# Projekt - Podstawy Baz Danych

*Tomasz Koszarek, Jakub Koźlak*

*Projekt dotyczy systemu wspomagania działalności firmy świadczącej usługi gastronomiczne dla klientów indywidualnych oraz firm.*

## *I. Analiza wymagań*

1. **Firma oferuje w sprzedaży dania (Dishes)**
   - każde danie ma nazwę (DishName) oraz identyfikator (DishID)
   - każde danie należy do jednej z kategorii (CategoryID), posiada swoją cenę (UnitPrice), która jest stała dla każdego zamówienia (nie biorąc pod uwagę rabatów dla klientów).
   - danie składa się z jednego lub więcej składników (nazywanych dalej półproduktami) w konkretnej, określonej wcześniej ilości.
   - danie może zniknąć z menu z powodu wyczerpania się półproduktów.

2. **Firma może posiadać kilka restauracji (Restaurants)**
   - każda restauracja ma nazwę (RestaurantName) oraz identyfikator (RestaurantID)
   - każda restauracja ma swoje menu, własne stoliki i swoich klientów.

3. **Produkty w menu (Menu Dishes)**, czyli żywność i napoje,
   - encja przechowująca pozycje w menu z przeszłości, pozycje aktualnie dostępne oraz te zaplanowane na przyszłe dni. Dzięki temu możliwe jest ustalanie menu z dowolnym wyprzedzeniem, co najmniej jednego dnia, oraz śledzenie historii posiłków w karcie, po to, żeby móc ponownie wprowadzić pozycję po minimum miesiącu.
   - dane są postaci: danie (DishID), data początkowa (StartDate) oraz data końcowa (EndDate).

4. **Kategorie (Categories)**
   - to grupy, do których mogą należeć dania. Ułatwi to organizację i zwiększy czytelność menu, przykładowo posiłki dzielić można na lunche, śniadania, napoje lub dostępne w wybrane dni owoce morza.
   - kategorie posiadają nazwy (CategoryName) oraz identyfikator (CategoryID)

5. **Półprodukty (HalfProducts), czyli składniki dań:**
   - mają nazwę (HalfProductName) oraz identyfikator(HalfProductID). Jeden półprodukt może być składnikiem wielu dań (a jedno danie może składać się z kilku półproduktów).
   - każdy półprodukt ma określoną ilość dostępną w magazynie (UnitsInStock), na której podstawie można określić ilość dań możliwych jeszcze do wydania.

6. **Zamówienia (Orders)**
   - rozpoznawane są po unikatowych numerach (OrderID). Zamówienie jest składane przez klienta (CustomerID), którym może być firma lub osoba prywatna. Jedno zamówienie może się składać z wielu dań. Danie może występować wiele razy (Quantity) w jednym zamówieniu.
   - moment złożonego zamówienia (Orderdate) może różnić się od preferowanego momentu odbioru zamówienia (PreferredRealisationDate). Dzięki temu można składać zamówienia przed przyjściem do restauracji, lub zamówić catering z wyprzedzeniem. Przez to też, w dniach czwartek do soboty, zjeść można owoce morza, jednak zamówienie takie musi być maksymalnie do poniedziałku w tygodniu poprzednim.
   - zamówienie złożone z wyprzedzeniem może być połączone z rezerwacją stolika (ReservationID), a płatność jest możliwa do zrealizowania po otrzymaniu zamówienia (Paid).
   - zamówienia składane mogą być przez stronę internetową (isWeb)
   - jedno zamówienie może zawierać kilka dań oraz jedno danie może być składnikiem kilku zamówień (relacja wiele do wielu)

7. **Przy składaniu zamówienia z wyprzedzeniem, możliwa jest rezerwacja stolika (Reservations)**
   - możliwość rezerwacji stolików występuje przy składaniu zamówień przez internet, wypełniając formularz na stronie firmy. Rezerwacje dostępne są dla klientów indywidualnych, którzy spełnili jeden z dwóch warunków:
     1. złożenie wcześniej 5 zamówień na kwotę przynajmniej 50 zł (kwota zamówienia obliczana jest na podstawie produktów składających się na dane zamówienie oraz rabatów przypisanych klientowi)

2. złożenie zamówienia na kwotę przynajmniej 200 zł.

- każda rezerwacja posiada unikalny numer (ReservationID)
- rezerwowany jest wybrany z dostępnych stolików (Tableid)
- rezerwacji dokonać można na konkretną godzinę (ReservationDate)
- rezerwacji dokonuje klient (CustomerID)
- dla firm możliwa jest rezerwacja na całą firmę, lub na nazwisko jednego pracownika (Name)
- wprowadzone w pewnym momencie obostrzenia związane z ograniczeniem liczby osób przebywających na raz w restauracji mogą powodować konieczność odwoływania rezerwacji. Na podstawie daty złożenia rezerwacji (na korzyść tych złożonych najwcześniej) ustalane jest które z nich muszą zostać odwołane.

## 8. Stoliki (Tables)

- posiadają unikalne numery (TableID) oraz określoną liczbę miejsc (Seats)
- dzięki informacji o liczbie siedzeń możemy podczas rezerwacji udostępnić klientowi stolik o wybranej wielkości.
- na podstawie odgórnych obostrzeń narzucających liczbę miejsc w restauracjach możliwe jest wyłączanie stolików z użytku oraz zmiana ilości dostępnych miejsc przy stoliku

## 9. Obostrzenia covidowe na stoliki

- na podstawie odgórnych obostrzeń narzucających liczbę miejsc w restauracjach możliwe jest wyłączanie stolików z użytku oraz zmiana ilości dostępnych miejsc przy stoliku

## 10. Klienci indywidualni (PersonCustomers)

- mają mają unikatowy numer (CustomerID).
- przechowujemy ich imiona (FirstName) i nazwiska (LastName) oraz numer telefonu (Phone)
- w przypadku, gdy klient zamawia przez internet, wymagany jest również adres email (Email)
- klienci indywidualni mogą mieć przypisane rabaty (DiscountID)

## 11. Klienci- firmy (CompanyCustomer)

- posiadają nazwę (CompanyName) oraz unikatowy numer (CustomerID).

- przechowywany jest numer telefonu (Phone), adres firmy (Address), numer NIP (NIP)
- klienci firmowi mogą mieć przypisane rabaty (DiscountID)

## 12. Rabaty (Discounts)

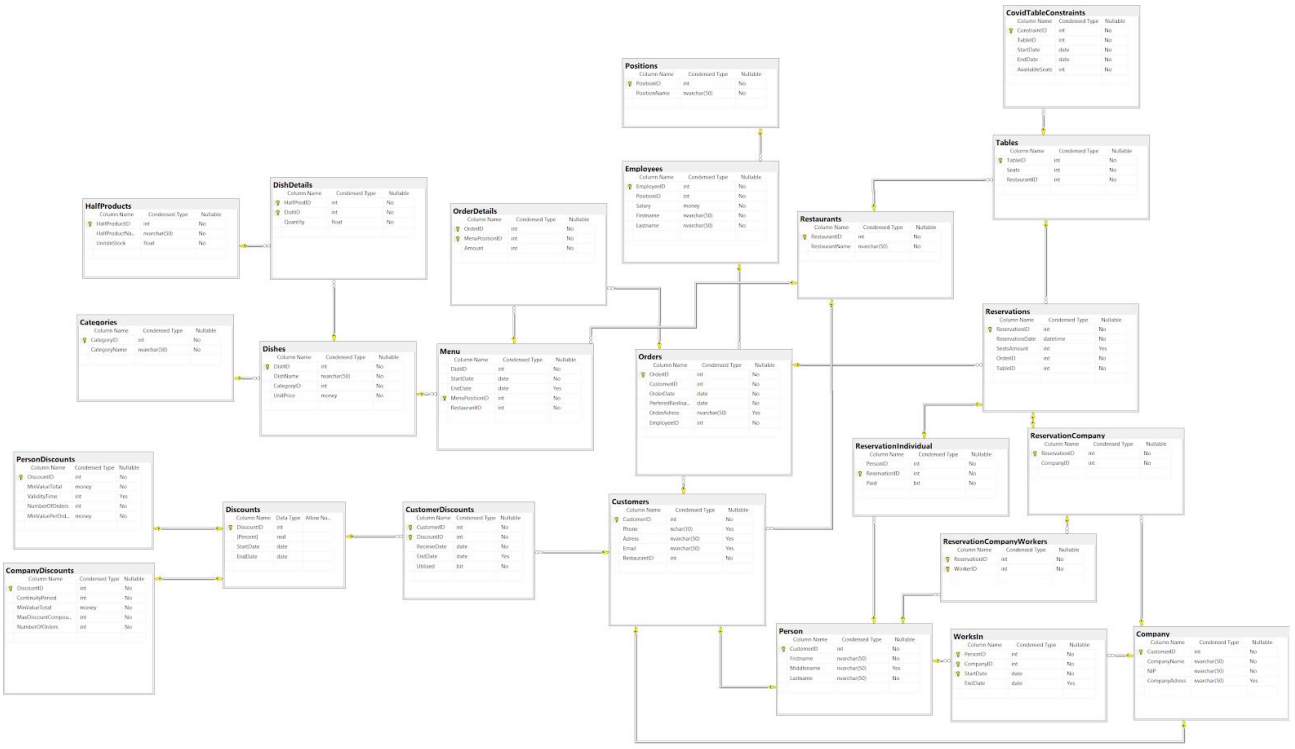
- mają unikalny numer (DiscountID)
- są przypisane do klienta (CustomerID), jeden klient może mieć kilka rabatów
- rabaty są procentowe (Amount)
- przechowywana jest data przyznania rabatu dla klienta (StartDate)
- klienci indywidualni
  1. po spełnieniu wymagań do przyznania rabatu mogą uzyskać rabat stały, procentowy, który nie ulega wygaśnięciu.
  2. klient indywidualny może uzyskać również rabat czasowy, aktywny przez 7 dni od dnia przyznania
- klienci firmowi również mogą otrzymać rabaty:
  1. procentowy, którego aktualność zależy od ciągłości zamówień. Rabat może ulec wygaśnięciu
     - rabaty procentowe, sumując się nie mogą przekroczyć zadanej wartości procentowej
  2. kwotowy, równy odpowiedniemu procentowi łącznej kwoty zamówień

## 13. Generowanie raportów

- System umożliwia generowanie raportów podsumowujących dany przedział czasowy, tygodnia, miesiąca itp. Mogą one dotyczyć stolików, menu, rabatów itd. Jest to możliwe dzięki przechowywaniu w bazie danych dat zamówień, rezerwacji itp.

---

## II. Diagram




[<< link do diagramu w lepszej jakości >>]

---

## III. Tabele

1. **Menu**

- **MenuPositionID** (int) - identyfikator, klucz główny, autoinkrementowany (1, 1)
- **DishID** (int) - klucz obcy, łączy ze słownikiem Dishes
- **StartDate** (date) – data wejścia potrawy do menu
- **EndDate** (date) – data wyjścia potrawy z menu. Wartość null w tej kolumnie oznacza brak zaplanowanej daty wyjścia z menu)
- **RestaurantID** (int) - klucz obcy do restauracji

**Warunki:**

- Data **EndDate** późniejsza niż **StartDate.**
- Para **MenuPositionID DishID** unikalna

```
CREATE TABLE [dbo].[Menu](
```

```
     [DishID] [int] NOT NULL,
     [StartDate] [date] NOT NULL,
     [EndDate] [date] NULL,
     [MenuPositionID] [int] IDENTITY(1,1) NOT NULL,
     [RestaurantID] [int] NOT NULL,
 CONSTRAINT [PK_Menu] PRIMARY KEY CLUSTERED
(
     [MenuPositionID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Menu]  WITH CHECK
ADD  CONSTRAINT [CK_Menu] CHECK  (([startdate]<[enddate]))
GO
ALTER TABLE [dbo].[Menu] CHECK CONSTRAINT [CK_Menu]
GO
ALTER TABLE [dbo].[Menu]  WITH CHECK
ADD  CONSTRAINT [FK_Menu_Dishes] FOREIGN KEY([DishID])
REFERENCES [dbo].[Dishes] ([DishID])
GO
ALTER TABLE [dbo].[Menu] CHECK CONSTRAINT [FK_Menu_Dishes]
GO
ALTER TABLE [dbo].[Menu]  WITH CHECK
ADD  CONSTRAINT [FK_Menu_Restaurants] FOREIGN KEY([RestaurantID])
REFERENCES [dbo].[Restaurants] ([RestaurantID])
GO
ALTER TABLE [dbo].[Menu] CHECK CONSTRAINT [FK_Menu_Restaurants]
GO
```

2. **Dishes**

- **DishID** (int) – identyfikator, klucz główny, autoinkrementowany
- **Dish Name** (nvarchar(50))– nazwa dania
- **CategoryID** (int) – klucz obcy do słownika kategorii
- **Unit Price** (money) – cena za jedną jednostką dania

  **Warunki:**

  - **Unit Price** większe niż 0.
  - **Dish Name** unikalne

```
CREATE TABLE [dbo].[Dishes](
     [DishID] [int] IDENTITY(1,1) NOT NULL,
     [DishName] [nvarchar](50) NOT NULL,
     [CategoryID] [int] NOT NULL,
```

```
     [UnitPrice] [money] NOT NULL,
 CONSTRAINT [PK_Dishes] PRIMARY KEY CLUSTERED
(
     [DishID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY],
 CONSTRAINT [dishid] UNIQUE NONCLUSTERED
(
     [DishID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY],
 CONSTRAINT [UC2_Dishes] UNIQUE NONCLUSTERED
(
     [Dish Name] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Dishes]  WITH CHECK
ADD  CONSTRAINT [FK_Dishes_Categories] FOREIGN KEY([CategoryID])
REFERENCES [dbo].[Categories] ([CategoryID])
GO
ALTER TABLE [dbo].[Dishes] CHECK CONSTRAINT [FK_Dishes_Categories]
GO
ALTER TABLE [dbo].[Dishes]  WITH CHECK
ADD  CONSTRAINT [CK_Dishes] CHECK  (([Unitprice]>(0)))
GO
ALTER TABLE [dbo].[Dishes] CHECK CONSTRAINT [CK_Dishes]
GO
```

3. **DishDetails**

- **HalfProductID** (int) – klucz obcy do słownika półproduktów
- **DishID** (int) – klucz obcy do słownika dań
- **Quantity** (float) – ilość półproduktu potrzebnego na jedno danie wyrażona w [g]

  **Warunki:**
  - **Quantity** większe od zera.
  - Para (**HalfProductID, DishID)** jest kluczem głównym

```
CREATE TABLE [dbo].[DishDetails](
     [HalfProdID] [int] NOT NULL,
     [DishID] [int] NOT NULL,
     [Quantity] [float] NOT NULL,
 CONSTRAINT [PK_DishDetails] PRIMARY KEY CLUSTERED
```

```
(
     [HalfProdID] ASC,
     [DishID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GOALTER TABLE [dbo].[DishDetails]  WITH CHECK
ADD  CONSTRAINT [FK_DishDetails_Dishes] FOREIGN KEY([DishID])
REFERENCES [dbo].[Dishes] ([DishID])
GO
ALTER TABLE [dbo].[DishDetails] CHECK CONSTRAINT [FK_DishDetails_Dishes]
GO
ALTER TABLE [dbo].[DishDetails]  WITH CHECK
ADD  CONSTRAINT [FK_DishDetails_HalfProducts] FOREIGN KEY([HalfProdID])
REFERENCES [dbo].[HalfProducts] ([HalfProductID])
GO
ALTER TABLE [dbo].[DishDetails] CHECK CONSTRAINT
[FK_DishDetails_HalfProducts]
GO
ALTER TABLE [dbo].[DishDetails]  WITH CHECK
ADD  CONSTRAINT [CK_DishDetails] CHECK  (([quantity]>(0)))
GO
ALTER TABLE [dbo].[DishDetails] CHECK CONSTRAINT [CK_DishDetails]
GO
```

4. **HalfProducts**

- **HalfProductID** (int) – identyfikator, klucz główny, autoinkrementowany
- **HalfProductName** (nvarchar(50)) – nazwa półproduktu
- **UnitsInStock** (float) – ilość produktu aktualnie przechowywana w magazynie (w kg)

  **Warunki:**

  - **UnitsInStock** większe lub równe 0

```
CREATE TABLE [dbo].[HalfProducts](
     [HalfProductID] [int] IDENTITY(1,1) NOT NULL,
     [HalfProductName] [nvarchar](50) NOT NULL,
     [UnitsInStock] [float] NOT NULL,
 CONSTRAINT [PK_HalfProducts] PRIMARY KEY CLUSTERED
(
     [HalfProductID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
```

```
GO
ALTER TABLE [dbo].[HalfProducts]  WITH CHECK
ADD  CONSTRAINT [CK_HalfProducts] CHECK  (([unitsinstock]>=(0)))
GO
ALTER TABLE [dbo].[HalfProducts] CHECK CONSTRAINT [CK_HalfProducts]
GO
```

5. **Categories**

- **CategoryID** (int) – identyfikator, klucz główny
- **CategoryName** (nvarchar(50)) – nazwa kategorii

  **Warunki:**

  - **CategoryName** unikalna

```
CREATE TABLE [dbo].[Categories](
     [CategoryID] [int] IDENTITY(1,1) NOT NULL,
     [CategoryName] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_Categories] PRIMARY KEY CLUSTERED
(
     [CategoryID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY],
 CONSTRAINT [UC_CATEGORIES] UNIQUE NONCLUSTERED
(
     [CategoryName] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

6. **Orders**

- **OrderID** (int) – identyfikator, klucz główny, autoinkrementowany
- **CustomerID** (int) – klucz obcy łączący z tabelą klientów. Oznacza Id klienta, który złożył zamówienie
- **EmployeeID** (int) – klucz obcy łączący z tabelą pracowników firmy, oznacza pracownika odpowiedzialnego za obsłużenie zamówienia
- **OrderDate** (date) – data, w której zamówienie zostało złożone.

- **PreferedRealisationDate** (date) – data preferowana przez klienta do wykonania zamówienia. Jeśli jest ona późniejsza, niż data złożenia zamówienia, oznacza zamówienie złożone z wyprzedzeniem. W pozostałych przypadkach jest ona równa dacie złożenia zamówienia.
- **OrderAdress** (nvarchar(50)) – adres, pod który zamówienie ma być dostarczone. Wartość null oznacza zamówienie na miejscu

  **Warunki**:

  - Data **OrderDate** wcześniejsza lub równa **PreferedRealisationDate**
  - **OrderDate** i **PreferedRealisationDate** domyślnie obecną datą

```
CREATE TABLE [dbo].[Orders](
     [OrderID] [int] IDENTITY(1,1) NOT NULL,
     [CustomerID] [int] NOT NULL,
     [OrderDate] [date] NOT NULL,
     [PreferedRealisationDate] [date] NOT NULL,
     [OrderAdress] [nvarchar](50) NULL,
     [EmployeeID] [int] NOT NULL,
 CONSTRAINT [PK_Orders] PRIMARY KEY CLUSTERED
(
     [OrderID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Orders]  WITH CHECK
ADD  CONSTRAINT [FK_Orders_Customers] FOREIGN KEY([CustomerID])
REFERENCES [dbo].[Customers] ([CustomerID])
GO
ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [FK_Orders_Customers]
GO
ALTER TABLE [dbo].[Orders]  WITH CHECK
ADD  CONSTRAINT [FK_Orders_Employees] FOREIGN KEY([EmployeeID])
REFERENCES [dbo].[Employees] ([EmployeeID])
GO
ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [FK_Orders_Employees]
GO
ALTER TABLE [dbo].[Orders]  WITH CHECK
ADD  CONSTRAINT [CK_Orders] CHECK
(([orderdate]<=[preferedrealisationdate]))
GO
ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [CK_Orders]
GO
ALTER TABLE [dbo].[Orders]
ADD CONSTRAINT df_Orders1
DEFAULT GETDATE() FOR [OrderDate]
ALTER TABLE [dbo].[Orders]
```

```
ADD CONSTRAINT df_Orders2
DEFAULT GETDATE() FOR [PreferedRealisationDate]
```

7. **OrderDetails**

- **OrderID** (int) – klucz obcy łączący z tabelą zamówień
- **MenuPositionID** (int) – klucz obcy do tabeli z pozycjami w menu
- **Amount** (int) – liczba jednostek danego dania w zamówieniu

  **Warunki:**

  - **Amount** większe niż 0.
  - Para ( **OrderID, MenuPositionID)** jest kluczem głównym

```
CREATE TABLE [dbo].[OrderDetails](
     [OrderID] [int] NOT NULL,
     [MenuPositionID] [int] NOT NULL,
     [Amount] [int] NOT NULL,
 CONSTRAINT [PK_OrderDetails] PRIMARY KEY CLUSTERED
(
     [OrderID] ASC,
     [MenuPositionID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[OrderDetails]  WITH CHECK
ADD  CONSTRAINT [FK_OrderDetails_Menu] FOREIGN KEY([MenuPositionID])
REFERENCES [dbo].[Menu] ([MenuPositionID])
GO
ALTER TABLE [dbo].[OrderDetails] CHECK CONSTRAINT [FK_OrderDetails_Menu]
GO
ALTER TABLE [dbo].[OrderDetails]  WITH CHECK
ADD  CONSTRAINT [FK_OrderDetails_Orders] FOREIGN KEY([OrderID])
REFERENCES [dbo].[Orders] ([OrderID])
GO
ALTER TABLE [dbo].[OrderDetails] CHECK CONSTRAINT [FK_OrderDetails_Orders]
GO
ALTER TABLE [dbo].[OrderDetails]  WITH CHECK
ADD  CONSTRAINT [CK_OrderDetails] CHECK  (([amount]>(0)))
GO
ALTER TABLE [dbo].[OrderDetails] CHECK CONSTRAINT [CK_OrderDetails]
GO
```

8. **Employees**

- **EmployeeID** (int) – identyfikator pracownika, klucz główny tabeli, autonkrementowany
- **PositionID** (int) – klucz obcy do tabeli pozycji pracowników
- **Salary** (money) – obecna miesięczna płaca pracownika
- **Firstname** (nvarchar(50)) - imię pracownika
- **Lastname** (nvarchar(50)) - nazwisko pracownika

  **Warunki:**

  - **Salary** większe od 0.

```
CREATE TABLE [dbo].[Employees](
     [EmployeeID] [int] IDENTITY(1,1) NOT NULL,
     [PositionID] [int] NOT NULL,
     [Salary] [money] NOT NULL,
     [Firstname] [nvarchar](50) NOT NULL,
     [Lastname] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_Employees] PRIMARY KEY CLUSTERED
(
     [EmployeeID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Employees]  WITH CHECK
ADD  CONSTRAINT [FK_Employees_Positions] FOREIGN KEY([PositionID])
REFERENCES [dbo].[Positions] ([PositionID])
GO
ALTER TABLE [dbo].[Employees] CHECK CONSTRAINT [FK_Employees_Positions]
GO
ALTER TABLE [dbo].[Employees]  WITH CHECK
ADD  CONSTRAINT [CK_Employees] CHECK  (([salary]>(0)))
GO
ALTER TABLE [dbo].[Employees] CHECK CONSTRAINT [CK_Employees]
GO
```

9. **Positions**

- **PositionID** (int) – identyfikator pozycji, klucz główny tabeli
- **PositionName** (nvarchar(50)) – nazwa pozycji

  **Warunki:**

  - **PositionName** unikalne

```
CREATE TABLE [dbo].[Positions](
     [PositionID] [int] IDENTITY(1,1) NOT NULL,
```

```
     [PositionName] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_Positions] PRIMARY KEY CLUSTERED
(
     [PositionID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY],
 CONSTRAINT [UC_POSITIONS] UNIQUE NONCLUSTERED
(
            [PositionName] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

**10. Tables**

- **TableID** (int) – identyfikator stolika, autoinkrementowany
- **Seats** (int) – liczba miejsc przy stoliku, czyli maksymalna liczba miejsc na którą można zarezerwować dany stolik.
- **RestaurantID** (int) - identyfikator restauracji

  **Warunki:**

  - **Seats** większe od 0.

```
CREATE TABLE [dbo].[Tables](
     [TableID] [int] IDENTITY(1,1) NOT NULL,
     [Seats] [int] NOT NULL,
     [RestaurantID] [int] NOT NULL,
 CONSTRAINT [PK_Tables] PRIMARY KEY CLUSTERED
(
     [TableID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Tables]  WITH CHECK
ADD  CONSTRAINT [CK_Tables] CHECK  (([seats]>(0)))
GO
ALTER TABLE [dbo].[Tables] CHECK CONSTRAINT [CK_Tables]
GO
ALTER TABLE [dbo].[Tables]  WITH CHECK
ADD  CONSTRAINT [FK_Tables_Restaurants] FOREIGN KEY([RestaurantID])
REFERENCES [dbo].[Restaurants] ([RestaurantID])
GO
ALTER TABLE [dbo].[Tables] CHECK CONSTRAINT [FK_Tables_Restaurants]
GO
```

**11. CovidTableConstraints**

- **ConstraintID** (int) – identyfikator obostrzenia, autoinkrementowany
- **TableID** (int) – klucz obcy łączący z tabelą stolików
- **StartDate** (date) – data od której obowiązuje ograniczenie liczby miejsc przy stoliku
- **EndDate** (date) – data zakończenia działania obostrzenia
- **AvailableSeats**(int) – liczba dostępnych w tym okresie siedzeń przy stoliku

  **Warunki:**

  - **AvailableSeats** w przedziale od 0 do **Tables.Seats** dla danego stolika
  - Data **EndDate** późniejsza niż **StartDate**
  - Trójka **( TableID, StartDate, EndDate)** unikalna

```
CREATE TABLE [dbo].[CovidTableConstraints](
     [ConstraintID] [int] IDENTITY(1,1) NOT NULL,
     [TableID] [int] NOT NULL,
     [StartDate] [date] NOT NULL,
     [EndDate] [date] NOT NULL,
     [AvailableSeats] [int] NOT NULL,
 CONSTRAINT [PK_CovidTableConstraints] PRIMARY KEY CLUSTERED
(
     [ConstraintID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY],
 CONSTRAINT [UC_CovidTableConstaint] UNIQUE NONCLUSTERED
(
     [TableID] ASC,
     [StartDate] ASC,
     [EndDate] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[CovidTableConstraints]  WITH CHECK
ADD  CONSTRAINT [CK_CovidTableConstraints] CHECK
(([startdate]<[enddate]))
GO
ALTER TABLE [dbo].[CovidTableConstraints] CHECK CONSTRAINT
[CK_CovidTableConstraints]
GO
ALTER TABLE [dbo].[CovidTableConstraints]  WITH CHECK
ADD  CONSTRAINT [CK_CovidTableConstraints1] CHECK
(([AvailableSeats]>(0)))
GO
```

```
ALTER TABLE [dbo].[CovidTableConstraints] CHECK CONSTRAINT
[CK_CovidTableConstraints1]
GO
ALTER TABLE [dbo].[CovidTableConstraints]  WITH CHECK
ADD  CONSTRAINT [FK_CovidTableConstraints_Tables] FOREIGN KEY([TableID])
REFERENCES [dbo].[Tables] ([TableID])
GO
ALTER TABLE [dbo].[CovidTableConstraints] CHECK CONSTRAINT
[FK_CovidTableConstraints_Tables]
GO
```

**12. Customers**

- **CustomerID** (int) – identyfikator klienta, klucz główny do tabeli, autoinkrementowany
- **Phone** (nchar(10)) – numer telefonu klienta
- **Adress** (nvarchar(50)) – adres korespondencyjny klienta
- **Email** (nvarchar(50)) – adres email klienta
- **RestaurantID** (int) - identyfikator restauracji w której pracuje klient

  **Warunki**:
  - **Email** zawiera „@”
  - **Phone** składa się wyłącznie z cyfr

```
CREATE TABLE [dbo].[Customers](
     [CustomerID] [int] IDENTITY(1,1) NOT NULL,
     [Phone] [nchar](10) NULL,
     [Adress] [nvarchar](50) NULL,
     [Email] [nvarchar](50) NULL,
     [RestaurantID] [int] NOT NULL,
 CONSTRAINT [PK_Customers] PRIMARY KEY CLUSTERED
(
     [CustomerID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Customers]  WITH CHECK
ADD  CONSTRAINT [CK_Customers] CHECK  (([email] like '%@%'))
GO
ALTER TABLE [dbo].[Customers] CHECK CONSTRAINT [CK_Customers]
GO
ALTER TABLE [dbo].[Customers]  WITH CHECK
ADD  CONSTRAINT [CK_Customers_1] CHECK  ((isnumeric([phone])=(1)))
GO
ALTER TABLE [dbo].[Customers] CHECK CONSTRAINT [CK_Customers_1]
```

```
GO
ALTER TABLE [dbo].[Customers]  WITH CHECK
ADD  CONSTRAINT [FK_Customers_Restaurants] FOREIGN KEY([RestaurantID])
REFERENCES [dbo].[Restaurants] ([RestaurantID])
GO
ALTER TABLE [dbo].[Customers] CHECK CONSTRAINT [FK_Customers_Restaurants]
GO
```

**13. Person**

- **CustomerID** (int) – identyfikator klienta, klucz główny do tabeli
- **Firstname** (nvarchar(50)) – imię klienta
- **Middlename** (nvarchar(50)) – drugie imię klienta
- **Lastname** (nvarchar(50)) – nazwisko klienta

```
CREATE TABLE [dbo].[Person](
     [CustomerID] [int] NOT NULL,
     [Firstname] [nvarchar](50) NOT NULL,
     [Middlename] [nvarchar](50) NULL,
     [Lastname] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_Person_1] PRIMARY KEY CLUSTERED
(
     [CustomerID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Person]  WITH CHECK
ADD  CONSTRAINT [FK_Person_Customers] FOREIGN KEY([CustomerID])
REFERENCES [dbo].[Customers] ([CustomerID])
GO
ALTER TABLE [dbo].[Person] CHECK CONSTRAINT [FK_Person_Customers]
GO
```

**14. Company**

- **CustomerID** (int) – identyfikator klienta, klucz główny do tabeli
- **CompanyName** (nvarchar(50)) – nazwa firmy
- **NIP** (nvarchar(50)) – numer NIP firmy
- **CompanyAdress** (nvarchar(50)) – adres siedziby firmy

  **Warunki:**

  - **NIP** składa się wyłącznie z cyfr

- **NIP** unikalny

```
CREATE TABLE [dbo].[Company](
     [CustomerID] [int] NOT NULL,
     [CompanyName] [nvarchar](50) NOT NULL,
     [NIP] [nvarchar](50) NOT NULL,
     [CompanyAdress] [nvarchar](50) NULL,
 CONSTRAINT [PK_Company] PRIMARY KEY CLUSTERED
(
     [CustomerID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY],
 CONSTRAINT [UC_COMPANY] UNIQUE NONCLUSTERED
(
     [NIP] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Company]  WITH CHECK ADD  CONSTRAINT
[FK_Company_Customers] FOREIGN KEY([CustomerID])
REFERENCES [dbo].[Customers] ([CustomerID])
GO
ALTER TABLE [dbo].[Company] CHECK CONSTRAINT [FK_Company_Customers]
GO
ALTER TABLE [dbo].[Company]  WITH CHECK
ADD  CONSTRAINT [CK_Company] CHECK  ((isnumeric([nip])=(1)))
GO
ALTER TABLE [dbo].[Company] CHECK CONSTRAINT [CK_Company]
GO
```

**15. WorksIn**

- **PersonID** (int) – klucz obcy do tabeli klientów indywidualnych
- **CompanyID** (int) – klucz obcy do tabeli klientów firmowych
- **StartDate** (date) – data rozpoczęcia pracy w firmie
- **EndDate** (date) – analogicznie, data zakończenia pracy w firmie

  **Warunki:**

  - Data **EndDate** późniejsza niż **StartDate**
  - Trójka **(PersonID, CompanyID, StartDate)** jest kluczem głównym
  - **EndDate** unikalna

```
CREATE TABLE [dbo].[WorksIn](
     [PersonID] [int] NOT NULL,
```

```
     [CompanyID] [int] NOT NULL,
     [StartDate] [date] NOT NULL,
     [EndDate] [date] NULL,
 CONSTRAINT [PK_WorksIn] PRIMARY KEY CLUSTERED
(
     [PersonID] ASC,
     [CompanyID] ASC,
     [StartDate] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[WorksIn]  WITH CHECK
ADD  CONSTRAINT [FK_WorksIn_Company] FOREIGN KEY([CompanyID])
REFERENCES [dbo].[Company] ([CustomerID])
GO
ALTER TABLE [dbo].[WorksIn] CHECK CONSTRAINT [FK_WorksIn_Company]
GO
ALTER TABLE [dbo].[WorksIn]  WITH CHECK
ADD  CONSTRAINT [FK_WorksIn_Person] FOREIGN KEY([PersonID])
REFERENCES [dbo].[Person] ([CustomerID])
GO
ALTER TABLE [dbo].[WorksIn] CHECK CONSTRAINT [FK_WorksIn_Person]
GO
ALTER TABLE [dbo].[WorksIn]  WITH CHECK
ADD  CONSTRAINT [CK_WorksIn] CHECK  (([startdate]<[enddate]))
GO
ALTER TABLE [dbo].[WorksIn] CHECK CONSTRAINT [CK_WorksIn]
GO
```

**16. CustomerDiscounts**

- **DiscountID** (int) – klucz obcy do tabeli rabatów
- **CustomerID** (int) – klucz obcy do tabeli klientów
- **ReceiveDate** (date) – data otrzymania rabatu
- **EndDate** (date) – data końca ważności rabatu, w przypadku wartości null rabat jest nieograniczony czasowo
- **Utilised** (bit) – informacja o tym, czy rabat jednorazowego użytku został już użyty, czy nie

  **Warunki:**

  - Data **EndDate** późniejsza niż **ReceiveDate**
  - **Utilised** domyślnie fałszem
  - Para (**DiscountID, CustomerID)** jest kluczem głównym
  - **ReceiveDate** unikalna

- trójka **(DiscountID, CustomerID, RecieveDate)** unikalna

```
CREATE TABLE [dbo].[CustomerDiscounts](
     [CustomerID] [int] NOT NULL,
     [DiscountID] [int] NOT NULL,
     [RecieveDate] [date] NOT NULL,
     [EndDate] [date] NULL,
     [Utilised] [bit] NOT NULL,
 CONSTRAINT [PK_CustomerDiscounts] PRIMARY KEY CLUSTERED
(
     [CustomerID] ASC,
     [DiscountID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY],
 CONSTRAINT [UC_CUSTOMERDISCOUNT] UNIQUE NONCLUSTERED
(
     [DiscountID] ASC,
     [CustomerID] ASC,
     [RecieveDate] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[CustomerDiscounts]  WITH CHECK
ADD  CONSTRAINT [FK_CustomerDiscounts_CompanyDiscount] FOREIGN
KEY([DiscountID])
REFERENCES [dbo].[CompanyDiscount] ([DiscountID])
GO
ALTER TABLE [dbo].[CustomerDiscounts] CHECK CONSTRAINT
[FK_CustomerDiscounts_CompanyDiscount]
GO
ALTER TABLE [dbo].[CustomerDiscounts]  WITH CHECK
ADD  CONSTRAINT [FK_CustomerDiscounts_Customers] FOREIGN KEY([CustomerID])
REFERENCES [dbo].[Customers] ([CustomerID])
GO
ALTER TABLE [dbo].[CustomerDiscounts] CHECK CONSTRAINT
[FK_CustomerDiscounts_Customers]
GO
ALTER TABLE [dbo].[CustomerDiscounts]  WITH CHECK
ADD  CONSTRAINT [FK_CustomerDiscounts_Discounts] FOREIGN KEY([DiscountID])
REFERENCES [dbo].[PersonDiscounts] ([DiscountID])
GO
ALTER TABLE [dbo].[CustomerDiscounts] CHECK CONSTRAINT
[FK_CustomerDiscounts_Discounts]
GO
ALTER TABLE [dbo].[CustomerDiscounts]  WITH CHECK
ADD  CONSTRAINT [CK_CustomerDiscounts] CHECK  (([recievedate]<[enddate]))
GO
```

```
ALTER TABLE [dbo].[CustomerDiscounts] CHECK CONSTRAINT
[CK_CustomerDiscounts]
GO
```

**17. PersonDiscounts**

- **DiscountID** (int) – identyfikator rabatu, klucz główny do tabeli, autoinkrementowany
- **MinValueTotal** (money) – minimalna łączna wartość zamówień klienta wymagana do otrzymania rabatu
- **ValidityTime** (int) – liczba dni, przez które rabat jest możliwy do wykorzystania, począwszy od daty przyznania (null oznacza nieokreślony termin ważności)
- **NumberOfOrders** (int) – minimalna liczba zamówień które klient musi złożyć, aby otrzymać rabat
- **MinValuePerOrder** (money) – minimalna wartość zamówienia liczącego się do „naliczania” zamówień potrzebnych do otrzymania rabatu

  **Warunki:**

  - **Percent** w przedziale od 0 do 1
  - **MinValueTotal, NumberOfOrders, MinValuePerOrder** większe bądź równe 0

```
CREATE TABLE [dbo].[PersonDiscounts](
     [DiscountID] [int] NOT NULL,
     [MinValueTotal] [money] NOT NULL,
     [ValidityTime] [int] NULL,
     [NumberOfOrders] [int] NOT NULL,
     [MinValuePerOrder] [money] NOT NULL,
 CONSTRAINT [PK_PersonDiscounts] PRIMARY KEY CLUSTERED
(
     [DiscountID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[PersonDiscounts]  WITH CHECK ADD  CONSTRAINT
[FK_PersonDiscounts_Discounts] FOREIGN KEY([DiscountID])
REFERENCES [dbo].[Discounts] ([DiscountID])
GO

ALTER TABLE [dbo].[PersonDiscounts] CHECK CONSTRAINT
[FK_PersonDiscounts_Discounts]
GO
```

```
ALTER TABLE [dbo].[PersonDiscounts]  WITH CHECK ADD  CONSTRAINT
[CK_PersonDiscounts_2] CHECK  (([validitytime]>=(0)))
GO

ALTER TABLE [dbo].[PersonDiscounts] CHECK CONSTRAINT
[CK_PersonDiscounts_2]
GO

ALTER TABLE [dbo].[PersonDiscounts]  WITH CHECK ADD  CONSTRAINT
[CK_PersonDiscounts_3] CHECK  (([minvaluetotal]>=(0)))
GO

ALTER TABLE [dbo].[PersonDiscounts] CHECK CONSTRAINT
[CK_PersonDiscounts_3]
GO

ALTER TABLE [dbo].[PersonDiscounts]  WITH CHECK ADD  CONSTRAINT
[CK_PersonDiscounts_4] CHECK  (([numberoforders]>=(0)))
GO

ALTER TABLE [dbo].[PersonDiscounts] CHECK CONSTRAINT
[CK_PersonDiscounts_4]
GO

ALTER TABLE [dbo].[PersonDiscounts]  WITH CHECK ADD  CONSTRAINT
[CK_PersonDiscounts_5] CHECK  (([minvalueperorder]>=(0)))
GO

ALTER TABLE [dbo].[PersonDiscounts] CHECK CONSTRAINT
[CK_PersonDiscounts_5]
GO
```

**18. CompanyDisounts**

- **DiscountID** (int) – identyfikator rabatu, klucz główny do tabeli, autoinkrementowany
- **ContinuityPeriod** (int) – maksymalna liczba miesięcy przestoju w zamówieniach, która nie powoduje przerwania posiadania rabatu
- **MaxDiscountCompound**(int) – maksymalna wielokrotność zniżki możliwej do otrzymania w związku z nakładaniem się rabatów
- **MinValueTotal** (money) – minimalna łączna wartość zamówień klienta wymagana do otrzymania rabatu

- **NumberOfOrders** (int) – minimalna liczba zamówień które klient musi złożyć, aby otrzymać rabat

  **Warunki:**

  - **MaxCompoundPercent** w przedziale od 0 do 1
  - **MinValueTotal, NumberOfOrders, MinValuePerOrder, MaxDiscountCompound, ContinuityPeriod** od 0

```
CREATE TABLE [dbo].[CompanyDiscounts](
     [DiscountID] [int] NOT NULL,
     [ContinuityPeriod] [int] NOT NULL,
     [MinValueTotal] [money] NOT NULL,
     [MaxDiscountCompound] [int] NOT NULL,
     [NumberOfOrders] [int] NOT NULL,
 CONSTRAINT [PK_CompanyDiscounts] PRIMARY KEY CLUSTERED
(
     [DiscountID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[CompanyDiscounts]  WITH CHECK ADD  CONSTRAINT
[FK_CompanyDiscounts_Discounts] FOREIGN KEY([DiscountID])
REFERENCES [dbo].[Discounts] ([DiscountID])
GO

ALTER TABLE [dbo].[CompanyDiscounts] CHECK CONSTRAINT
[FK_CompanyDiscounts_Discounts]
GO

ALTER TABLE [dbo].[CompanyDiscounts]  WITH CHECK ADD  CONSTRAINT
[CK_CompanyDiscount_1] CHECK  (([numberoforders]>(0)))
GO

ALTER TABLE [dbo].[CompanyDiscounts] CHECK CONSTRAINT
[CK_CompanyDiscount_1]
GO

ALTER TABLE [dbo].[CompanyDiscounts]  WITH CHECK ADD  CONSTRAINT
[CK_CompanyDiscount_2] CHECK  (([continuityperiod]>(0)))
GO

ALTER TABLE [dbo].[CompanyDiscounts] CHECK CONSTRAINT
[CK_CompanyDiscount_2]
GO

ALTER TABLE [dbo].[CompanyDiscounts]  WITH CHECK ADD  CONSTRAINT
[CK_CompanyDiscount_3] CHECK  (([minvaluetotal]>(0)))
```

```
GO

ALTER TABLE [dbo].[CompanyDiscounts] CHECK CONSTRAINT
[CK_CompanyDiscount_3]
GO

ALTER TABLE [dbo].[CompanyDiscounts]  WITH CHECK ADD  CONSTRAINT
[CK_CompanyDiscount_4] CHECK  (([maxdiscountcompound]>(0)))
GO

ALTER TABLE [dbo].[CompanyDiscounts] CHECK CONSTRAINT
[CK_CompanyDiscount_4]
GO
```

**19. Discounts**

- **DiscountID** (int) - identyfikator zniżki
- **Percent** (real) - wartość procentowa zniżki
- **StartDate** (date) - początek obowiązywania rabatu
- **EndDate** (date)- koniec rabatu

Warunki: **StartDate** mniejsze niż **EndDate**

```
CREATE TABLE [dbo].[Discounts](
     [DiscountID] [int] IDENTITY(1,1) NOT NULL,
     [Percent] [real] NOT NULL,
     [StartDate] [date] NOT NULL,
     [EndDate] [date] NOT NULL,
 CONSTRAINT [PK_Discounts_1] PRIMARY KEY CLUSTERED
(
     [DiscountID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Discounts]  WITH CHECK ADD  CONSTRAINT [CK_Discounts]
CHECK  (([startdate]<[enddate]))
GO

ALTER TABLE [dbo].[Discounts] CHECK CONSTRAINT [CK_Discounts]
GO

ALTER TABLE [dbo].[Discounts]  WITH CHECK ADD  CONSTRAINT
[CK_Discounts_1] CHECK  (((0)<[Percent] AND [Percent]<(1)))
```

```
GO

ALTER TABLE [dbo].[Discounts] CHECK CONSTRAINT [CK_Discounts_1]
GO
```

**20. Reservations**

- **ReservationID** (int) – identyfikator rezerwacji, klucz główny do tabeli, autoinkrementowany
- **ReservationDate** (datetime) – data rezerwacji
- **SeatsAmmount** (int) – liczba zarezerwowanych miejsc przy stoliku, w przypadku null rezerwowany jest cały stolik
- **AvailableTableID** (int) – klucz obcy do tabeli dostępnych stolików
- **OrderID** (int) – klucz obcy do tabeli zamówień

```
CREATE TABLE [dbo].[Reservations](
     [ReservationID] [int] IDENTITY(1,1) NOT NULL,
     [ReservationDate] [datetime] NOT NULL,
     [SeatsAmount] [int] NULL,
     [OrderID] [int] NOT NULL,
     [TableID] [int] NOT NULL,
 CONSTRAINT [PK_Reservations] PRIMARY KEY CLUSTERED
(
     [ReservationID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY
= OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Reservations]  WITH CHECK
ADD  CONSTRAINT [FK_Reservations_Orders] FOREIGN KEY([OrderID])
REFERENCES [dbo].[Orders] ([OrderID])
GO
ALTER TABLE [dbo].[Reservations] CHECK CONSTRAINT
[FK_Reservations_Orders]
GO
ALTER TABLE [dbo].[Reservations]  WITH CHECK
ADD  CONSTRAINT [FK_Reservations_Tables] FOREIGN KEY([TableID])
REFERENCES [dbo].[Tables] ([TableID])
GO
ALTER TABLE [dbo].[Reservations] CHECK CONSTRAINT
[FK_Reservations_Tables]
GO
```

### 21. ReservationIndividual

- **PersonID** (int) – klucz obcy do tabeli klientów indywidualnych
- **ReservationID** (int) klucz główny, identyfikator rezerwacji
- **Paid** (bit) – informacja o tym, czy zamówienie było opłacone

```
CREATE TABLE [dbo].[ReservationIndividual](
     [PersonID] [int] NOT NULL,
     [ReservationID] [int] NOT NULL,
     [Paid] [bit] NOT NULL,
 CONSTRAINT [PK_ReservationIndividual] PRIMARY KEY CLUSTERED
(
     [ReservationID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY
= OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[ReservationIndividual]  WITH CHECK
ADD  CONSTRAINT [FK_ReservationIndividual_Person] FOREIGN KEY([PersonID])
REFERENCES [dbo].[Person] ([CustomerID])
GO
ALTER TABLE [dbo].[ReservationIndividual] CHECK CONSTRAINT
[FK_ReservationIndividual_Person]
GO
ALTER TABLE [dbo].[ReservationIndividual]  WITH CHECK
ADD  CONSTRAINT [FK_ReservationIndividual_Reservations] FOREIGN
KEY([ReservationID])
REFERENCES [dbo].[Reservations] ([ReservationID])
GO
ALTER TABLE [dbo].[ReservationIndividual] CHECK CONSTRAINT
[FK_ReservationIndividual_Reservations]
GO
```

### 22. ReservationCompany

- **CompanyID** (int) – klucz obcy do tabeli klientów firmowych
- **ReservationID** (int) klucz główny, identyfikator rezerwacji

```
CREATE TABLE [dbo].[ReservationCompany](
     [ReservationID] [int] NOT NULL,
     [CompanyID] [int] NOT NULL,
 CONSTRAINT [PK_ReservationCompany_1] PRIMARY KEY CLUSTERED
(
     [ReservationID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY
= OFF) ON [PRIMARY]
) ON [PRIMARY]
```

```
GO
ALTER TABLE [dbo].[ReservationCompany]  WITH CHECK
ADD  CONSTRAINT [FK_ReservationCompany_Company] FOREIGN KEY([CompanyID])
REFERENCES [dbo].[Company] ([CustomerID])
GO
ALTER TABLE [dbo].[ReservationCompany] CHECK CONSTRAINT
[FK_ReservationCompany_Company]
GO
ALTER TABLE [dbo].[ReservationCompany]  WITH CHECK
ADD  CONSTRAINT [FK_ReservationCompany_Reservations] FOREIGN
KEY([ReservationID])
REFERENCES [dbo].[Reservations] ([ReservationID])
GO
ALTER TABLE [dbo].[ReservationCompany] CHECK CONSTRAINT
[FK_ReservationCompany_Reservations]
GO
```

**23. ReservationCompanyWorkers**

- **ReservationID** (int) – klucz obcy do tabeli rezerwacji firmowych
- **WorkerID** (int) – klucz obcy do tabeli klientów indywidualnych (pracowników firmy)

**Warunki:**

- Para **(ReservationID, WorkerID)** unikalna (klucz)

```
CREATE TABLE [dbo].[ReservationCompanyWorkers](
     [ReservationID] [int] NOT NULL,
     [WorkerID] [int] NOT NULL,
 CONSTRAINT [PK_ReservationCompanyWorkers] PRIMARY KEY CLUSTERED
(
     [ReservationID] ASC,
     [WorkerID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY
= OFF) ON [PRIMARY],
 CONSTRAINT [UC_RESERVATIONCOMPANYWORKERS] UNIQUE NONCLUSTERED
(
     [ReservationID] ASC,
     [WorkerID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY
= OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[ReservationCompanyWorkers]  WITH CHECK
ADD  CONSTRAINT [FK_ReservationCompanyWorkers_Person] FOREIGN
KEY([WorkerID])
REFERENCES [dbo].[Person] ([CustomerID])
```

```
GO
ALTER TABLE [dbo].[ReservationCompanyWorkers] CHECK CONSTRAINT
[FK_ReservationCompanyWorkers_Person]
GO
ALTER TABLE [dbo].[ReservationCompanyWorkers]  WITH CHECK
ADD  CONSTRAINT [FK_ReservationCompanyWorkers_ReservationCompany1]
FOREIGN KEY([ReservationID])
REFERENCES [dbo].[ReservationCompany] ([ReservationID])
GO
ALTER TABLE [dbo].[ReservationCompanyWorkers] CHECK CONSTRAINT
[FK_ReservationCompanyWorkers_ReservationCompany1]
GO
```

**24. Restaurants**

- **RestaurantID** (int) identyfikator restauracji
- **RestaurantName** (int) nazwa restauracji

Warunki:

- **RestaurantName** unikalne

```
CREATE TABLE [dbo].[Restaurants](
     [RestaurantID] [int] IDENTITY(1,1) NOT NULL,
     [RestaurantName] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_Restaurants] PRIMARY KEY CLUSTERED
(
     [RestaurantID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY
= OFF) ON [PRIMARY],
 CONSTRAINT [UC_DISHES] UNIQUE NONCLUSTERED
(
     [RestaurantName] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY
= OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

## *IV. Widoki*

*Aktualne pozycje w menu*

```
CREATE   VIEW [dbo].[Current Menu] AS
     SELECT MenuPositionID, Dishes.DishID,
          DishName, StartDate, EndDate, UnitPrice
     FROM Menu
     INNER JOIN Dishes
     ON menu.DishID= Dishes.DishID
     WHERE StartDate <= GETDATE() AND GETDATE() <= EndDate
GO
```

*Aktualne rezerwacje indywidualne*

```
CREATE OR ALTER     VIEW [dbo].[Current Reservations Individual]
AS
     SELECT Reservations.ReservationID, OrderDate, ReservationDate,
TableID,
               SeatsAmount, dbo.ReservationIndividual.PersonID AS ID,
Firstname+' '+Lastname AS Name
     FROM dbo.Reservations
     INNER JOIN dbo.Orders
          ON Orders.OrderID = Reservations.OrderID
     INNER JOIN dbo.ReservationIndividual
          ON ReservationIndividual.ReservationID =
Reservations.ReservationID
     INNER JOIN dbo.Person
          ON Person.CustomerID = ReservationIndividual.PersonID
     WHERE GETDATE() <= ReservationDate
GO
```

*Aktualne rezerwacje firmowe*

```
CREATE VIEW [dbo].[Current Reservations Company]
AS
     SELECT Reservations.ReservationID, OrderDate,
          ReservationDate, TableID,
               SeatsAmount, dbo.Reservationcompany.CompanyID AS ID,
                   companyname AS [Company Name],
               firstname+' '+lastname AS worker_name
     FROM dbo.Reservations
     INNER JOIN dbo.Orders
          ON Orders.OrderID = Reservations.OrderID

     INNER JOIN dbo.ReservationCompany
          ON ReservationCompany.ReservationID =
                 Reservations.ReservationID
     INNER JOIN dbo.Company
          ON Company.CustomerID = Reservationcompany.companyID
```

```
     LEFT JOIN dbo.ReservationCompanyWorkers
          ON ReservationCompanyWorkers.ReservationID =
                  ReservationCompany.ReservationID
     LEFT JOIN dbo.Person
          ON Person.CustomerID = ReservationCompanyWorkers.WorkerID
     WHERE GETDATE() <= ReservationDate
GO
```

*Aktualnie dostępne stoliki*

```
CREATE VIEW [dbo].[Current Available Tables] AS
     SELECT t.tableid AS TableID, c.AvailableSeats
     FROM dbo.Tables AS t
     INNER JOIN dbo.CovidTableConstraints AS c
         ON C.TableID = T.TableID
     WHERE GETDATE() NOT BETWEEN c.StartDate AND c.EndDate
GO
```

*Aktualne rabaty dostępne dla klientów firmowych*

```
CREATE VIEW [dbo].[Current Company Discounts] AS
     SELECT *
     FROM dbo.CompanyDiscount
     WHERE StartDate <= GETDATE() and GETDATE() <= EndDate
GO
```

*Aktualne rabaty dostępne dla klientów indywidualnych*

```
CREATE VIEW [dbo].[Current Person Discounts] AS
     SELECT *
     FROM dbo.PersonDiscounts
     WHERE StartDate <= GETDATE() and GETDATE() <= EndDate
GO
```

*Raporty:*

1. *Raport dot. liczby zamówień dla wszystkich klientów*

```
CREATE VIEW [dbo].[Customers Orders Ammount] AS
     select p.firstname + p.lastname as name,
                    COUNT (*) as [Number of orders]
     from Person as p
```

```
     left join Customers as c
          on p.CustomerID = c.CustomerID
     left join Orders as o
          on c.CustomerID = o.CustomerID
     group by o.CustomerID, p.Firstname, p.Lastname
     union
     select com.companyname as name,
                    COUNT (*) as [Number of orders]
     from company as com
     left join Customers as c
          on com.CustomerID = c.CustomerID
     left join Orders as o
          on c.CustomerID = o.CustomerID
     group by o.CustomerID, com.CompanyName
GO
```

*2. Raport dot. kwot i dat zamówień dla wszystkich klientów*

```
CREATE   VIEW [dbo].[Order Statistics] AS
     SELECT o.CustomerID, o.OrderID, o.OrderDate,
SUM(d.UnitPrice*od.Amount) AS [value]
     FROM Orders AS o
     INNER JOIN OrderDetails AS od
          ON o.OrderID = od.OrderID
     INNER JOIN Menu AS m
          ON m.MenuPositionID = od.MenuPositionID
     INNER JOIN Dishes AS d
          ON d.DishID = m.DishID
     GROUP BY o.OrderID, o.CustomerID, o.Orderdate
GO
```

3. Raporty tygodniowe
    - dot. rezerwacji stolików

```
CREATE VIEW [dbo].[Table reservations last week] AS
     SELECT * FROM Reservations
     WHERE DATEDIFF(DAY, ReservationDate, GETDATE()) <= 7
GO
```

- dot. rabatów przyznanych klientom

```
CREATE VIEW [dbo].[Given Discounts last week] AS
     select * from CustomerDiscounts
     where DATEDIFF(DAY, RecieveDate, GETDATE()) <= 7
GO
```

- dot. pozycji pojawiających się w menu

```
CREATE VIEW [dbo].[Menu Positions last week] AS
```

```
     select * from Menu
     where DATEDIFF(DAY, StartDate, GETDATE()) <= 7
GO
```

4. Raporty miesięczne
    - dot. rezerwacji stolików

```
CREATE VIEW [dbo].[Table reservations last months] AS
     SELECT * FROM Reservations
     WHERE DATEDIFF(MONTH, ReservationDate, GETDATE()) <= 1
GO
```

    - dot. rabatów przyznanych klientom

```
CREATE VIEW [dbo].[Given Discounts last month] AS
     select * from CustomerDiscounts
     where DATEDIFF(MONTH, RecieveDate, GETDATE()) <= 1
GO
```

    - dot. pozycji pojawiających się w menu

```
CREATE VIEW [dbo].[Menu Positions last month] AS
     select * from Menu
     where DATEDIFF(MONTH, StartDate, GETDATE()) <= 1
GO
```

---

## *V. Typy*

- *na potrzeby implementacji kilku procedur tworzyliśmy table-value parametry*

*Typ reprezentujący dania wchodzące w skład zamówienia*

```
CREATE TYPE [dbo].[OrderPositionsType] AS TABLE(
     [MenuPosition] [INT] NULL,
     [ammount] [INT] NULL
)
GO
```

*Typ reprezentujący półprodukty wchodzące w skład dania*

```
CREATE TYPE [dbo].[DishDetailsType] AS TABLE(
```

```
     [HalfProductID] [INT] NOT NULL,
     [quantity] [FLOAT] NOT NULL
)
GO
```

*Typ reprezentujący pracowników których imiona są na rezerwacji firmowej*

```
CREATE TYPE [dbo].[ReservationCompanyWorkers] AS TABLE(
     [ReservationID] [int] NOT NULL,
     [PersonID] [float] NOT NULL
)
GO
```

---

## *VI. Procedury*

*GENEROWANIE FAKTURY*

```
CREATE OR ALTER PROCEDURE [dbo].[generateInvoice]
     (
     @CustomerID int,
     @OrderID int,
     @DiscountID real
     )
AS
BEGIN

     declare @orderValue money = (
          Select sum(unitprice*amount) as 'Price'from OrderDetails
          inner join menu
          on OrderDetails.MenuPositionID = Menu.MenuPositionID
          inner join Dishes
          on Dishes.DishID = Menu.DishID
          where orderDetails.OrderID= @OrderID
          )

     if @CustomerID in (select CustomerID from Person)
     begin
          Select Firstname as 'First name',  LastName as 'Last name' from
Person
          where CustomerID = @CustomerID
     END
     ELSE
     BEGIN
          Select CompanyName as 'Company Name' from Company
          where CustomerID= @CustomerID
     END
          Select OrderDate as 'Order Date', PreferedRealisationDate as
'Preffered Date' from Orders
          where OrderID=@OrderID

          Select dishes.Dishname as ' dish name', OrderDetails.Amount as
'amount',OrderDetails.Amount*UnitPrice as 'Price'from OrderDetails
          inner join menu
          on OrderDetails.MenuPositionID = Menu.MenuPositionID
          inner join Dishes
          on Dishes.DishID = Menu.DishID
          where orderDetails.OrderID= @OrderID

          select @orderValue as 'order value without discount'

          --exec getDiscountForOrder @customerID, @OrderID, @DiscountID
          if (select  (1-dbo.getDiscountPercent( @CustomerID, @OrderID,
@DiscountID))*@orderValue as 'order value with discount') is not null
          begin
          select  (1-dbo.getDiscountPercent( @CustomerID, @OrderID,
@DiscountID))*@orderValue as 'order value with discount'
          end

END
```

*Dodawanie klienta indywidualnego*

```
CREATE PROCEDURE [dbo].[add_individual_customer]
(@firstname nvarchar(50), @middlename nvarchar(50), @lastname
nvarchar(50),
 @phone nchar(10), @adress nvarchar(50), @email nvarchar(50),
@restaurantid int,
 @companyid int, @startdate date, @enddate date)
as
     begin
          insert into customers
               values (@phone, @adress, @email, @restaurantid)
          insert into Person
               values (@@IDENTITY, @firstname,
                         @middlename, @lastname)
          if @companyid is not null AND @startdate is not null
              AND @enddate is not null
          begin
               insert into WorksIn
                    values (@@IDENTITY, @companyid,
                              @startdate, @enddate);
          end
     end
GO
```

*Dodawanie klienta firmowego*

```
CREATE PROCEDURE [dbo].[add_company_customer]
(@companyname nvarchar(50), @nip nvarchar(50), @companyadress
nvarchar(50),
 @phone nchar(10), @adress nvarchar(50), @email nvarchar(50),
@restaurantid int)
as
     begin
          insert into customers
               values (@phone, @adress, @email, @restaurantid)
          insert into Company
               values (@@IDENTITY, @companyname,
                         @nip, @companyadress)
     end
GO
```

*Dodawanie zamówienia*

```
CREATE   PROCEDURE [dbo].[add_order]
   (@orderpositions OrderPositionsType READONLY,
```

```
     @customerid INT, @orderdate DATE, @preferedrealdate DATE,
     @orderadress NVARCHAR(50), @employeeid INT)
AS
     BEGIN
          INSERT INTO Orders
               VALUES (@customerid, @orderdate, @preferedrealdate,
                         @orderadress, @employeeid)

          INSERT INTO OrderDetails
               SELECT @@IDENTITY, * FROM @orderpositions
     END
GO
```

*Dodawanie pozycji pracowniczych*

```
CREATE PROCEDURE [dbo].[add_position]
     (@positionname NVARCHAR(50))
AS
     BEGIN
          INSERT INTO Positions
               VALUES (@positionname)
     END
GO
```

*Dodawanie dania*

```
CREATE   PROCEDURE [dbo].[add_dish]
(
     @dishdetails DishDetailsType READONLY,
     @dishname nvarchar(50), @categoryid int, @unitprice money
)
as
     begin
          insert into dishes
               values (@dishname, @categoryid, @unitprice)

          insert into DishDetails
               select HalfProductID, @@IDENTITY,
                         quantity from @dishdetails
     end
GO
```

*Dodawanie pozycji do menu*

```
CREATE   PROCEDURE [dbo].[add_position_to_menu]
     @DishID INT,
     @StartDate DATE,
     @EndDate DATE,
     @Restaurant INT = 1
AS
```

```
     BEGIN
          SET NOCOUNT ON;

          IF DATEADD(DAY, 1, GETDATE()) <= @StartDate
          BEGIN
             RAISERROR ('Cannot insert into menu, there has to be at least
                              one day before inserting', 16, 1)
          end

          INSERT INTO dbo.Menu(DishID, StartDate,
              EndDate, RestaurantID)
               VALUES
               (   @DishID,        -- DishID - int
                    @StartDate,          -- StartDate - date
                    @EndDate,      -- EndDate - date
                    @Restaurant     -- RestaurantID - int
               )
     END
GO
```

*Dodawanie dostępnych rabatów indywidualnych do Tabeli CustomerDiscounts*

```
CREATE PROCEDURE [dbo].[updatePersonDiscounts] (@customerID INT)
AS
BEGIN

     -- sprawdz czy klient jest indywidualny
     IF EXISTS (SELECT * FROM dbo.Company WHERE CustomerID = @customerID)
     BEGIN
          RAISERROR ('Cannot use this procedure with a company customer',
16, 1)
          RETURN
     END

     -- procedura sprawdzajaca czy klient ma nowe rabaty, po dodaniu do
listy
     --definiowanie tablicy obecnych rabatow
     Declare @actualDiscountsTable table(
     DiscountID int, [Percent] real, StartDate date, EndDate date,
     MinValueTotal money, ValidityTime int, NumberOfOrders int,
MinValuePerOrder money)

     -- wstawianie wartosci
     insert into @actualDiscountsTable (DiscountID,  [Percent], StartDate,
EndDate,
     MinValueTOtal , ValidityTime, NumberOfOrders, MinValuePerOrder)
     select discounts.DiscountID,  [Percent], StartDate, EndDate ,
                    MinValueTotal , ValidityTime, NumberOfOrders,
MinValuePerOrder
     FROM PersonDiscounts
     INNER JOIN discounts ON discounts.discountid =
```

```
persondiscounts.discountid
     where StartDate<GETDATE() and EndDate>GETDATE()

     Declare @DiscountID int, @Percent real, @StartDate date, @EndDate
date,
     @MinValueTOtal money, @ValidityTime int, @NumberOfOrders int,
     @MinValuePerOrder money

     -- kursor
     Declare curs cursor for (select * from @actualDiscountsTable)
     open curs
     fetch next from curs into  @DiscountID, @Percent, @StartDate,
@EndDate,
     @MinValueTotal, @ValidityTime, @NumberOfOrders,
     @MinValuePerOrder


     WHILE @@FETCH_STATUS = 0
     BEGIN
          -- tabela z ordersami dla danego klienta + suma odpowiedniego
zamówienia

          DECLARE @ordersTabWithPrices TABLE (OrderID INT, OrderPrice
FLOAT)
          INSERT INTO @ordersTabWithPrices (OrderID, OrderPrice)
          SELECT Orders.OrderID, SUM(Amount*UnitPrice) AS OrderPrice FROM
Orders
          INNER JOIN OrderDetails
          ON OrderDetails.OrderID = Orders.OrderID
          INNER JOIN Menu
          ON Menu.MenuPositionID = OrderDetails.MenuPositionID
          INNER JOIN Dishes
          ON Dishes.DishID = Menu.MenuPositionID
          WHERE (Orders.CustomerID = @customerID)
          AND (Orders.OrderDate > @StartDate AND Orders.OrderDate<
@EndDate)
          GROUP BY Orders.OrderID

          IF (@NumberOfOrders = 0)
          BEGIN
                    -- 1 typ rabatu- rabat za ilosc zamowien
                    DECLARE @okOrdersCounter INT = (
                    SELECT COUNT(*) FROM @ordersTabWithPrices
                    WHERE (OrderPrice>@MinValuePerOrder))

                    -- czy ilosc zamowien ok, czy juz nie byl dodany
                    IF ((@okOrdersCounter >= @NumberOfOrders) AND
@DiscountID NOT IN (SELECT DiscountID FROM CustomerDiscounts WHERE
CustomerID = @customerID))
                    BEGIN
                         INSERT INTO CustomerDiscounts VALUES(
                         @customerID, @DiscountID, GETDATE(), @EndDate, 0)
```

```
                    END
          END
          ELSE
               BEGIN
                    DECLARE @summaryOrderValues FLOAT = (
                    SELECT SUM(OrderPrice) FROM @ordersTabWithPrices
                    )
                    -- czy kwota ok, czy juz nie byl dodany
                    IF ((@summaryOrderValues>=@MinValueTOtal) AND
@DiscountID NOT IN (SELECT DiscountID FROM CustomerDiscounts WHERE
CustomerID = @customerID))
                    BEGIN
                         INSERT INTO CustomerDiscounts VALUES(
                         @customerID, @DiscountID, GETDATE(), DATEADD(DAY,
@ValidityTime, GETDATE()), 0
                         )
                    END
               END
          FETCH NEXT FROM curs INTO  @DiscountID, @Percent, @StartDate,
@EndDate,
          @MinValueTOtal, @ValidityTime, @NumberOfOrders,
          @MinValuePerOrder
     END
END
GO
```

*Dodawanie dostępnych rabatów firmowych do Tabeli CustomerDIscounts*

```
CREATE OR ALTER       PROCEDURE [dbo].[updateCompanyDiscounts]
(
@customerID INT
)

AS
BEGIN
-- procedura sprawdzajaca czy klient ma nowe rabaty, po dodaniu do listy
     --definiowanie tablicy obecnych rabatow
     Declare @actualDiscountsTable table(
     DiscountID int, [Percent] real, NumberOfOrders int, ContinuityPeriod
int, MinValueTotal money,
     MaxDiscountCompound int, StartDate date, EndDate date)

     -- wstawianie wartosci
     insert into @actualDiscountsTable (DiscountID,  [Percent],
NumberOfOrders, ContinuityPeriod,
     MinValueTotal, MaxDiscountCompound, StartDate, EndDate)
```

```
     select discounts.DiscountID,  [Percent], NumberOfOrders,
ContinuityPeriod,
                    MinValueTotal, MaxDiscountCompound, StartDate, EndDate
     FROM CompanyDiscounts
     INNER JOIN discounts ON discounts.discountid =
companydiscounts.discountid
     where StartDate<GETDATE() and EndDate>GETDATE()


     Declare @DiscountID int,  @Percent real, @NumberOfOrders int,
@ContinuityPeriod int,
     @MinValueTotal int, @MaxDiscountCompound int, @StartDate date,
@EndDate date

     -- kursor
     Declare curs cursor for (select * from @actualDiscountsTable)
     open curs
     fetch next from curs into  @DiscountID, @Percent, @NumberOfOrders,
@ContinuityPeriod,
     @MinValueTotal, @MaxDiscountCompound, @StartDate, @EndDate


     WHILE @@FETCH_STATUS = 0
     BEGIN


          -- tabela z ordersami dla danego klienta + suma odpowiednich
zamówień

          Declare @ordersTabWithPrices table (monthID int, OrderPrice
float)
          INSERT INTO @ordersTabWithPrices (monthID, OrderPrice)
          SELECT DATEPART(MONTH, Orders.OrderDate), SUM(Amount*UnitPrice)
AS OrderPrice FROM Orders
          INNER JOIN OrderDetails
          ON OrderDetails.OrderID = Orders.OrderID
          INNER JOIN Menu
          ON Menu.MenuPositionID = OrderDetails.MenuPositionID
          INNER JOIN Dishes
          ON Dishes.DishID = Menu.DishID
          WHERE (Orders.CustomerID =@customerID) AND OrderDate =
CONVERT(DATE, GETDATE())
          AND (Orders.OrderDate > @StartDate AND Orders.OrderDate<
@EndDate)
          GROUP BY DATEPART(MONTH,  Orders.OrderDate)

          --- sprawdz ostatni miesiac jesli nie ma
          DECLARE @monthIndex DATE =  @StartDate


          DECLARE @discountCounter INT = 0
```

```
          WHILE  @monthIndex <  GETDATE()
          BEGIN



               IF (((SELECT SUM(OrderPrice) FROM @ordersTabWithPrices WHERE
monthID = DATEPART(MONTH,@monthIndex))> @MinValueTotal)
               AND (SELECT COUNT(OrderID) FROM Orders WHERE OrderDate
BETWEEN DATEADD(MONTH, -1, @monthIndex) AND DATEADD(MONTH, 1,
@monthIndex))>@NumberOfOrders)
               BEGIN
                    SET @discountCounter = @discountCounter + 1
               END
               ELSE
               BEGIN
                    SET @discountCounter = 0
               END
               SET @monthIndex = DATEADD(MONTH, 1, @monthIndex)
          END

          IF (@discountCounter >= @ContinuityPeriod)
          BEGIN
               IF (SELECT COUNT(*) FROM CustomerDiscounts
               WHERE CustomerID= @customerID AND DiscountID =  @DiscountID)
= 0
                    BEGIN
                         INSERT INTO CustomerDiscounts VALUES(@customerID,
@discountID, GETDATE(), DATEADD(MONTH, 1, GETDATE()), 0)
                    END
          END
          -- sprawdzilismy obecna ciaglosc rabatowa, teraz trzeba sprawdzić
ile rabatu sie nalezy

          FETCH NEXT FROM curs INTO  @DiscountID, @Percent,
@NumberOfOrders, @ContinuityPeriod,
          @MinValueTotal, @MaxDiscountCompound, @StartDate, @EndDate
     END

END
GO
```

*Sprawdzanie dostępnych dla danego klienta rabatów*

```
CREATE PROCEDURE [dbo].[getPersonAvailableDiscounts]
(
@customerID int
)
AS
BEGIN
```

```
     select * from CustomerDiscounts
     where Utilised=0 and CustomerID=@customerID
END
GO
```

*Wyświetlanie wartości zamówień z określonego przedziału czasowego dla konkretnego klienta*

```
CREATE     PROCEDURE [dbo].[ordersValuesFromTimeInterval]
(
@customerID int,
@StartDate date,
@EndDate date
)
AS
BEGIN
     select Orders.OrderID, sum(Amount*UnitPrice) as OrderPrice from Orders
     inner join OrderDetails
     on OrderDetails.OrderID = Orders.OrderID
     inner join Menu
     on Menu.MenuPositionID = OrderDetails.MenuPositionID
     inner join Dishes
     on Dishes.DishID = Menu.DishID
     where (Orders.CustomerID = @customerID)
     and (Orders.OrderDate > @StartDate and Orders.OrderDate< @EndDate)
     group by Orders.OrderID

END
GO
```

---

## *VII. Triggery*

### *Dla tabeli menu*

- Sprawdzanie czy pozycja była w menu w ciągu ostatniego miesiąca lub jest obecnie

```
CREATE             TRIGGER [dbo].[onMenuAdd]
ON [dbo].[Menu]
```

```
AFTER INSERT
AS
BEGIN
          DECLARE @newproductID INT = (SELECT DishID FROM inserted)
          DECLARE @newstartdate DATE = (SELECT Startdate FROM Inserted)
          DECLARE @newenddate DATE = (SELECT enddate FROM Inserted)

          -- sprawdz czy ta pozycja nie wystepuje juz w menu w danym
przedziale czasowym
          IF (SELECT COUNT(*) FROM menu
               WHERE DishID = @newproductID AND
               (startdate BETWEEN @newstartdate AND @newenddate
                    OR EndDate BETWEEN @newstartdate AND @newenddate)) > 1
               BEGIN
                    RAISERROR ( 'Product is/was/will be already in menu in
given period.', 16, 1)
                    ROLLBACK TRANSACTION
                    RETURN
               END


          DECLARE @count INT = (
          SELECT COUNT(*) FROM Menu
          WHERE (endDate BETWEEN (DATEADD(MONTH, -1,@newstartdate)) AND
@newstartdate)
          AND menu.dishID = @newproductID
          )

          IF ISNULL(@count, 0) > 0
               BEGIN
                    RAISERROR ('Product was in menu during last month',16,1)
                    ROLLBACK TRANSACTION
                    RETURN
               END

END
GO
```

### *Dla tabeli CovidTableConstraints*

- *Sprawdzanie, czy restrykcja nałożona na stolik jest możliwa do spełnienia, czyli liczba miejsc przy stoliku większa niż te w restrykcji*

```
CREATE   TRIGGER [dbo].[tableSeatsMaxAmount]
   ON  [dbo].[CovidTableConstraints]
   AFTER INSERT
AS
```

```
BEGIN
     DECLARE @TableID INT = (SELECT TableID FROM inserted)
     DECLARE @AvailableSeats INT = (SELECT AvailableSeats FROM inserted)

     DECLARE @maxSeats INT = (
          SELECT Seats FROM Tables
          WHERE @TableID=TableID
     )

     IF (@AvailableSeats > @maxSeats)
     BEGIN
          RAISERROR ('This table does not have so many seats', 16, 1)
          ROLLBACK TRANSACTION
          RETURN
     END

END
GO
```

### *Dla tabeli OrderDetails*

- Sprawdzanie czy pozycja jest możliwa do dodania, czyli czy wystarcza półproduktów + aktualizacja tabeli HalfProducts

```
CREATE           TRIGGER [dbo].[onAddOrderDetails]
   ON  [dbo].[OrderDetails]
   AFTER INSERT
AS
BEGIN
     --- sprawdz czy danie jest w menu w preferowanej dacie zamowienia
     DECLARE @orderID INT = (SELECT orderID FROM Inserted)
     DECLARE @prefRealDate date = (SELECT PreferedRealisationDate
                                        FROM dbo.Orders
                                        WHERE @orderID = OrderID)
     Declare @menuID int = (select MenuPositionID from inserted)
     if NOT EXISTS (select MenuPositionID
                         FROM menu
                         WHERE dbo.Menu.MenuPositionID = @menuID
                              AND (@prefRealDate BETWEEN StartDate AND
EndDate))
     begin
          RAISERROR ('Menu position is not or will not be in menu in given
date', 16, 1)
```

```
          DELETE FROM dbo.Orders WHERE orderid = @orderID

          ROLLBACK TRANSACTION
          RETURN
     END

     -- sprawdza czy mozna dodac danie do zamowienia -> update w
halfproducts

     -- ID zamowienia, od ktorego dodajemy potrawe ###
     Declare  @newOrderID int = (select OrderID from inserted)

     -- ID dodawanej potrawy ###
     Declare @newDishID int = (
     select DishID from Menu
     inner join inserted
     on Menu.MenuPositionID=inserted.MenuPositionID
     )

     -- Ilosc dodawanych porcji ####
     Declare @amountInOrder int = (select Amount from inserted)

     -- tabela (polprodukty, ilosc)
     Declare @halfproductTable TABLE ( HalfProductID int, Quantity float)
     insert into @halfproductTable (HalfProductID, Quantity)
     select halfProdID, Quantity from OrderDetails
     inner join Menu
     on Menu.MenuPositionID= OrderDetails.MenuPositionID
     inner join Dishes
     on Dishes.DishID = Menu.DishID
     inner join DishDetails
     on DishDetails.DishID = Menu.DishID
     where [OrderDetails].OrderID=@newOrderID and
(OrderDetails.MenuPositionID = (select MenuPositionID from inserted))

     DECLARE @HalfProductID INT, @Quantity FLOAT
     DECLARE curs CURSOR FOR (SELECT HalfProductID, Quantity FROM
@halfproductTable)
     OPEN curs
     FETCH NEXT FROM curs INTO @HalfProductID, @Quantity

     WHILE @@FETCH_STATUS = 0
     BEGIN
          -- ilosc polproduktu na danie
          DECLARE @halfProductAmount FLOAT = (
          SELECT Quantity FROM DishDetails
          WHERE DishDetails.DishID = @newDishID AND DishDetails.HalfProdID=
@HalfProductID
          )

          -- sprawdzamy czy jest wystarczajaco duzo polproduktow
          IF ((SELECT UnitsInStock FROM HalfProducts
```

```
          WHERE
HalfProductID=@HalfProductID)-(@halfProductAmount*@amountInOrder))<0
               BEGIN
               RAISERROR ('Not enough halfproducts', 16, 1)
               ROLLBACK TRANSACTION
               RETURN
               END
          FETCH NEXT FROM curs INTO @HalfProductID, @Quantity
     END

     DECLARE @HalfProductID2 INT, @Quantity2 FLOAT
     DECLARE curs2 CURSOR FOR (SELECT HalfProductID, Quantity FROM
@halfproductTable)
     OPEN curs2
     FETCH NEXT FROM curs2 INTO @HalfProductID2, @Quantity2

     WHILE @@FETCH_STATUS = 0
     BEGIN
          -- ilosc polproduktu na danie
          DECLARE @halfProductAmount2 FLOAT = (
          SELECT Quantity FROM DishDetails
          WHERE DishDetails.DishID = @newDishID AND DishDetails.HalfProdID=
@HalfProductID2
          )

          -- update w tabeli skladnikow
          UPDATE HalfProducts
          SET UnitsInStock = (UnitsInStock -
(@halfProductAmount2*@amountInOrder))
          WHERE HalfProductID=@HalfProductID2

          FETCH NEXT FROM curs2 INTO @HalfProductID2, @Quantity2
     END
END
GO
```

- Sprawdzanie czy owoce morza zostały zamówione odpowiednio wcześniej

```
CREATE   TRIGGER [dbo].[onAddSeaFood]
   ON  [dbo].[OrderDetails]
   AFTER INSERT
AS
BEGIN

     declare @insertedDishCategory int = (
     select Categories.CategoryID from Categories
     inner join Dishes
     on categories.CategoryID = dishes.CategoryID
     inner join menu
     on Menu.DishID = Dishes.DishID
```

```
where Menu.MenuPositionID = (select MenuPositionID from inserted)

)


if @insertedDishCategory = 3
begin
     declare @OrderID int = (select OrderID from inserted)

     declare @realisationWeekDay int  = (
     select DATEPART(weekday, PreferedRealisationDate) from Orders
     where OrderId = @OrderID
     )

     declare @realisationDate date =(
     select PreferedRealisationDate from Orders
     where OrderId = @OrderID
     )

     declare @orderDate date =(
     select OrderDate from Orders
     where OrderId = @OrderID
     )


     if(@realisationWeekDay<5 or @realisationWeekDay>7)
     begin
          RAISERROR ('Cannot eat Seafood in this day', 16, 1)
          ROLLBACK TRANSACTION
          RETURN
     end

     if (@realisationWeekDay = 5)
     begin
          if (@orderDate > DATEADD(day, -3 ,@realisationDate))
          begin
               RAISERROR ('Too late for seafood 5', 16, 1)
               ROLLBACK TRANSACTION
               RETURN
          end
     end

     if (@realisationWeekDay = 6)
     begin
          if @orderDate > DATEADD(day, -4 ,@realisationDate)
          begin
               RAISERROR ('Too late for seafood 6', 16, 1)
               ROLLBACK TRANSACTION
               RETURN
          end
     end
```

```
          if (@realisationWeekDay = 7)
          begin
               if @orderDate > DATEADD(day, -5 ,@realisationDate)
               begin
                    RAISERROR ('Too late for seafood 7', 16, 1)
                    ROLLBACK TRANSACTION
                    RETURN
               end
          end
     end
END
GO
```

### *Dla tabeli ReservationIndividual*

- Sprawdza czy klient złożył zamówienia: 5 za minimum 50 lub jedno za 200

```
CREATE OR ALTER     TRIGGER [dbo].[onAddIndividualReservation]
   ON [dbo].[ReservationIndividual]
   AFTER INSERT
AS
BEGIN

     DECLARE @addPersonID INT = (SELECT PersonID FROM inserted)

     DECLARE @addOrderID INT =
     (SELECT Orders.OrderID FROM Orders
     INNER JOIN Reservations
     ON Orders.OrderID = Reservations.OrderID
     INNER JOIN ReservationIndividual
     ON ReservationIndividual.ReservationID = Reservations.ReservationID
     WHERE ReservationIndividual.PersonID = @addPersonID
     and Reservations.ReservationID = (select ReservationID from inserted)
     )

     DECLARE @orderPrice INT = (
     SELECT SUM(OrderDetails.Amount*UnitPrice) FROM OrderDetails
     INNER JOIN Orders
     ON Orders.OrderID= OrderDetails.OrderID
     INNER JOIN Menu
     ON menu.MenuPositionID = OrderDetails.Amount
     INNER JOIN Dishes
     ON dishes.DishID = menu.DishID
     WHERE Orders.CustomerID = @addPersonID
     )

     DECLARE @ordersAmount INT = (
     SELECT COUNT(*) FROM Orders
     WHERE CustomerID = @addPersonID
     )
```

```
     IF NOT (@orderPrice >= 200 OR (@orderPrice >= 50 AND @ordersAmount >=
5))
     BEGIN
          RAISERROR ('Person does not have permisssion to take a reservation.
                              Not big enough total order value or number of
orders', 16, 1)
          ROLLBACK TRANSACTION
     end
END
GO
```

### *Dla tabeli Reservations*

- Sprawdza czy termin rezerwacji jest wolny, czy jest odpowiednia liczba miejsc przy stoliku

```
CREATE   TRIGGER [dbo].[onReservationAdd]
   ON  [dbo].[Reservations]
   after insert
AS
BEGIN

     declare @tableID int =( select tableID from inserted )
     declare @reservationDate datetime = ( select reservationDate from
inserted )
     declare @seatsAmount int = ( select SeatsAmount from inserted )


     declare @isEmpty int = (
     select count(*) from Reservations
     where TableID=@tableID
     and
     (reservationDate between DATEADD(hour, -2,@reservationDate) and
     DATEADD(hour, 2, @reservationDate ) )
     )

     if  @isEmpty > 1
     begin
          RAISERROR ('This table is occupied', 16, 1)
          ROLLBACK TRANSACTION
          RETURN
     end

     declare @areSeatsOK int = (
     select AvailableSeats from CovidTableConstraints
     where TableID=@tableID and (@reservationDate between StartDate and
EndDate)
```

```
     )

     declare @tableSeats int = (
     select Seats from Tables
     where TableID=@tableID
     )

     if ISNULL(@areSeatsOK,@tableSeats)<@seatsAmount
     begin
          RAISERROR ('This table does not have enough seats', 16, 1)
          ROLLBACK TRANSACTION
          RETURN
     end

END
GO
```