
AS
BEGIN
      DECLARE @reservationID int = (Select ReservationID from deleted);
      DELETE from IndividualReservations where ReservationID = @reservationID;
      DELETE from CompanyReservations where ReservationID = @reservationID;
      DELETE from CompanyReservationsDetails where ReservationID = @reservationID;
      DELETE from TableDetails where ReservationID = @reservationID;
END
```

Trigger RemoveOrderFromReservationTRIGGER

Po usunięciu zamówienia indywidualnego, usuwa powiązane z nim zamówienie

```
CREATE TRIGGER RemoveOrderFromReservationTRIGGER
ON IndividualReservations
FOR DELETE
AS
BEGIN
      DECLARE @orderID int = (Select OrderID from deleted);
```

```
      DELETE from Orders where OrderID = @orderID;

END
```

Indeksy

Indeksy tabeli **Client**

```
CREATE UNIQUE INDEX idx_client_client_id ON Client (ClientID);
```

```
CREATE UNIQUE INDEX idx_client_client_id ON Client (ClientID);
```

```
CREATE UNIQUE NONCLUSTERED INDEX idx_client_phone ON Client (Phone);
```

Indeksy tabeli **Individuals**

```
CREATE NONCLUSTERED INDEX idx_individuals_clientid ON Individuals (ClientID);
```

```
CREATE NONCLUSTERED INDEX idx_individuals_name ON Individuals (FirstName, LastName);
```

Indeksy tabeli **Reservations**

```
CREATE UNIQUE INDEX idx_reservations_client_id ON Reservations (ClientID);
```

```
CREATE NONCLUSTERED INDEX idx_reservations_start_end_date ON Reservations (StartDate, EndDate);
```

```
CREATE NONCLUSTERED INDEX idx_reservations_reservation_date ON Reservations (ReservationDate);
```

Indeksy tabeli **CompanyReservations**

```
CREATE INDEX idx_company_reservations_reservation_id ON CompanyReservations (ReservationID);
```

Indeksy tabeli **CompanyReservationsDetails**

```
CREATE INDEX idx_company_reservations_details__individual_reservation_id ON CompanyReservationsDetails
(ReservationID, IndividualID);
```

Indeksy tabeli **IndividualReservations**

```
CREATE INDEX idx_individualreservations_reservationid ON IndividualReservations (ReservationID);
```

```
CREATE INDEX idx_individualreservations_prepaid ON IndividualReservations (Prepaid);
```

Indeksy tabeli **Employees**

```
CREATE UNIQUE INDEX idx_employees_employeeid ON Employees (EmployeeID);
```

```
CREATE NONCLUSTERED INDEX idx_employees_name ON Employees (FirstName, Surename);
```

```
CREATE NONCLUSTERED INDEX idx_employees_birthdate ON Employees (BirthDate);
```

Indeksy tabeli **Orders**

```
CREATE NONCLUSTERED INDEX idx_orders_client_id on Orders(ClientID)
```

```
CREATE NONCLUSTERED INDEX idx_orders_order_serving_date ON Orders (OrderDate, ServingDate)
```

Indeksy tabeli **OrderDetails**

```
CREATE INDEX idx_orderdetails_orderid ON OrderDetails (OrderID);
```

```
CREATE NONCLUSTERED INDEX idx_orderdetails_productid ON OrderDetails (ProductID);
```

Indeksy tabeli **Takeaway**

```
CREATE INDEX idx_takeaway_orderid ON Takeaway (OrderID);
```

```
CREATE NONCLUSTERED INDEX idx_takeaway_preferreddate ON Takeaway (PreferredDate);
```

Indeksy tabeli **TableDetails**

```
CREATE NONCLUSTERED INDEX idx_tabledetails_tableid ON TableDetails (TableID);
```

```
CREATE INDEX idx_tabledetails_reservationid ON TableDetails (ReservationID);
```

```
CREATE NONCLUSTERED INDEX idx_tabledetails_numberofguests ON TableDetails (NumberOfGuests);
```

Indeksy tabeli **Tables**

```
CREATE UNIQUE INDEX idx_tables_tableid ON Tables (TableID);
```

```
CREATE INDEX idx_tables_name ON Tables (Name);
```

```
CREATE NONCLUSTERED INDEX idx_tables_maxnumberofguests ON Tables (MaxNumberOfGuests);
```

Indeksy tabeli **Menu**

```
CREATE UNIQUE INDEX idx_menu_menuid ON Menu (MenuID);
```

```
CREATE NONCLUSTERED INDEX idx_menu_productid ON Menu (ProductID);
```

```
CREATE NONCLUSTERED INDEX idx_menu_startdate ON Menu (StartDate);
```

Indeksy tabeli **Products**

```
CREATE UNIQUE INDEX idx_products_productid ON Products (ProductID);
```

```
CREATE NONCLUSTERED INDEX idx_products_categoryid ON Products (CategoryID);
```

```
CREATE NONCLUSTERED INDEX idx_products_unitprice ON Products (UnitPrice);
```

Uprawnienia

Administrator

Jest to osoba, która ma pełny dostęp do bazy i jej funkcjonalności

```
CREATE ROLE Administrator
GRANT ALL TO Administrator
```

Customer

Jest to baza pod IndividualCustomer oraz CompanyCustomer

```
CREATE ROLE Customer
GRANT SELECT ON CurrentlyFreeTables TO Customer
GRANT SELECT ON CurrentMenu TO Customer
GRANT EXECUTE ON CanOrderSeafood TO Customer
GRANT EXECUTE ON GetPreviousMonday TO Customer
GRANT EXECUTE ON HasSeafood TO Customer
```

IndividualCustomer

Klient indywidualny - może np sprawdzać swoje obecne zniżki, oraz zmieniać status zamówienia na “na wynos”

```
CREATE ROLE IndividualCustomer
GRANT Customer TO IndividualCustomer
GRANT SELECT ON IndividualReservationsHistory TO IndividualCustomer
GRANT SELECT ON CurrentDiscounts TO IndividualCustomer
GRANT EXECUTE ON GetCurrentDiscount TO IndividualCustomer
GRANT EXECUTE ON AddIndividualReservation TO IndividualCustomer
GRANT EXECUTE ON AddOrder TO IndividualCustomer
GRANT EXECUTE ON MakeTakeaway TO IndividualCustomer
```

CompanyCustomer

Klient firmowy - Może np tworzyć rezerwację firmową oraz stworzyć nową rezerwację

```
CREATE ROLE CompanyCustomer
GRANT Customer TO CompanyCustomer
GRANT SELECT ON CompanyReservationsHistory TO CompanyCustomer
GRANT EXECUTE ON AddOrder TO CompanyCustomer
GRANT EXECUTE ON AddNamedCompanyReservation TO Employee
GRANT EXECUTE ON AddAnonymousCompanyReservation TO Employee
GRANT EXECUTE ON MakeTakeaway TO CompanyCustomer
```

Employee

Uprawnienia pracownika - Może on np sprawdzać nadchodzące zamówienia, dodawać klientów oraz sprawdzać które stoły są wolne

```
CREATE ROLE Employee
GRANT SELECT ON CompanyOrders TO Employee
GRANT SELECT ON IndividualOrders TO Employee
GRANT SELECT ON IndividualReservationsHistory TO Employee
GRANT SELECT ON CurrentDiscounts TO Employee
GRANT SELECT ON CurrentlyFreeTables TO Employee
GRANT SELECT ON CurrentMenu TO Employee
GRANT SELECT ON ProductDetails TO Employee
GRANT SELECT ON UpcommingTakeawayOrders TO Employee
GRANT SELECT ON UpcommingReservations TO Employee
GRANT SELECT ON CompanyReservationsHistory TO Employee
GRANT SELECT ON SeaFoodOrders TO Employee
GRANT EXECUTE ON CanOrderSeafood TO Employee
GRANT EXECUTE ON GetPreviousMonday TO Employee
GRANT EXECUTE ON HasSeafood TO Employee
GRANT EXECUTE ON AddIndividual TO Employee
GRANT EXECUTE ON AddComapny TO Employee
GRANT EXECUTE ON AddIndividualReservation TO Employee
GRANT EXECUTE ON AddNamedCompanyReservation TO Employee
GRANT EXECUTE ON AddAnonymousCompanyReservation TO Employee
GRANT EXECUTE ON AddOrder TO Employee
GRANT EXECUTE ON MakeTakeaway TO Employee
GRANT EXECUTE ON VerifyMenu TO Employee
GRANT EXECUTE ON UpdateMenu TO Employee
```

Manager

Uprawnienia Kierownika - Może on np sprawdzać widoki związane ze statystykami produktów oraz pracowników. Może także akceptować rezerwacje.

```
CREATE ROLE Manager
GRANT Employee TO Manager
GRANT SELECT ON EmployeeStats TO Manager
GRANT SELECT ON GeneralProductStats TO Manager
GRANT SELECT ON MonthlyOrderStats TO Manager
GRANT SELECT ON WeeklyOrderStats TO Manager
GRANT SELECT ON MonthlyProductStats TO Manager
GRANT SELECT ON ProductDetails TO Manager
GRANT SELECT ON ReservationsToCheck TO Manager
GRANT SELECT ON TableReservationDetails TO Manager
GRANT SELECT ON MonthlyCompanyOrders TO Manager
GRANT SELECT ON AverageServiceTime TO Manager
GRANT SELECT ON AverageProductServingTime TO Manager
GRANT EXECUTE ON AddToMenu TO Manager
GRANT EXECUTE ON AcceptReservation TO Manager
GRANT EXECUTE ON AddEmployee TO Manager
GRANT EXECUTE ON MakeManager TO Manager
GRANT EXECUTE ON AddTable To Manager
GRANT EXECUTE ON AddTableToReservation To Manager
GRANT EXECUTE ON ChangeParameter To Manager
```