Autorzy:

Mateusz Mazur

Wojciech Łącki

Bartłomiej Chwast

PODSTAWY BAZ DANYCH | PROJEKT

INSTYTUT INFORMATYKI WIEIT AGH

2021

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

Listopad 2021 - Styczeń 2022

1. Aktorzy:

1. Administrator systemu
2. Menadżer restauracji
3. Pracownik restauracji
4. Firma
5. Klient prywatny (klient indywidualny)
6. System

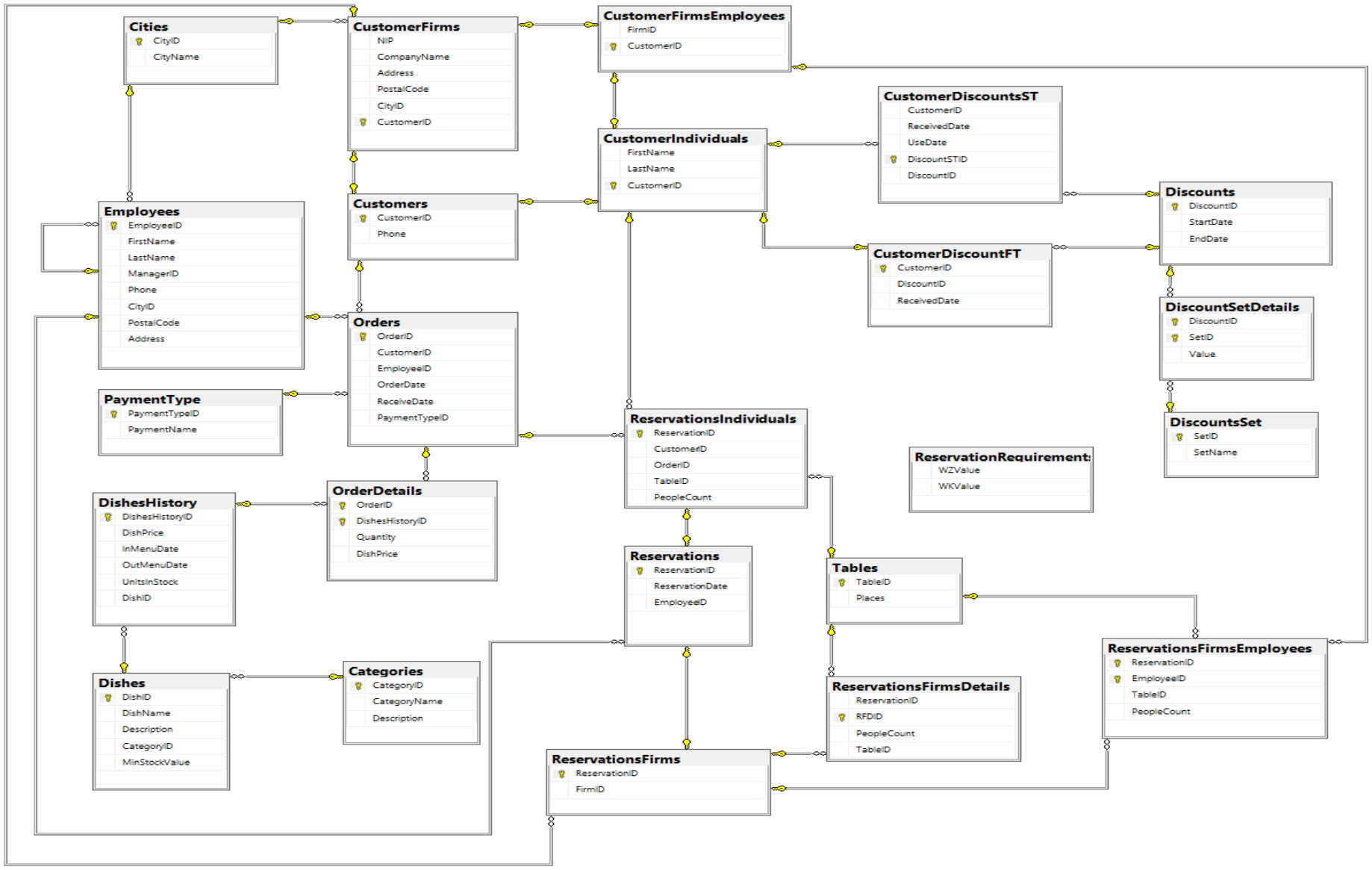
2. Funkcje bazy danych:

1. Wybieranie menu:
    - System wybiera menu z co najmniej dziennym wyprzedzeniem
    - System automatycznie wymienia danie z menu, którego ilość w magazynie (**UnitsInStock** z tabeli **DishesHistory**), jest mniejsza od minimalnej wymaganej wartości (**MinStockValue** z tabeli **Dishes**)
    - W tabeli **DishesHistory** będzie widniała data dodania dania oraz usunięcia z menu
    - Menedżer w dowolnym momencie może zmienić pozycję z menu
2. Obsługa zamówień:
    - Klient ma możliwość złożenia zamówienia online, po wcześniejszym założeniu konta
    - Klient może zamówić dowolną dostępną ilość danego dania, jeśli widnieje ono w menu
    - Jeśli klient chce zamówić danie z kategorii “Owoce morza”, to musi zrobić do poniedziałku poprzedzającego zamówienie
    - System automatycznie zatwierdza zamówienia składane online

3. Zarządzanie rezerwacjami:
    - Klient indywidualny może zarezerwować jeden stolik, gdy spełni odpowiednie założenia widoczne w tabeli **ReservationRequirements**:
        - Wcześniej dokonał **WKValue** zamówień
        - Wartość wszystkich poprzednich zamówień wyniosła przynajmniej **WZValue** złotych
    - Klient indywidualny musi złożyć zamówienie przy rezerwacji stolika
    - Rezerwacja musi zostać zatwierdzona przez pracownika, który przydziela stolik
    - Firmy mogą dokonywać rezerwacji stolików bez imiennego rozróżnienia gości lub dla każdego pracownika imiennie.
4. Zarządzanie rabatami:
    - Rabaty są automatycznie przyznawane po spełnieniu przez klienta aktualnych warunków widocznych w **Discounts** i opisanych w **DiscountSetDetails**.
    - Menedżer może zmienić progi rabatowe oraz dodać nowe warunki rabatów
    - Rabaty podzielone są na dwa typy, dożywotni oraz cykliczny

5. Monitorowanie magazynu:
    - Podczas składania zamówienia sprawdzana jest ilość dostępnych porcji danego dania w magazynie (**UnitsInStock** z tabeli **DishesHistory**)
    - Ilość dostępnych dań **UnitsInStock** jest automatycznie aktualizowana po zatwierdzeniu płatności
    - Gdy zabraknie dania, to jest ono automatycznie usuwane z menu
    - We wtorek menedżer zostanie poinformowany ile konkretnych dań z kategorii “Owoce morza” musi zostać zamówione, aby wszyscy klienci zostali obsłużeni (generowana informacja jest na podstawie tabeli **Orders** i **Order Details** po uwzględnieniu zakresu daty, w którym składane były zamówienia)
6. Generowanie raportów:
    - Menadżer ma możliwość generowanie miesięcznych i tygodniowych raportów, dotyczących rezerwacji stolików, rabatów, menu, a także statystyk zamówienia – dla klientów indywidualnych oraz firm – dotyczących kwot oraz czasu składania zamówień.
7. Generowanie faktur:
    - Pracownik ma możliwość wystawienia faktury dla danego zamówienia lub faktury zbiorczej raz na miesiąc
8. Tworzenie backupu:
    - System tworzy kopię zapasową bazy danych codziennie o 2:00

3. Schemat bazy danych



4. Opisy tabel oraz warunki integralności:

1. Tabela **Categories** – Kategorie dań

CategoryID – (klucz główny) – (**int**) – ID kategorii
CategoryName – (**varchar50**) – Nazwa kategorii
Description – (**varchar500**) – Opis kategorii

```
CREATE TABLE [Categories](
	[CategoryName] [varchar](50) NOT NULL,
	[Description] [varchar](500) NULL,
	[CategoryID] [int] IDENTITY(1,1) 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]
) ON [PRIMARY]
```

Warunki integralności:

- **[CategoryName]** unikalne

```
UNIQUE AK_CategoryName UNIQUE(CategoryName)
```

2. Tabela **Cities** – Słownik miast

CityID (klucz główny) – (**int**) - ID miasta
CityName – (**varchar50**) - Nazwa miasta

```
CREATE TABLE [Cities](
	[CityName] [varchar](50) NOT NULL,
	[CityID] [int] IDENTITY(1,1) NOT NULL,
CONSTRAINT [PK_Cities] PRIMARY KEY CLUSTERED
(
	[CityID] 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]
```

Warunki integralności:

- **[CityName]** unikalne

```
UNIQUE AK_CityName UNIQUE(CityName)
```

3. Tabela **CustomerDiscountFT** – Przyznane zniżki pierwszego typu

CustomerID (klucz główny) – (**int**) – ID klienta indywidualnego
DiscountID (klucz obcy do **[DiscountID]** w **Discount**) – (**int**) – ID zniżki
ReceivedDate – (**date**) – Data przyznania zniżki

```
CREATE TABLE [CustomerDiscountFT](
           [CustomerID] [int] NOT NULL,
           [DiscountID] [int] NOT NULL,
           [ReceivedDate] [date] NOT NULL,
CONSTRAINT [PK_CustomerDiscountFT] 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]

ALTER TABLE [CustomerDiscountFT] WITH CHECK ADD CONSTRAINT
[FK_CustomerDiscountFT_CustomerIndividuals]
FOREIGN KEY([CustomerID])
REFERENCES [CustomerIndividuals] ([CustomerID])
GO

ALTER TABLE [CustomerDiscountFT] WITH CHECK ADD CONSTRAINT
[FK_CustomerDiscountFT_Discounts] FOREIGN KEY([DiscountID])
REFERENCES [Discounts] ([DiscountID])
GO
```

Warunki integralności:

- Data przyznania zniżki **[ReceivedDate]** domyślnie **GETDATE()**

```
CONSTRAINT [DF_CustomerDiscountFT_ReceivedDate]
DEFAULT (GETDATE()) FOR [ReceivedDate]
```

4. Tabela **CustomerDiscountsST** – Przyznane zniżki drugiego typu

DiscountSTID (klucz główny) – (**int**) – ID zniżki drugiego typu
DiscountID (klucz obcy do **[DiscountID]** z **Discounts**) – (**int**) – ID zniżki
CustomerID (klucz obcy do **[CustomerID]** z **CustomerIndividuals**) – (**int**) – ID klienta indywidualnego
ReceivedDate – (**date**) – Data przyznania zniżki
UseDate – (**date** lub **null**) – Data wykorzystania zniżki

```
CREATE TABLE [CustomerDiscountsST](
      [CustomerID] [int] NOT NULL,
      [ReceivedDate] [date] NOT NULL,
      [UseDate] [date] NULL,
      [DiscountID] [int] NOT NULL,
      [DiscountSTID] [int] IDENTITY(1,1) NOT NULL,
CONSTRAINT [PK_CustomerDiscountsST] PRIMARY KEY CLUSTERED
(
      [DiscountSTID] 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]

ALTER TABLE [CustomerDiscountsST] WITH CHECK ADD CONSTRAINT
[FK_CustomerDiscountsST_CustomerIndividuals]
FOREIGN KEY([CustomerID])
REFERENCES [CustomerIndividuals] ([CustomerID])
GO

ALTER TABLE [CustomerDiscountsST] WITH CHECK ADD CONSTRAINT
[FK_CustomerDiscountsST_Discounts] FOREIGN KEY([DiscountID])
REFERENCES [Discounts] ([DiscountID])
GO
```

Warunki integralności:

- Data wykorzystania zniżki **[UseDate]** nie może być wcześniejsza od daty otrzymania zniżki **[ReceivedDate]**

```
CONSTRAINT [CK_UseDate] CHECK (([UseDate] >= [ReceivedDate]))
```

- Data przyznania zniżki **[ReceivedDate]** domyślnie **GETDATE()**

```
CONSTRAINT [DF_CustomerDiscountST_ReceivedDate] DEFAULT
(GETDATE()) FOR [ReceivedDate]
```

- Data wykorzystania zniżki **[UseDate]** domyślnie **null**

```
CONSTRAINT [DF_CustomerDiscountST_UseDate] DEFAULT (NULL) FOR
[UseDate]
```

5. Tabela **CustomersFirms** – Firmy zarejestrowane jako klient

CustomerID (klucz główny) – (**int**) – ID klienta (firmy)
CityID (klucz obcy do **[CityID]** z **Cities**) – (**int**) – ID miasta, w którym firma jest zarejestrowana
NIP – (**nchar10**) – Numer NIP
CompanyName – (**varchar50**) – Nazwa firmy
Address – (**varchar50**) – Adres firmy
PostalCode – (**varchar50**) – Kod pocztowy

```
CREATE TABLE [CustomerFirms](
      [NIP] [nchar](10) NOT NULL,
      [CompanyName] [varchar](50) NOT NULL,
      [Address] [varchar](50) NOT NULL,
      [PostalCode] [varchar](50) NOT NULL,
      [CityID] [int] NOT NULL,
      [CustomerID] [int] NOT NULL,
CONSTRAINT [PK_CustomerFirms] 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]

ALTER TABLE [CustomerFirms] WITH CHECK ADD CONSTRAINT
[FK_CustomerFirms_Cities] FOREIGN KEY([CityID])
REFERENCES [Cities] ([CityID])
GO

ALTER TABLE [CustomerFirms] WITH CHECK ADD CONSTRAINT
[FK_CustomerFirms_Customers] FOREIGN KEY([CustomerID])
REFERENCES [Customers] ([CustomerID])
GO

ALTER TABLE [CustomerFirmsEmployees] WITH CHECK ADD CONSTRAINT
[FK_CustomerFirmsEmployees_CustomerFirms] FOREIGN KEY([FirmID])
REFERENCES [CustomerFirms] ([CustomerID])
GO
```

```
ALTER TABLE [CustomerFirmsEmployees] WITH CHECK ADD CONSTRAINT
[FK_CustomerFirmsEmployees_CustomerIndividuals]
FOREIGN KEY([CustomerID])
REFERENCES [CustomerIndividuals] ([CustomerID])
GO
```

Warunki integralności:

- **[NIP]** unikalne

```
UNIQUE AK_NIP UNIQUE(NIP)
```

- **[NIP]** składa się z cyfr

```
CONSTRAINT [CKNIP] CHECK (ISNUMERIC([NIP]) = 1)
```

- **[CompanyName]** unikalne

```
UNIQUE AK_CompanyName UNIQUE(CompanyName)
```

6. Tabela **CustomersFirmsEmployees** – Przypisanie klienta do firmy

CustomerID (klucz główny) – (**int**) – ID klienta indywidualnego
FirmID (klucz obcy do **[CustomersID]** z **CustomerFirms**) – (**int**) – ID firmy, w której pracuje

```
CREATE TABLE [CustomerFirmsEmployees](
      [FirmID] [int] NOT NULL,
      [CustomerID] [int] NOT NULL,
CONSTRAINT [PK_CustomerFirmsEmployees] 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]

ALTER TABLE [CustomerFirmsEmployees] WITH CHECK ADD CONSTRAINT
[FK_CustomerFirmsEmployees_CustomerFirms] FOREIGN KEY([FirmID])
REFERENCES [CustomerFirms] ([CustomerID])
GO

ALTER TABLE [CustomerFirmsEmployees] WITH CHECK ADD CONSTRAINT
[FK_CustomerFirmsEmployees_CustomerIndividuals]
FOREIGN KEY([CustomerID])
REFERENCES [CustomerIndividuals] ([CustomerID])
GO
```

7. Tabela **CustomersIndividuals** – Klient indywidualny

CustomerID (klucz główny) – (**int**) – ID klienta indywidualnego
FirstName – (**varchar50**) – Imię klienta
LastName – (**varchar50**) – Nazwisko klienta

```
CREATE TABLE [CustomerIndividuals](
      [FirstName] [varchar](50) NOT NULL,
      [LastName] [varchar](50) NOT NULL,
      [CustomerID] [int] NOT NULL,
CONSTRAINT [PK_CustomerIndividuals] 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]

ALTER TABLE [CustomerIndividuals] WITH CHECK ADD CONSTRAINT
[FK_CustomerIndividuals_Customers] FOREIGN KEY([CustomerID])
REFERENCES [Customers] ([CustomerID])
GO
```

8. Tabela **Customers** – Zbiorcza tabela klientów

CustomerID (klucz główny) – (**int**) – ID klienta
Phone – (**nchar9**) – Numer telefonu

```
CREATE TABLE [Customers](
      [Phone] [nchar](9) NOT NULL,
      [CustomerID] [int] IDENTITY(1,1) 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]
```

Warunki integralności:

- Numer telefonu [**Phone**] składa się z samych cyfr

```
CONSTRAINT [CKPhone] CHECK (ISNUMERIC([Phone]) = 1)
```

- Numer telefonu [**Phone**] jest unikalny

```
UNIQUE AK_Phone UNIQUE(Phone)
```

9. Tabela **Discounts** – Zniżki

DiscountID (klucz główny) – (**int**) – ID zniżki
StartDate – (**date**) – Data, od której obowiązuje zniżka
EndDate – (**date** lub **null**) – Data, do której obowiązuje zniżka

```
CREATE TABLE [Discounts](
	[StartDate] [date] NOT NULL,
	[EndDate] [date] NULL,
	[DiscountID] [int] IDENTITY(1,1) NOT NULL,
CONSTRAINT [PK_Discounts] 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]
```

Warunki integralności:

- Data, do której obowiązuje zniżka **[EndDate]** nie może być wcześniejsza od daty, od której obowiązuje zniżka **[StartDate]**

```
CONSTRAINT [CK_EndDate] CHECK (([EndDate] > [StartDate]))
```

- Data, od której obowiązuje zniżka **[StartDate]** domyślnie **GETDATE()**

```
CONSTRAINT [DF_Discounts_StartDate] DEFAULT (GETDATE()) FOR
[StartDate]
```

- Data, do której obowiązuje zniżka **[EndDate]** domyślnie **null**

```
CONSTRAINT [DF_Discounts_EndDate] DEFAULT ((NULL)) FOR [EndDate]
```

10. Tabela **DiscountSetDetails** – Wysokości zniżek

DiscountID (klucz główny) – (**int**) – ID zniżki
SetID (klucz główny) – (**int**) – ID parametru
Value – (**int**) – Wysokość zniżki

```
CREATE TABLE [DiscountSetDetails](
      [DiscountID] [int] NOT NULL,
      [SetID] [int] NOT NULL,
      [Value] [int] NOT NULL,
CONSTRAINT [PK_DiscountSetDetails] PRIMARY KEY CLUSTERED
(
      [DiscountID] ASC,
      [SetID] 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]

ALTER TABLE [DiscountSetDetails] WITH CHECK ADD CONSTRAINT
[FK_DiscountSetDetails_Discounts] FOREIGN KEY([DiscountID])
REFERENCES [Discounts] ([DiscountID])
GO

ALTER TABLE [DiscountSetDetails] WITH CHECK ADD CONSTRAINT
[FK_DiscountSetDetails_DiscountsSet] FOREIGN KEY([SetID])
REFERENCES [DiscountsSet] ([SetID])
GO
```

Warunki integralności:

- Wysokość zniżki **[Value]** domyślnie **0**

```
CONSTRAINT [DF_DiscountSetDetails_Value] DEFAULT ((0)) FOR [Value]
```

11. Tabela **DiscountsSet** – Parametry zniżek

SetID (klucz główny) – (**int**) – ID parametru
SetName – (**varchar50**) – Nazwa parametru

```
CREATE TABLE [DiscountsSet](
      [SetName] [varchar](50) NOT NULL,
      [SetID] [int] IDENTITY(1,1) NOT NULL,
CONSTRAINT [PK_DiscountsSet] PRIMARY KEY CLUSTERED
(
      [SetID] 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]
```

Warunki integralności:

- Nazwa parametru [**SetName**] jest unikalna

```
UNIQUE AK_SetName UNIQUE(SetName)
```

12. Tabela **Dishes** – Opis dania

DishID (klucz główny) – (**int**) – ID dania
CategoryID (klucz obcy do **[CategoryID]** z **Categories**) – (**int**) – ID kategorii
DishName – (**varchar50**) – Nazwa dania
MinStockValue – (**int**) – Minimalna ilość dania w magazynie aby móc je włączyć do menu
Description – (**varchar500** lub **null**) – Opis dania
StartUnits – (**int**) – Startowa ilość dania po włączeniu do menu
BasicDishPrice – (**decimal(10, 2)**) – Startowa cena dania po włączeniu do menu

```
CREATE TABLE [Dishes](
      [DishName] [varchar](50) NOT NULL,
      [Description] [varchar](500) NULL,
      [CategoryID] [int] NOT NULL,
      [MinStockValue] [int] NULL,
      [DishID] [int] IDENTITY(1,1) NOT NULL,
      [StartUnits] [int] NOT NULL,
     [BasicDishPrice] [decimal](10, 2) 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]
) ON [PRIMARY]

ALTER TABLE [Dishes] WITH CHECK ADD CONSTRAINT
[FK_Dishes_Categories] FOREIGN KEY([CategoryID])
REFERENCES [Categories] ([CategoryID])
GO
```

Warunki integralności:

- Minimalna ilość dania w magazynie aby móc je włączyć do menu [**MinStockValue**] musi być większa od 0

```
CONSTRAINT [CK_Dishes] CHECK (([MinStockValue] > 0))
```

- Startowa ilość dań po dodaniu do menu (**StartUnits**) domyślnie 30

```
CONSTRAINT [DF_Dishes_StartUnits] DEFAULT (30) FOR [StartUnits]
```

- Startowa cena dania po dodaniu do menu (**BasicDishPrice**) domyślnie 0

```
CONSTRAINT [DF_Dishes_BasicDishPrice] DEFAULT (0) FOR
[BasicDishPrice]
```

13. Tabela **DishesHistory** – Historia dań z menu

DishesHistoryID (klucz główny) – (**int**) – ID zapisu dania z menu
DishID (klucz obcy do **[DishID]** z **Dishes**) – (**int**) – ID dania
DishPrice – (**decimal(10, 2**) – Cena dania
InMenuDate – (**date** lub **null**) – Data włączenia dania do menu
OutMenuDate – (**date**) – Data usunięcia dania z menu
UnitsInStock – (**int**) – Ilość dania w magazynie

```
CREATE TABLE [DishesHistory](
      [DishPrice] [decimal](10, 2) NOT NULL,
      [InMenuDate] [date] NOT NULL,
      [OutMenuDate] [date] NULL,
      [UnitsInStock] [int] NOT NULL,
      [DishID] [int] NOT NULL,
      [DishesHistoryID] [int] IDENTITY(1,1) NOT NULL,
CONSTRAINT [PK_DishesHistory] PRIMARY KEY CLUSTERED
(
      [DishesHistoryID] 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]

ALTER TABLE [DishesHistory] WITH CHECK ADD CONSTRAINT
[FK_DishesHistory_Dishes] FOREIGN KEY([DishID])
REFERENCES [Dishes] ([DishID])
GO
```

Warunki integralności:

- Cena dania [**DishPrice**] musi być większa od 0

```
CONSTRAINT [CK_DishesHistory] CHECK (([DishPrice] > 0))
```

- Data usunięcia dania z menu [**OutMenuDate**] nie może być wcześniejsza od daty jego dołączenia [**InMenuDate**]

```
CONSTRAINT [CK_DishesHistory_1] CHECK (([OutMenuDate] IS NULL OR
[OutMenuDate] > [InMenuDate]))
```

- Data włączenia dania do menu [**InMenuDate**] domyślnie **GETDATE()**

```
CONSTRAINT [DF_DishesHistory_InMenuDate] DEFAULT (GETDATE()) FOR
[InMenuDate]
```

- Data usunięcia dania z menu [**OutMenuDate**] domyślnie **null**

```
CONSTRAINT [DF_DishesHistory_OutMenuDate] DEFAULT (NULL) FOR
[OutMenuDate]
```

- Ilość dania w magazynie [**UnitsInStock**] nie może być mniejsza od 0

```
CONSTRAINT [CK_DishesHistory_2] CHECK (([UnitsInStock] >= 0))
```

14. Tabela **Employees** – Pracownicy restauracji

EmployeeID (klucz główny) – (**int**) – ID pracownika
CityID (klucz obcy do **[CityID]** z **Cities**) – (**int**) – ID miasta
ManagerID (klucz obcy do **[EmployeeID]** z **EmployeeID**) – (**int** lub **null**) – identyfikator menadżera
PostalCode – (**varchar50**) – kod pocztowy
FirstName – (**varchar50**) – imię
LastName – (**varchar50**) – nazwisko
Phone – (**nchar9**) – numer telefonu
Address – (**varchar50**) – adres

```
CREATE TABLE [Employees](
	[FirstName] [varchar](50) NOT NULL,
	[LastName] [varchar](50) NOT NULL,
	[ManagerID] [int] NULL,
	[Phone] [nchar](9) NOT NULL,
	[CityID] [int] NOT NULL,
	[PostalCode] [varchar](50) NOT NULL,
	[Address] [varchar](50) NOT NULL,
	[EmployeeID] [int] IDENTITY(1,1) 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]

ALTER TABLE [Employees] WITH CHECK ADD CONSTRAINT
[FK_Employees_Cities] FOREIGN KEY([CityID])
REFERENCES [Cities] ([CityID])
GO

ALTER TABLE [Employees] WITH CHECK ADD CONSTRAINT
[FK_Employees_Employees] FOREIGN KEY([ManagerID])
REFERENCES [Employees] ([EmployeeID])
GO
```

Warunki integralności:

- Numer telefonu [**Phone**] składa się z samych cyfr

```
CONSTRAINT [CK_Employees] CHECK ((ISNUMERIC([phone])))
```

- Numer telefonu [**Phone**] jest unikalny

```
UNIQUE AK_Phone UNIQUE(Phone)
```

15. Tabela **OrderDetails** – Szczegóły zamówienia

OrderID (klucz główny) – (**int**) – ID zamówienia
DishesHistoryID (klucz główny) – (**int**) – ID zapisu dania z menu
Quantity – (**int**) – Zamówiona ilość dania
DishPrice – (**money**) – Cena dania

```
CREATE TABLE [OrderDetails](
      [OrderID] [int] NOT NULL,
      [DishesHistoryID] [int] NOT NULL,
      [Quantity] [int] NOT NULL,
      [DishPrice] [decimal](10, 2) NOT NULL,
CONSTRAINT [PK_Order Details] PRIMARY KEY CLUSTERED
(
      [OrderID] ASC,
      [DishesHistoryID] 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]

ALTER TABLE [OrderDetails] WITH CHECK ADD CONSTRAINT
[FK_OrderDetails_DishesHistory] FOREIGN KEY([DishesHistoryID])
REFERENCES [DishesHistory] ([DishesHistoryID])
GO

ALTER TABLE [OrderDetails] WITH CHECK ADD CONSTRAINT
[FK_OrderDetails_Orders] FOREIGN KEY([OrderID])
REFERENCES [Orders] ([OrderID])
GO
```

Warunki integralności:

- Zamówiona ilość dania [**Quantity**] musi być większa od 0

```
CONSTRAINT [CK_OrderDetails] CHECK (([Quantity] > 0))
```

- Cena dania [**DishPrice**] musi być większa od 0

```
CONSTRAINT [CK_OrderDetails_1] CHECK (([DishPrice] >= 0))
```

16. Tabela **Orders** – Zamówienia

OrderID (klucz główny) – (**int**) – ID zamówienia
CustomerID (klucz obcy do **[CustomerID]** w **Customers**) – (**int**) – ID klienta
EmployeeID (klucz obcy do **[EmployeeID]** w **Employees**) – (**int**) – ID pracownika
PaymentTypeID (klucz obcy do **[PaymentTypeID]** w **PaymentType**) – (**int**) – ID sposobu płatności
OrderDate – (**date**) – data złożenia zamówienia
ReceiveDate – (**datetime**) – data odbioru zamówienia

```
CREATE TABLE [Orders](
      [CustomerID] [int] NOT NULL,
      [EmployeeID] [int] NULL,
      [OrderDate] [date] NOT NULL,
      [ReceiveDate] [datetime] NOT NULL,
      [PaymentTypeID] [int] NOT NULL,
      [OrderID] [int] IDENTITY(1,1) 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]

ALTER TABLE [Orders] WITH CHECK ADD CONSTRAINT
[FK_Orders_Customers] FOREIGN KEY([CustomerID])
REFERENCES [Customers] ([CustomerID])
GO

ALTER TABLE [Orders] WITH CHECK ADD CONSTRAINT
[FK_Orders_Employees] FOREIGN KEY([EmployeeID])
REFERENCES [Employees] ([EmployeeID])
GO

ALTER TABLE [Orders] WITH CHECK ADD CONSTRAINT
[FK_Orders_PaymentType] FOREIGN KEY([PaymentTypeID])
REFERENCES [PaymentType] ([PaymentTypeID])
GO
```

Warunki integralności:

- Data odebrania nie może być wcześniejsza od daty złożenia zamówienia

```
CONSTRAINT [CK_Orders] CHECK (([ReceiveDate] >= [OrderDate]))
```

- Data złożenia zamówienia [**OrderDate**] domyślnie **GETDATE()**

```
CONSTRAINT [DF_Orders_OrderDate] DEFAULT (GETDATE()) FOR
[OrderDate]
```

- Data odbioru zamówienia [**ReceiveDate**] domyślnie **GETDATE()**

```
CONSTRAINT [DF_Orders_ReceiveDate] DEFAULT (GETDATE()) FOR
[ReceiveDate]
```

17. Tabela **PaymentType** – Sposób płatności

PaymentTypeID (klucz główny) – (**int**) – ID sposobu płatności
PaymentName – (**varchar50**) – Nazwa sposobu płatności

```
CREATE TABLE [PaymentType](
      [PaymentName] [varchar](50) NOT NULL,
      [PaymentTypeID] [int] IDENTITY(1,1) NOT NULL,
CONSTRAINT [PK_PaymentType] PRIMARY KEY CLUSTERED
(
      [PaymentTypeID] 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]
```

Warunki integralności:

- **[PaymentName]** unikalne

```
UNIQUE AK_PaymentName UNIQUE(PaymentName)
```

18. Tabela **ReservationRequirements** – Wymagania dotyczące możliwości składania rezerwacji przez klientów indywidualnych

WZValue – (**int**) – Minimalna wartość zamówień
WKValue – (**int**) – Minimalna ilość zamówień

```
CREATE TABLE [ReservationRequirements](
      [WZValue] [int] NOT NULL,
      [WKValue] [int] NOT NULL
) ON [PRIMARY]
```

19. Tabela **Reservations** – Rezerwacje

ReservationID (klucz główny) – (**int**) – ID rezerwacji
ReservationDate – (**date**) – Data na którą została złożona rezerwacja
EmployeeID (klucz obcy do **[EmployeeID]** w **Employees**) – (**int** lub **null**) –ID pracownika, który zaakceptował rezerwację (pojawia się po jej zaakceptowaniu)

```
CREATE TABLE [Reservations](
      [ReservationDate] [date] NOT NULL,
      [EmployeeID] [int] NULL,
      [ReservationID] [int] IDENTITY(1,1) 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]

ALTER TABLE [Reservations] WITH CHECK ADD CONSTRAINT
[FK_Reservations_Employees] FOREIGN KEY([EmployeeID])
REFERENCES [Employees] ([EmployeeID])
GO
```

Warunki integralności:

- ID pracownika **[EmployeeID]** domyślnie null

```
CONSTRAINT [DF_Reservations_EmployeeID] DEFAULT (NULL) FOR
[EmployeeID]
```

- Data rezerwacji **[ReservationDate]** nie może być wcześniejsza niż obecna (w momencie umieszczania rekordu w bazie)

```
CONSTRAINT [CK_Reservations] CHECK (([ReservationDate] >
GETDATE()))
```

20. Tabela **ReservationsFirms** – Rezerwacje złożone przez firmy

ReservationID (klucz główny) – (**int**) – ID rezerwacji
FirmID (klucz obcy do **[CustomerID]** w **CustomerFirms**) – (**int**) – ID klienta firmowego, który złożył daną rezerwację

```
CREATE TABLE [ReservationsFirms](
      [ReservationID] [int] NOT NULL,
      [FirmID] [int] NOT NULL,
CONSTRAINT [PK_ReservationFirms] 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]

ALTER TABLE [ReservationsFirms] WITH CHECK ADD CONSTRAINT
[FK_ReservationsFirms_CustomerFirms] FOREIGN KEY([FirmID])
REFERENCES [CustomerFirms] ([CustomerID])
GO

ALTER TABLE [ReservationsFirms] WITH CHECK ADD CONSTRAINT
[FK_ReservationsFirms_Reservations] FOREIGN KEY([ReservationID])
REFERENCES [Reservations] ([ReservationID])
GO
```

21. Tabela **ReservationsFirmsDetails** – Rezerwacje nieimienne złożone przez firmy

ReservationID (klucz obcy do **[ReservationID]** w **ReservationsFirms**) – (**int**) – ID rezerwacji
RFDID (klucz główny) – (**int**) – Sztuczny klucz główny służący do rozróżniania „podrezerwacji” danej rezerwacji
PeopleCount – (**int**) – Ilość osób, dla których potrzebny jest pojedynczy stolik
TableID (klucz obcy do **[TableID]** w **Tables**) – (**int** lub **null**) – ID stolika przypisanego przez obsługę (pojawia się po zaakceptowaniu rezerwacji)

```
CREATE TABLE [ReservationsFirmsDetails](
      [ReservationID] [int] NOT NULL,
      [PeopleCount] [int] NOT NULL,
      [TableID] [int] NULL,
      [RFDID] [int] IDENTITY(1,1) NOT NULL,
CONSTRAINT [PK_ReservationsFirmsDetails] PRIMARY KEY CLUSTERED
(
      [RFDID] 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]

ALTER TABLE [ReservationsFirmsDetails] WITH CHECK ADD CONSTRAINT
[FK_ReservationsFirmsDetails_ReservationsFirms]
FOREIGN KEY([ReservationID])
REFERENCES [ReservationsFirms] ([ReservationID])
GO

ALTER TABLE [ReservationsFirmsDetails] WITH CHECK ADD CONSTRAINT
[FK_ReservationsFirmsDetails_Tables] FOREIGN KEY([TableID])
REFERENCES [Tables] ([TableID])
GO
```

Warunki integralności:

- ID stolika **[TableID]** domyślnie null

```
CONSTRAINT [DF_ReservationsFirmsDetails_TableID] DEFAULT (NULL)
FOR [TableID]
```

- Ilość osób **[PeopleCount]** nie może być mniejsza niż 2

```
CONSTRAINT [CK_ReservationsFirmsDetails] CHECK (([PeopleCount] >=
2))
```

22. Tabela **ReservationsFirmsEmployees** – Rezerwacje imienne złożone przez firmy

ReservationID (klucz główny) – (**int**) – ID rezerwacji
EmployeeID (klucz główny) – (**int**) – ID klienta będącego pracownikiem klienta firmowego
TableID (klucz obcy do **[TableID]** w **Tables**) – (**int** lub **null**) – ID stolika przypisanego przez obsługę (pojawia się po zaakceptowaniu rezerwacji)
PeopleCount – (**int**) – Ilość osób, dla których potrzebny jest pojedynczy stolik

```
CREATE TABLE [ReservationsFirmsEmployees](
	[ReservationID] [int] NOT NULL,
	[EmployeeID] [int] NOT NULL,
	[TableID] [int] NULL,
	[PeopleCount] [int] NOT NULL,
CONSTRAINT [PK_ReservationDetailsFirms_1] PRIMARY KEY CLUSTERED
(
	[ReservationID] ASC,
	[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]

ALTER TABLE [ReservationsFirmsEmployees] WITH CHECK ADD
CONSTRAINT [FK_ReservationsFirmsEmployees_CustomerFirmsEmployees]
FOREIGN KEY([EmployeeID])
REFERENCES [CustomerFirmsEmployees] ([CustomerID])
GO

ALTER TABLE [ReservationsFirmsEmployees] WITH CHECK ADD
CONSTRAINT [FK_ReservationsFirmsEmployees_ReservationsFirms]
FOREIGN KEY([ReservationID])
REFERENCES [ReservationsFirms] ([ReservationID])
GO

ALTER TABLE [ReservationsFirmsEmployees] WITH CHECK ADD
CONSTRAINT [FK_ReservationsFirmsEmployees_Tables] FOREIGN
KEY([TableID])
REFERENCES [Tables] ([TableID])
GO
```

Warunki integralności:

- ID stolika **[TableID]** domyślnie null

```
CONSTRAINT [DF_ReservationsFirmsEmployees_TableID] DEFAULT (NULL)
FOR [TableID]
```

- Ilość osób **[PeopleCount]** nie może być mniejsza niż 2

```
CONSTRAINT [CK_ReservationsFirmsEmployees] CHECK (([PeopleCount]
>= 2))
```

23. Tabela **ReservationsIndividuals** – Rezerwacje klientów indywidualnych

ReservationID (klucz główny) – (**int**) – ID rezerwacji
CustomerID (klucz obcy do **[CustomerID]** w **CustomerIndividuals**) – (**int**) – ID klienta indywidualnego składającego zamówienia
OrderID (klucz obcy do **[OrderID]** w **Orders**) – (**int**) – ID zamówienia połączonego z rezerwacją
TableID (klucz obcy do **[TableID]** w **Tables**) – (**int** lub null) – ID stolika przypisanego przez obsługę (pojawia się po zaakceptowaniu rezerwacji)
PeopleCount – (**int**) – Ilość osób, dla których jest potrzebny stolik

```
CREATE TABLE [ReservationsIndividuals](
      [ReservationID] [int] NOT NULL,
      [CustomerID] [int] NOT NULL,
      [OrderID] [int] NOT NULL,
      [TableID] [int] NULL,
      [PeopleCount] [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]

ALTER TABLE [ReservationsIndividuals] WITH CHECK ADD CONSTRAINT
[FK_ReservationsIndividuals_CustomerIndividuals]
FOREIGN KEY([CustomerID])
REFERENCES [CustomerIndividuals] ([CustomerID])
GO

ALTER TABLE [ReservationsIndividuals] WITH CHECK ADD CONSTRAINT
[FK_ReservationsIndividuals_Orders] FOREIGN KEY([OrderID])
REFERENCES [Orders] ([OrderID])
GO

ALTER TABLE [ReservationsIndividuals] WITH CHECK ADD CONSTRAINT
[FK_ReservationsIndividuals_Reservations]
FOREIGN KEY([ReservationID])
REFERENCES [Reservations] ([ReservationID])
GO
```

```
ALTER TABLE [ReservationsIndividuals] WITH CHECK ADD CONSTRAINT
[FK_ReservationsIndividuals_Tables] FOREIGN KEY([TableID])
REFERENCES [Tables] ([TableID])
GO
```

Warunki integralności:

- ID stolika **[TableID]** domyślnie null

```
CONSTRAINT [DF_ReservationsIndividuals_TableID] DEFAULT (NULL) FOR
[TableID]
```

- Ilość osób **[PeopleCount]** nie może być mniejsza niż 2

```
CONSTRAINT [CK_ReservationsIndividuals] CHECK  (([PeopleCount] >=
2))
```

24. Tabela **Tables** – Stoliki

TableID (klucz główny) – (**int**) – ID stolika
Places – (**int**) – Ilość miejsc, jakie posiada dany stolik

```
CREATE TABLE [Tables](
      [Places] [int] NOT NULL,
      [TableID] [int] IDENTITY(1,1) 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]
```

Warunki integralności:

- Ilość miejsc **[Places]** nie może być mniejsza niż 2

```
CONSTRAINT [CK_Tables] CHECK (([Places] >= (2)))
```

5. Widoki

1. **CurrentDiscounts** – obecnie obowiązujące parametry zniżek

```
CREATE VIEW [CurrentDiscounts] AS
SELECT D.[DiscountID], DS.[SetName], DSD.[Value]
FROM [Discounts] AS D
INNER JOIN [DiscountSetDetails] AS DSD
ON D.[DiscountID] = DSD.[DiscountID]
INNER JOIN [DiscountsSet] AS DS ON DSD.[SetID] = DS.[SetID]
WHERE D.[EndDate] IS NULL
```

2. **CustomerDiscountFirstType** – klienci, którzy zdobyli zniżkę pierwszego typu wraz z jej wartością i datą otrzymania

```
CREATE VIEW [CustomerDiscountFirstType] AS
SELECT CI.[CustomerID] ,CI.[FirstName] + ' ' + CI.[LastName]
AS 'Name', CDFT.[ReceivedDate], DS.[SetName], DSD.[Value]
FROM [CustomerIndividuals] AS CI
INNER JOIN [CustomerDiscountFT] AS CDFT
ON CI.[CustomerID] = CDFT.[CustomerID]
INNER JOIN [Discounts] AS D ON CDFT.[DiscountID] = D.[DiscountID]
INNER JOIN [DiscountSetDetails] AS DSD
ON D.[DiscountID] = DSD.[DiscountID]
INNER JOIN [DiscountsSet] AS DS ON DS.[SetID] = DSD.[SetID]
```

3. **CustomerDiscountsSecondType** – klienci, którzy mogą skorzystać ze zniżki drugiego typu wraz z jej wartością i datą uzyskania

```
CREATE VIEW [CustomerDiscountsSecondType] AS
SELECT CI.[CustomerID], CI.[FirstName] + ' ' + CI.[LastName]
AS 'Name', CDST.[ReceivedDate], CDST.[UseDate], DS.[SetName],
DSD.[Value]
FROM [CustomerIndividuals] AS CI
INNER JOIN [CustomerDiscountsST] AS CDST
ON CI.[CustomerID] = CDST.[CustomerID]
INNER JOIN [Discounts] AS D ON CDST.[DiscountID] = D.[DiscountID]
INNER JOIN [DiscountSetDetails] AS DSD
ON D.[DiscountID] = DSD.[DiscountID]
INNER JOIN [DiscountsSet] AS DS ON DSD.[SetID] = DS.[SetID]
```

4. **DiscountsFTMonthly** – Ilość przyznanych zniżek pierwszego typu w każdym miesiącu

```
CREATE VIEW [DiscountsFTMonthly] AS
SELECT YEAR(ReceivedDate) AS 'Year', MONTH(ReceivedDate) AS
'Month', COUNT(*) AS 'Discounts Count' FROM CustomerDiscountFT
GROUP BY MONTH(ReceivedDate), YEAR(ReceivedDate)
```

5. **DiscountsFTWeekly** – Ilość przyznanych zniżek pierwszego typu w każdym tygodniu

```
CREATE VIEW [DiscountsFTWeekly] AS
SELECT YEAR(ReceivedDate) AS 'Year', DATEPART(WEEK, ReceivedDate)
AS 'Week', COUNT(*) AS 'Discounts Count' FROM CustomerDiscountFT
GROUP BY DATEPART(WEEK, ReceivedDate), YEAR(ReceivedDate)
```

6. **DiscountsSTMonthly** – Ilość przyznanych zniżek drugiego typu w każdym miesiącu

```
CREATE VIEW [DiscountsSTMonthly] AS
SELECT YEAR(ReceivedDate) AS 'Year', MONTH(ReceivedDate) AS
'Month', COUNT(*) AS 'Discounts Count' FROM CustomerDiscountsST
GROUP BY MONTH(ReceivedDate), YEAR(ReceivedDate)
```

7. **DiscountsSTWeekly** – Ilość przyznanych zniżek drugiego typu w każdym tygodniu

```
CREATE VIEW [DiscountsSTWeekly] AS
SELECT YEAR(ReceivedDate) AS 'Year', DATEPART(WEEK, ReceivedDate)
AS 'Week', COUNT(*) AS 'Discounts Count' FROM CustomerDiscountsST
GROUP BY DATEPART(WEEK, ReceivedDate), YEAR(ReceivedDate)
```

8. **DiscountsThisMonth** – Wszystkie zmiany w zniżkach wprowadzone w bieżącym miesiącu

```
CREATE VIEW [DiscountsThisMonth] AS
SELECT D.[DiscountID], D.[StartDate], D.[EndDate], DS.[SetName],
DSD.[Value]
FROM [Discounts] AS D
INNER JOIN [DiscountSetDetails] AS DSD
ON D.[DiscountID] = DSD.[DiscountID]
INNER JOIN [DiscountsSet] AS DS ON DSD.[SetID] = DS.[SetID]
WHERE DATEDIFF(MONTH, GETDATE(), D.[StartDate]) = 0
```

9. **DiscountsThisWeek** – Wszystkie zmiany w zniżkach wprowadzone w bieżącym tygodniu

```
CREATE VIEW [DiscountsThisWeek] AS
SELECT D.[DiscountID], D.[StartDate], D.[EndDate], DS.[SetName],
DSD.[Value]
FROM [Discounts] AS D
INNER JOIN [DiscountSetDetails] AS DSD
ON D.[DiscountID] = DSD.[DiscountID]
INNER JOIN [DiscountsSet] AS DS ON DSD.[SetID] = DS.[SetID]
WHERE DATEDIFF(WEEK, GETDATE(), D.[StartDate]) = 0
```

10. **DishesCategories** – wszystkie dania wraz z kategoriami

```
CREATE VIEW [DishesCategories] AS
SELECT D.[DishName], C.[CategoryName], C.[Description]
FROM [Dishes] AS D
INNER JOIN [Categories] AS C ON D.[CategoryID] = C.[CategoryID]
```

11. **DishesHistoryPrices** – Wszystkie dania wraz z historią ich cen i występowania w menu

```
CREATE VIEW [DishesHistoryPrices] AS
SELECT TOP(100) PERCENT D.[DishName], DH.[DishPrice],
DH.[InMenuDate], DH.[OutMenuDate]
FROM [DishesHistory] AS DH
INNER JOIN [Dishes] AS D ON DH.[DishID] = D.[DishID]
ORDER BY D.[DishID], DH.[DishPrice] DESC
```

12. **DishesToOrder** – dania, których jest za mało i muszą zostać usunięte z menu

```
CREATE VIEW [DishesToOrder] AS
SELECT D.[DishName], D.[Description], DH.[DishPrice]
FROM [DishesHistory] AS DH
INNER JOIN [Dishes] AS D ON DH.[DishID] = D.[DishID]
WHERE [D.MinStockValue] > DH.[UnitsInStock]
AND DH.[OutMenuDate] IS NULL
```

13. **DishIncome** – Dania w kolejności wygenerowanego przychodu

```
CREATE VIEW [DishIncome] AS
SELECT TOP(100) PERCENT D.[DishID], D.[DishName],
ISNULL(SUM(OD.[Quantity] * OD.[DishPrice]), 0) AS 'Income'
FROM [OrderDetails] AS OD
INNER JOIN [DishesHistory] AS DH
ON OD.[DishesHistoryID] = DH.[DishesHistoryID]
RIGHT OUTER JOIN [Dishes] AS D ON DH.[DishID] = D.[DishID]
GROUP BY OD.[DishesHistoryID], D.[DishID], D.[DishName]
ORDER BY 3 DESC
```

14. **DishPopularity** – Dania w kolejności ilości zamówień

```
CREATE VIEW [DishPopularity] AS
SELECT TOP(100) PERCENT D.[DishID], D.[DishName],
ISNULL(SUM([OD.Quantity]), 0) AS 'Quantity'
FROM [OrderDetails] AS OD
INNER JOIN [DishesHistory] AS DH
ON OD.[DishesHistoryID] = DH.[DishesHistoryID]
RIGHT OUTER JOIN [Dishes] AS D ON DH.[DishID] = D.[DishID]
GROUP BY OD.[DishesHistoryID], D.[DishID], D.[DishName]
ORDER BY 3 DESC
```

15. **FirmsEmployees** – pracownicy klientów firmowych

```
CREATE VIEW [FirmsEmployees] AS
SELECT C.[CustomerID], CI.[FirstName] + ' ' + CI.[LastName] AS
'Name', CF.[CustomerID], CF.[CompanyName] FROM [Customers] AS C
INNER JOIN [CustomerIndividuals] AS CI
ON C.[CustomerID] = CI.[CustomerID]
INNER JOIN [CustomerFirmsEmployees] AS CFE
ON CI.[CustomerID] = CFE.[CustomerID]
INNER JOIN [CustomerFirms] AS CF ON CFE.[FirmID] = CF.[CustomerID]
```

16. **FirmsReservationsCount** – Klienci firmowi wraz z ilością dokonanych rezerwacji

```
CREATE VIEW [FirmsReservationsCount] AS
SELECT CF.[CustomerID], CF.[CompanyName],
ISNULL(COUNT(RF.[ReservationID]), 0) AS 'Count'
FROM [ReservationsFirms] AS RF
RIGHT OUTER JOIN [CustomerFirms] AS CF
ON RF.[FirmID] = CF.[CustomerID]
GROUP BY CF.[CustomerID], CF.[CompanyName]
```

17. **FreeTablesForToday** – Wszystkie pozostałe wolne stoliki na dzisiaj

```
CREATE VIEW [FreeTablesForToday] AS
SELECT T.[TableID], T.[Places] FROM [Reservations] AS R
INNER JOIN [ReservationsFirms] AS RF
ON R.[ReservationID] = RF.[ReservationID]
INNER JOIN [ReservationsFirmsEmployees] AS RFE
ON RF.[ReservationID] = RFE.[ReservationID]
RIGHT OUTER JOIN [Tables] AS T
ON RFE.[TableID] = T.[TableID]
AND DATEDIFF(DAY, GETDATE(), R.[ReservationDate]) = 0
WHERE R.[ReservationID] IS NULL
INTERSECT
SELECT T.[TableID], T.[Places] FROM [Reservations] AS R
INNER JOIN [ReservationsFirms] AS RF
ON R.[ReservationID] = RF.[ReservationID]
INNER JOIN [ReservationsFirmsDetails] AS RFD
ON RF.[ReservationID] = RFD.[ReservationID]
RIGHT OUTER JOIN [Tables] AS T
ON RFD.[TableID] = T.[TableID]
AND DATEDIFF(DAY, GETDATE(), R.[ReservationDate]) = 0
WHERE R.[ReservationID] IS NULL
INTERSECT
SELECT T.[TableID], T.[Places] FROM [Reservations] AS R
INNER JOIN [ReservationsIndividuals] AS RI
ON R.[ReservationID] = RI.[ReservationID]
RIGHT OUTER JOIN [Tables] AS T
ON RI.[TableID] = T.[TableID]
AND DATEDIFF(DAY, GETDATE(), R.[ReservationDate]) = 0
WHERE R.[ReservationID] IS NULL
```

18. **IncomePerCustomerFirmThisMonth** – Łączny przychód wygenerowany z jednego klienta firmowego w bieżącym miesiącu

```
CREATE VIEW [IncomePerCustomerFirmThisMonth] AS
SELECT CF.[CustomerID], CF.[CompanyName], C.[Phone],
SUM(OD.[DishPrice] * OD.[Quantity]) AS 'Income' FROM
[CustomerFirms] AS CF
INNER JOIN [Customers] AS C ON CF.[CustomerID] = C.[CustomerID]
INNER JOIN [Orders] AS O ON C.[CustomerID] = O.[CustomerID]
INNER JOIN [OrderDetails] AS OD ON O.[OrderID] = OD.[OrderID]
WHERE DATEDIFF(MONTH, GETDATE(), O.[OrderDate]) = 0
GROUP BY CF.[CustomerID], CF.[CompanyName], C.[Phone]
```

19. **IncomePerCustomerFirmThisWeek** – Łączny przychód wygenerowany z jednego klienta firmowego w bieżącym tygodniu

```
CREATE VIEW [IncomePerCustomerFirmThisWeek] AS
SELECT CF.[CustomerID], CF.[CompanyName], C.[Phone],
SUM(OD.[DishPrice] * OD.[Quantity]) AS 'Income' FROM
[CustomerFirms] AS CF
INNER JOIN [Customers] AS C ON CF.[CustomerID] = C.[CustomerID]
INNER JOIN [Orders] AS O ON C.[CustomerID] = O.[CustomerID]
INNER JOIN [OrderDetails] AS OD ON O.[OrderID] = OD.[OrderID]
WHERE DATEDIFF(WEEK, GETDATE(), O.[OrderDate]) = 0
GROUP BY CF.[CustomerID], CF.[CompanyName], C.[Phone]
```

20. **IncomePerCustomerIndividualThisMonth** – Łączny przychód wygenerowany z jednego klienta indywidualnego w bieżącym miesiącu

```
CREATE VIEW [IncomePerCustomerIndividualThisMonth] AS
SELECT CI.[CustomerID], CI.[FirstName] + ' ' + CI.[LastName] AS
'Name', C.[Phone], SUM(OD.[DishPrice] * OD.[Quantity]) AS 'Income'
FROM [CustomerIndividuals] AS CI
INNER JOIN [Customers] AS C ON CI.[CustomerID] = C.[CustomerID]
INNER JOIN [Orders] AS O ON C.[CustomerID] = O.[CustomerID]
INNER JOIN [OrderDetails] AS OD ON O.[OrderID] = OD.[OrderID]
WHERE DATEDIFF(MONTH, GETDATE(), O.[OrderDate]) = 0
GROUP BY CI.[CustomerID], CI.[FirstName], CI.[LastName], C.[Phone]
```

21. **IncomePerCustomerIndividualThisWeek** – Łączny przychód wygenerowany z jednego klienta indywidualnego w bieżącym tygodniu

```
CREATE VIEW [IncomePerCustomerIndividualThisWeek] AS
SELECT CI.[CustomerID], CI.[FirstName] + ' ' + CI.[LastName] AS
'Name', C.[Phone], SUM(OD.[DishPrice] * OD.[Quantity]) AS 'Income'
FROM [CustomerIndividuals] AS CI
INNER JOIN [Customers] AS C ON CI.[CustomerID] = C.[CustomerID]
INNER JOIN [Orders] AS O ON C.[CustomerID] = O.[CustomerID]
INNER JOIN [OrderDetails] AS OD ON O.[OrderID] = OD.[OrderID]
WHERE DATEDIFF(WEEK, GETDATE(), O.[OrderDate]) = 0
GROUP BY CI.[CustomerID], CI.[FirstName], CI.[LastName], C.[Phone]
```

22. **IndividualsReservationsCount** – Klienci indywidualni wraz z ilością dokonanych rezerwacji

```
CREATE VIEW [IndividualsReservationsCount] AS
SELECT CI.[CustomerID], CI.[FirstName] + ' ' + CI.[LastName]
AS 'Name', ISNULL(COUNT(RI.[ReservationID]), 0) AS 'Count'
FROM [ReservationsIndividuals] AS RI
RIGHT OUTER JOIN [CustomerIndividuals] AS CI
ON RI.[CustomerID] = CI.[CustomerID]
GROUP BY CI.[CustomerID], CI.[FirstName], CI.[LastName]
```

23. **Menu** – aktualne menu restauracji

```
CREATE VIEW [Menu] AS
SELECT D.[DishName], D.[Description], DH.[DishPrice]
FROM [DishesHistory] AS DH
INNER JOIN [Dishes] AS D ON DH.[DishID] = D.[DishID]
WHERE DH.[OutMenuDate] IS NULL
```

24. **MenuMonthly** – Dania występujące w menu wraz z ceną w każdym miesiącu

```
CREATE VIEW [MenuMonthly] AS
SELECT D.[DishName], D.[Description], DH.[DishPrice],
MONTH(DH.InMenuDate) AS 'Month', YEAR(DH.InMenuDate) AS 'Year'
FROM [DishesHistory] AS DH
INNER JOIN [Dishes] AS D ON DH.[DishID] = D.[DishID]
GROUP BY MONTH(DH.InMenuDate), YEAR(DH.InMenuDate), D.[DishName],
D.[Description], DH.[DishPrice]
```

25. **MenuMonthlyCount** – Dania występujące w menu wraz z ilością pojawień się w menu w każdym miesiącu

```
CREATE VIEW [MenuMonthlyCount] AS
SELECT D.[DishName], D.[Description], COUNT(*) AS 'CountDish',
MONTH(DH.InMenuDate) AS 'Month', YEAR(DH.InMenuDate) AS 'Year'
FROM [DishesHistory] AS DH
INNER JOIN [Dishes] AS D ON DH.[DishID] = D.[DishID]
GROUP BY MONTH(DH.InMenuDate), YEAR(DH.InMenuDate), D.[DishName],
D.[Description]
```

26. **MenuWeekly** – Dania występujące w menu wraz z ceną w każdym tygodniu

```
CREATE VIEW [MenuWeekly] AS
SELECT D.[DishName], D.[Description], DH.[DishPrice],
DATEPART(WEEK, DH.InMenuDate) AS 'Week',
YEAR(DH.InMenuDate) AS 'Year' FROM [DishesHistory] AS DH
INNER JOIN [Dishes] AS D ON DH.[DishID] = D.[DishID]
GROUP BY DATEPART(WEEK, DH.InMenuDate), YEAR(DH.InMenuDate),
D.[DishName], D.[Description], DH.[DishPrice]
```

27. **MenuWeeklyCount** – Dania występujące w menu wraz z ilością pojawień się w menu w każdym tygodniu

```
CREATE VIEW [MenuWeeklyCount] AS
SELECT D.[DishName], D.[Description], COUNT(*) AS 'CountDish',
DATEPART(WEEK, DH.InMenuDate) AS 'Week',
YEAR(DH.InMenuDate) AS 'Year' FROM [DishesHistory] AS DH
INNER JOIN [Dishes] AS D ON DH.[DishID] = D.[DishID]
GROUP BY DATEPART(WEEK, DH.InMenuDate), YEAR(DH.InMenuDate),
D.[DishName], D.[Description]
```

28. **OrdersPerCustomerFirmThisMonth** – Wszystkie zamówienia złożone przez klientów firmowych w bieżącym miesiącu

```
CREATE VIEW [OrdersPerCustomerFirmThisMonth] AS
SELECT CF.[CustomerID], CF.[CompanyName] , C.[Phone],
SUM(OD.[DishPrice] * OD.[Quantity])
AS 'Income', O.[OrderDate], O.[OrderID]
FROM [CustomerFirms] AS CF
INNER JOIN [Customers] AS C ON CF.[CustomerID] = C.[CustomerID]
INNER JOIN [Orders] AS O ON C.[CustomerID] = O.[CustomerID]
INNER JOIN [OrderDetails] AS OD ON O.[OrderID] = OD.[OrderID]
WHERE DATEDIFF(MONTH, GETDATE(), O.[OrderDate]) = 0
GROUP BY CF.[CustomerID], CF.[CompanyName], C.[Phone]
```

29. **OrdersPerCustomerFirmThisWeek** – Wszystkie zamówienia złożone przez klientów firmowych w bieżącym tygodniu

```
CREATE VIEW [OrdersPerCustomerFirmThisWeek] AS
SELECT CF.[CustomerID], CF.[CompanyName] , C.[Phone],
SUM(OD.[DishPrice] * OD.[Quantity])
AS 'Income', O.[OrderDate], O.[OrderID]
FROM [CustomerFirms] AS CF
INNER JOIN [Customers] AS C ON CF.[CustomerID] = C.[CustomerID]
INNER JOIN [Orders] AS O ON C.[CustomerID] = O.[CustomerID]
INNER JOIN [OrderDetails] AS OD ON O.[OrderID] = OD.[OrderID]
WHERE DATEDIFF(WEEK, GETDATE(), O.[OrderDate]) = 0
GROUP BY CF.[CustomerID], CF.[CompanyName], C.[Phone]
```

30. **OrdersPerCustomerIndividualThisMonth** – Wszystkie zamówienia złożone przez klientów indywidualnych w bieżącym miesiącu

```
CREATE VIEW [OrdersPerCustomerIndividualThisMonth] AS
SELECT CI.[CustomerID], CI.[FirstName] + ' ' + CI.[LastName]
AS 'Name', C.[Phone], SUM(OD.[DishPrice] * OD.[Quantity])
AS 'Income', O.[OrderDate], O.[OrderID]
FROM [CustomerIndividuals] AS CI
INNER JOIN [Customers] AS C ON CI.[CustomerID] = C.[CustomerID]
INNER JOIN [Orders] AS O ON C.[CustomerID] = O.[CustomerID]
INNER JOIN [OrderDetails] AS OD ON O.[OrderID] = OD.[OrderID]
WHERE DATEDIFF(MONTH, GETDATE(), O.[OrderDate]) = 0
GROUP BY CI.[CustomerID], CI.[FirstName], CI.[LastName],
C.[Phone], O.[OrderDate], O.[OrderID]
```

31. **OrdersPerCustomerIndividualThisWeek** – Wszystkie zamówienia złożone przez klientów indywidualnych w bieżącym tygodniu

```
CREATE VIEW [OrdersPerCustomerIndividualThisWeek] AS
SELECT CI.[CustomerID], CI.[FirstName] + ' ' + CI.[LastName]
AS 'Name', C.[Phone], SUM(OD.[DishPrice] * OD.[Quantity])
AS 'Income', O.[OrderDate], O.[OrderID]
FROM [CustomerIndividuals] AS CI
INNER JOIN [Customers] AS C ON CI.[CustomerID] = C.[CustomerID]
INNER JOIN [Orders] AS O ON C.[CustomerID] = O.[CustomerID]
INNER JOIN [OrderDetails] AS OD ON O.[OrderID] = OD.[OrderID]
WHERE DATEDIFF(WEEK, GETDATE(), O.[OrderDate]) = 0
GROUP BY CI.[CustomerID], CI.[FirstName], CI.[LastName],
C.[Phone], O.[OrderDate], O.[OrderID]
```

32. **OrdersPerTimeOfDay** – ilość zamówień według pory dnia

```
CREATE VIEW [OrdersPerTimeOfDay] AS
SELECT COUNT(*) AS 'Quantity', 'Morning' AS 'Time of day'
FROM [Orders]
WHERE DATEPART(HOUR, [ReceiveDate]) < 11
UNION
SELECT COUNT(*) AS 'Quantity', 'Midday' AS 'Time of day'
FROM [Orders]
WHERE 11 <= DATEPART(HOUR, [ReceiveDate])
AND DATEPART(HOUR, [ReceiveDate]) < 16
UNION
SELECT COUNT(*) AS 'Quantity', 'Afternoon' AS 'Time of day'
FROM [Orders]
WHERE 16 <= DATEPART(HOUR, [ReceiveDate])
AND DATEPART(HOUR, [ReceiveDate]) < 20
UNION
SELECT COUNT(*) AS 'Quantity', 'Evening' AS 'Time of day'
FROM [Orders]
WHERE 20 <= DATEPART(HOUR, [ReceiveDate])
```

33. **OrdersPerTimeOfDayThisMonth** – ilość zamówień według pory dnia w bieżącym miesiącu

```
CREATE VIEW [OrdersPerTimeOfDay] AS
SELECT COUNT(*) AS 'Quantity', 'Morning' AS 'Time of day'
FROM [Orders]
WHERE DATEPART(HOUR, [ReceiveDate]) < 11
AND DATEDIFF(MONTH, [ReceiveDate], GETDATE()) = 0
UNION
SELECT COUNT(*) AS 'Quantity', 'Midday' AS 'Time of day'
FROM [Orders]
WHERE 11 <= DATEPART(HOUR, [ReceiveDate])
AND DATEPART(HOUR, [ReceiveDate]) < 16
AND DATEDIFF(MONTH, [ReceiveDate], GETDATE()) = 0
UNION
SELECT COUNT(*) AS 'Quantity', 'Afternoon' AS 'Time of day'
FROM [Orders]
WHERE 16 <= DATEPART(HOUR, [ReceiveDate])
AND DATEPART(HOUR, [ReceiveDate]) < 20
AND DATEDIFF(MONTH, [ReceiveDate], GETDATE()) = 0
UNION
SELECT COUNT(*) AS 'Quantity', 'Evening' AS 'Time of day'
FROM [Orders]
WHERE 20 <= DATEPART(HOUR, [ReceiveDate])
AND DATEDIFF(MONTH, [ReceiveDate], GETDATE()) = 0
```

34. **OrdersPerTimeOfDayThisWeek** – ilość zamówień według pory dnia w bieżącym tygodniu

```
CREATE VIEW [OrdersPerTimeOfDay] AS
SELECT COUNT(*) AS 'Quantity', 'Morning' AS 'Time of day'
FROM [Orders]
WHERE DATEPART(HOUR, [ReceiveDate]) < 11
AND DATEDIFF(WEEK, [ReceiveDate], GETDATE()) = 0
UNION
SELECT COUNT(*) AS 'Quantity', 'Midday' AS 'Time of day'
FROM [Orders]
WHERE 11 <= DATEPART(HOUR, [ReceiveDate])
AND DATEPART(HOUR, [ReceiveDate]) < 16
AND DATEDIFF(WEEK, [ReceiveDate], GETDATE()) = 0
UNION
SELECT COUNT(*) AS 'Quantity', 'Afternoon' AS 'Time of day'
FROM [Orders]
WHERE 16 <= DATEPART(HOUR, [ReceiveDate])
AND DATEPART(HOUR, [ReceiveDate]) < 20
AND DATEDIFF(WEEK, [ReceiveDate], GETDATE()) = 0
UNION
SELECT COUNT(*) AS 'Quantity', 'Evening' AS 'Time of day'
FROM [Orders]
WHERE 20 <= DATEPART(HOUR, [ReceiveDate])
AND DATEDIFF(WEEK, [ReceiveDate], GETDATE()) = 0
```

35. **OrdersPriceForToday** – Przychód z zamówień złożonych dzisiaj

```
CREATE VIEW [OrdersPriceForToday] AS
SELECT O.[OrderID], SUM(OD.[DishPrice] * OD.[Quantity]) AS 'Price'
FROM [Orders] AS O
INNER JOIN [OrderDetails] AS OD ON O.[OrderID] = OD.[OrderID]
WHERE DATEDIFF(DAY, O.[ReceiveDate], GETDATE()) = 0
GROUP BY O.[OrderID]
```

36. **OrdersToDo** – Wszystkie zamówienia, które nie zostały jeszcze zrealizowane

```
CREATE VIEW [OrdersToDo] AS
SELECT O.[OrderID], O.[CustomerID], O.[EmployeeID], O.[OrderDate],
O.[ReceiveDate], PT.[PaymentName] FROM [Orders] AS O
INNER JOIN [PaymentType] AS PT ON O.[PaymentTypeID] =
PT.[PaymentTypeID]
WHERE O.[ReceiveDate] > GETDATE()
```

37. **ReservatedTablesMonthly** – Ilość zarezerwowanych stolików w każdym miesiącu

```
CREATE VIEW [ReservatedTablesMonthly] AS
SELECT YEAR(R.ReservationDate) AS 'Year', MONTH(R.ReservationDate)
AS 'Month', COUNT(*) AS 'Reservations Count'
FROM [Reservations] AS R
INNER JOIN [ReservationsFirms] AS RF
ON R.[ReservationID] = RF.[ReservationID]
INNER JOIN [ReservationsFirmsEmployees] AS RFE
ON RF.[ReservationID] = RFE.[ReservationID]
RIGHT OUTER JOIN [Tables] AS T
ON RFE.[TableID] = T.[TableID]
WHERE R.[ReservationID] IS NOT NULL
GROUP BY MONTH(R.ReservationDate), YEAR(R.ReservationDate)
UNION
SELECT YEAR(R.ReservationDate) AS 'Year', MONTH(R.ReservationDate)
AS 'Month', COUNT(*) AS 'Reservations Count'
FROM [Reservations] AS R
INNER JOIN [ReservationsFirms] AS RF
ON R.[ReservationID] = RF.[ReservationID]
INNER JOIN [ReservationsFirmsDetails] AS RFD
ON RF.[ReservationID] = RFD.[ReservationID]
RIGHT OUTER JOIN [Tables] AS T
ON RFD.[TableID] = T.[TableID]
WHERE R.[ReservationID] IS NOT NULL
GROUP BY MONTH(R.ReservationDate), YEAR(R.ReservationDate)
UNION
SELECT YEAR(R.ReservationDate) AS 'Year', MONTH(R.ReservationDate)
AS 'Month', COUNT(*) AS 'Reservations Count'
FROM [Reservations] AS R
INNER JOIN [ReservationsIndividuals] AS RI
ON R.[ReservationID] = RI.[ReservationID]
RIGHT OUTER JOIN [Tables] AS T
ON RI.[TableID] = T.[TableID]
WHERE R.[ReservationID] IS NOT NULL
GROUP BY MONTH(R.ReservationDate), YEAR(R.ReservationDate)
```

38. **ReservatedTablesThisMonth** – Wszystkie rezerwacje stolików dokonane w bieżącym miesiącu

```
CREATE VIEW [ReservatedTablesThisMonth] AS
SELECT R.[ReservationID], R.[ReservationDate], T.[TableID]
FROM [Reservations] AS R
INNER JOIN [ReservationsFirms] AS RF
ON R.[ReservationID] = RF.[ReservationID]
INNER JOIN [ReservationsFirmsEmployees] AS RFE
ON RF.[ReservationID] = RFE.[ReservationID]
RIGHT OUTER JOIN [Tables] AS T
ON RFE.[TableID] = T.[TableID]
AND DATEDIFF(MONTH, GETDATE(), R.[ReservationDate]) = 0
WHERE R.[ReservationID] IS NOT NULL
UNION
SELECT R.[ReservationID], R.[ReservationDate], T.[TableID]
FROM [Reservations] AS R
INNER JOIN [ReservationsFirms] AS RF
ON R.[ReservationID] = RF.[ReservationID]
INNER JOIN [ReservationsFirmsDetails] AS RFD
ON RF.[ReservationID] = RFD.[ReservationID]
RIGHT OUTER JOIN [Tables] AS T
ON RFD.[TableID] = T.[TableID]
AND DATEDIFF(MONTH, GETDATE(), R.[ReservationDate]) = 0
WHERE R.[ReservationID] IS NOT NULL
UNION
SELECT R.[ReservationID], R.[ReservationDate], T.[TableID]
FROM [Reservations] AS R
INNER JOIN [ReservationsIndividuals] AS RFI
ON RF.[ReservationID] = RFI.[ReservationID]
RIGHT OUTER JOIN [Tables] AS T
ON RFI.[TableID] = T.[TableID]
AND DATEDIFF(MONTH, GETDATE(), R.[ReservationDate]) = 0
WHERE R.[ReservationID] IS NOT NULL
```

39. **ReservatedTablesThisWeek** – Wszystkie rezerwacje stolików dokonane w bieżącym tygodniu

```
CREATE VIEW [ReservatedTablesThisWeek] AS
SELECT R.[ReservationID], R.[ReservationDate], T.[TableID]
FROM [Reservations] AS R
INNER JOIN [ReservationsFirms] AS RF
ON R.[ReservationID] = RF.[ReservationID]
INNER JOIN [ReservationsFirmsEmployees] AS RFE
ON RF.[ReservationID] = RFE.[ReservationID]
RIGHT OUTER JOIN [Tables] AS T
ON RFE.[TableID] = T.[TableID]
AND DATEDIFF(WEEK, GETDATE(), R.[ReservationDate]) = 0
WHERE R.[ReservationID] IS NOT NULL
UNION
SELECT R.[ReservationID], R.[ReservationDate], T.[TableID]
FROM [Reservations] AS R
INNER JOIN [ReservationsFirms] AS RF
ON R.[ReservationID] = RF.[ReservationID]
INNER JOIN [ReservationsFirmsDetails] AS RFD
ON RF.[ReservationID] = RFD.[ReservationID]
RIGHT OUTER JOIN [Tables] AS T
ON RFD.[TableID] = T.[TableID]
AND DATEDIFF(WEEK, GETDATE(), R.[ReservationDate]) = 0
WHERE R.[ReservationID] IS NOT NULL
UNION
SELECT R.[ReservationID], R.[ReservationDate], T.[TableID]
FROM [Reservations] AS R
INNER JOIN [ReservationsIndividuals] AS RFI
ON RF.[ReservationID] = RFI.[ReservationID]
RIGHT OUTER JOIN [Tables] AS T
ON RFI.[TableID] = T.[TableID]
AND DATEDIFF(WEEK, GETDATE(), R.[ReservationDate]) = 0
WHERE R.[ReservationID] IS NOT NULL
```

40. **ReservatedTablesWeekly** – Ilość zarezerwowanych stolików w każdym tygodniu

```
CREATE VIEW [ReservatedTablesWeekly] AS
SELECT YEAR(R.ReservationDate) AS 'Year',
DATEPART(WEEK, R.ReservationDate) AS 'Week', COUNT(*) AS
'Reservations Count' FROM [Reservations] AS R
INNER JOIN [ReservationsFirms] AS RF
ON R.[ReservationID] = RF.[ReservationID]
INNER JOIN [ReservationsFirmsEmployees] AS RFE
ON RF.[ReservationID] = RFE.[ReservationID]
RIGHT OUTER JOIN [Tables] AS T
ON RFE.[TableID] = T.[TableID]
WHERE R.[ReservationID] IS NOT NULL
GROUP BY DATEPART(WEEK, R.ReservationDate),
YEAR(R.ReservationDate)
UNION
SELECT YEAR(R.ReservationDate) AS 'Year',
DATEPART(WEEK, R.ReservationDate) AS 'Week', COUNT(*) AS
'Reservations Count' FROM [Reservations] AS R
INNER JOIN [ReservationsFirms] AS RF
ON R.[ReservationID] = RF.[ReservationID]
INNER JOIN [ReservationsFirmsDetails] AS RFD
ON RF.[ReservationID] = RFD.[ReservationID]
RIGHT OUTER JOIN [Tables] AS T
ON RFD.[TableID] = T.[TableID]
WHERE R.[ReservationID] IS NOT NULL
GROUP BY DATEPART(WEEK, R.ReservationDate),
YEAR(R.ReservationDate)
UNION
SELECT YEAR(R.ReservationDate) AS 'Year',
DATEPART(WEEK, R.ReservationDate) AS 'Week', COUNT(*) AS
'Reservations Count' FROM [Reservations] AS R
INNER JOIN [ReservationsIndividuals] AS RI
ON R.[ReservationID] = RI.[ReservationID]
RIGHT OUTER JOIN [Tables] AS T
ON RI.[TableID] = T.[TableID]
WHERE R.[ReservationID] IS NOT NULL
GROUP BY DATEPART(WEEK, R.ReservationDate),
YEAR(R.ReservationDate)
```

41. **ReservationsThisMonth** – Wszystkie rezerwacje dokonane w bieżącym miesiącu

```
CREATE VIEW [ReservationsThisMonth] AS
SELECT * FROM [Reservations] AS R
WHERE DATEDIFF(MONTH, R.[ReservationDate], GETDATE()) = 0
```

42. **ReservationsThisWeek** – Wszystkie rezerwacje dokonane w bieżącym tygodniu

```
CREATE VIEW [ReservationsThisWeek] AS
SELECT * FROM [Reservations] AS R
WHERE DATEDIFF(WEEK, R.[ReservationDate], GETDATE()) = 0
```

43. **ReservationsToAccept** – Rezerwacje oczekujące na akceptację przez pracownika

```
CREATE VIEW [ReservationsToAccept] AS
SELECT R.[ReservationID], R.[ReservationDate], CI.[FirstName] + '
' + CI.[LastName] AS 'Customer name', CI.[CustomerID]
FROM [Reservations] AS R
INNER JOIN [ReservationsIndividuals] AS RI
ON R.[ReservationID] = RI.[ReservationID]
INNER JOIN [CustomerIndividuals] AS CI
ON RI.[CustomerID] = CI.[CustomerID]
WHERE R.[EmployeeID] IS NULL
UNION
SELECT R.[ReservationID], R.[ReservationDate], CF.[CompanyName] AS
'Customer Name', CF.[CustomerID]
FROM [Reservations] AS R
INNER JOIN [ReservationsFirms] AS RF
ON R.[ReservationID] = RF.[ReservationID]
INNER JOIN [CustomerFirms] AS CF
ON RF.[FirmID] = CF.[CustomerID]
WHERE R.[EmployeeID] IS NULL
```

44. **ReservationsForToday** – Rezerwacje na dzisiaj

```
CREATE VIEW [ReservationsForToday] AS
SELECT * FROM [Reservations] AS R
WHERE DATEDIFF(DAY, R.[ReservationDate], GETDATE()) = 0
```

45. **SeaFoodNeededForThisWeekend** – Dania z owocami morza, które należy przygotować na ten weekend

```
CREATE VIEW [SeaFoodNeededForThisWeekend] AS
SELECT D.[DishID], D.[DishName], SUM(OD.[Quantity]) AS 'Quantity'
FROM [Orders] AS O
INNER JOIN [OrderDetails] AS OD ON O.[OrderID] = OD.[OrderID]
INNER JOIN [DishesHistory] AS DH
ON OD.DishesHistoryID = DH.DishesHistoryID
INNER JOIN [Dishes] AS D ON DH.DishID = D.DishID
INNER JOIN [Categories] AS Ca ON D.CategoryID = Ca.CategoryID
WHERE Ca.CategoryID = 2
AND DATEDIFF(WEEK, GETDATE(), O.[ReceiveDate]) = 0
GROUP BY D.[DishID], D.[DishName]
```

6. Procedury

1. **AddCategory** – dodaje nową kategorię dania

```
CREATE PROCEDURE [AddCategory]
    @CategoryName varchar(50),
    @Description varchar(500) = NULL
AS
BEGIN
    BEGIN TRY
       INSERT INTO [Categories]([CategoryName], [Description])
       VALUES (@Categoryname, @Description)
       DECLARE  @CategoryID int
       SELECT @CategoryID = SCOPE_IDENTITY()
    END TRY
    BEGIN CATCH
    DELETE FROM [Categories]
       WHERE [CategoryID] = @CategoryID

    DECLARE @errorMsg nvarchar(2048) = 'Cannot add new Category.
    Error message: '
    +ERROR_MESSAGE();
    THROW 52000, @errorMsg, 1;
    END CATCH
END
GO
```

2. **AddCity**– dodaje nowe miasto

```
CREATE PROCEDURE [AddCity]
    @CityName varchar(50)
AS
BEGIN
    BEGIN TRY
      INSERT INTO [Cities]([CityName])
      VALUES (@CityName)
      DECLARE  @CityID int
      SELECT @CityID = SCOPE_IDENTITY()
    END TRY
    BEGIN CATCH
    DELETE FROM [Cities]
      WHERE [CityID] = @CityID
    DECLARE @errorMsg nvarchar(2048) = 'Cannot add new City. Error
    message: '
    +ERROR_MESSAGE();
    THROW 52000, @errorMsg, 1;
    END CATCH
END
GO
```

3. **AddDiscount** – dodaje nową zniżkę

```
CREATE PROCEDURE [AddDiscount]
AS
BEGIN
    BEGIN TRY

       INSERT INTO [Discounts]([StartDate])
       VALUES (GETDATE())
       DECLARE  @DiscountID int
       SELECT @DiscountID = SCOPE_IDENTITY()
    END TRY
    BEGIN CATCH
    DELETE FROM [Discounts]
       WHERE [DiscountID] = @DiscountID
    DECLARE @errorMsg nvarchar(2048) = 'Cannot add new discount.
    Error message: '
    +ERROR_MESSAGE();
    THROW 52000, @errorMsg, 1;
    END CATCH
END
GO
```

4. **AddDish**– dodaje nowe danie

```
CREATE PROCEDURE [AddDish]
    @DishName varchar(50),
    @Description varchar(500) = NULL,
    @CategoryName varchar(50),
    @MinStockValue int = NULL,
    @StartUnits int

AS
BEGIN
    BEGIN TRY
       IF NOT EXISTS
            (
            SELECT [CategoryID] FROM [Categories]
            WHERE [CategoryName] = @CategoryName
            )
            BEGIN
                 ;THROW 52000, 'Category does not exist.', 1
            END

       IF (@StartUnits < @MinStockValue)
       BEGIN
                  ;THROW 52000, 'Units on start has to be greater
                   than minimal value', 1
       END
       DECLARE @CategoryID int
       SET @CategoryID = (
       SELECT [CategoryID] FROM [Categories]
      WHERE [CategoryName] = @CategoryName
       );

       INSERT INTO [Dishes]([DishName], [Description],
      [CategoryID], [MinStockValue])
       VALUES (@DishName, @Description, @CategoryID,
      @MinStockValue)
       DECLARE  @DishID int
       SELECT @DishID = SCOPE_IDENTITY()
    END TRY
    BEGIN CATCH
    DELETE FROM [Dishes]
       WHERE [DishID] = @DishID
```

```
    DECLARE @errorMsg nvarchar(2048) = 'Cannot add new Dish. Error
    message: '
    +ERROR_MESSAGE();
    THROW 52000, @errorMsg, 1;
    END CATCH
END
GO
```

5. **AddDishToMenu** – dodaje danie do aktualnego menu

```
CREATE PROCEDURE [AddDishToMenu]
    @DishName varchar(50),
    @DishPrice int,
    @UnitsInStock int
AS
BEGIN
    BEGIN TRY
    DECLARE @DishID int;
      IF NOT EXISTS
      (
            SELECT [DishID] FROM [Dishes]
            WHERE [DishName] = @DishName
      )
      BEGIN
            ;THROW 52000, 'Dish does not exist.', 1
      END

      SET @DishID = (
            SELECT [DishID] FROM [Dishes]
            WHERE [DishName] = @DishName
      );

      IF EXISTS
      (
            SELECT [DishID] FROM [DishesHistory]
            WHERE [DishID] = @DishID AND [OutMenuDate] IS NULL
      )
      BEGIN
            ;THROW 52000, 'Dish is already in menu.', 1
      END

      IF NOT EXISTS
      (
            SELECT [DishID] FROM [Dishes] WHERE [DishName] =
            @DishName AND @UnitsInStock >= [MinStockValue]
      )
      BEGIN
            ;THROW 52000, 'There is not enough portions of this
            dish in stock.', 1
      END
```

```
      INSERT INTO [DishesHistory]([DishPrice], [InMenuDate],
      [OutMenuDate], [UnitsInStock], [DishID])
      VALUES (@DishPrice, GETDATE(), NULL, @UnitsInStock, @DishID)
      DECLARE  @DishesHistoryID int
      SELECT @DishesHistoryID = SCOPE_IDENTITY()
    END TRY
    BEGIN CATCH
    DELETE FROM [DishesHistory]
      WHERE [DishesHistoryID] = @DishesHistoryID
    DECLARE @errorMsg nvarchar(2048) = 'Cannot add new dish to
    Menu. Error message: '
    +ERROR_MESSAGE();
    THROW 52000, @errorMsg, 1;
    END CATCH
END
GO
```

6. **AddDishToOrder** – dodaje danie do zamówienia

```
CREATE PROCEDURE [AddDishToOrder]
    @OrderID int,
    @DishesHistoryID int,
    @Quantity int,
    @CustomerID int
AS
BEGIN
    BEGIN TRY
      DECLARE @Discount int;
      DECLARE @BasicDishPrice decimal(10, 2);
      DECLARE @UnitsInStock int;
      DECLARE @DiscountIDFT int = NULL;
      DECLARE @DiscountIDST int = NULL;


      IF NOT EXISTS
      (
            (SELECT [CustomerID] FROM [Customers] WHERE
[CustomerID] = @CustomerID)
      )
      BEGIN
            ;THROW 52000, 'Customer does not exist.', 1
      END

      IF NOT EXISTS
      (
            SELECT [UnitsInStock] FROM [DishesHistory]
            WHERE [DishesHistoryID] = @DishesHistoryID
            AND [OutMenuDate] IS NULL
      )
      BEGIN
            ;THROW 52000, 'Dish does not exist in menu.', 1
      END

      IF NOT EXISTS
      (
            (SELECT [OrderID] FROM [Orders]
            WHERE [OrderID] = @OrderID)
      )
      BEGIN
```

```
            ;THROW 52000, 'Order does not exist.', 1
      END

      SET @UnitsInStock = (
            SELECT [UnitsInStock] FROM [DishesHistory]
            WHERE [DishesHistoryID] = @DishesHistoryID
            AND [OutMenuDate] IS NULL
      );

      SET @BasicDishPrice=
      (
      SELECT [DishPrice] FROM [DishesHistory]
     WHERE [DishesHistoryID] = @DishesHistoryID
     AND OutMenuDate IS NULL
      )
      IF
      (
      @UnitsInStock < @Quantity
      )
      BEGIN
            ;THROW 52000, 'There is not enough dish in stock.', 1
      END

      IF EXISTS
      (
            SELECT [CustomerID] FROM [CustomerIndividuals]
            WHERE [CustomerID] = @CustomerID
      )
      BEGIN

            IF EXISTS
            (
                  SELECT [DiscountID] FROM [CustomerDiscountFT]
                  WHERE [CustomerID] = @CustomerID
            )
            BEGIN
                  SET @DiscountIDFT = (
                        SELECT [DiscountID] FROM
[CustomerDiscountFT] WHERE [CustomerID] = @CustomerID
                  );
            END
```

```
IF EXISTS
(
      SELECT [DiscountID] FROM [CustomerDiscountsST]
     WHERE [CustomerID] = @CustomerID
     AND [UseDate] IS NULL
)
BEGIN
      SET @DiscountIDST=(
            SELECT [DiscountID] FROM
            [CustomerDiscountsST]
            WHERE [CustomerID] = @CustomerID
            AND [UseDate] IS NULL
      );
END

IF
(
      (SELECT [Value] FROM [DiscountSetDetails] AS DSD
      INNER JOIN [DiscountsSet] AS DS
     ON DSD.[SetID] = DS.[SetID]
      WHERE DS.[SetName] = 'R'
     AND DSD.[DiscountID] = @DiscountIDFT)
      >=
      (SELECT [Value] FROM [DiscountSetDetails] AS DSD
      INNER JOIN [DiscountsSet] AS DS
     ON DSD.[SetID] = DS.[SetID]
      WHERE DS.[SetName] = 'R'
     AND DSD.[DiscountID] = @DiscountIDST)
)
BEGIN
      SET @Discount =
      (SELECT [Value] FROM [DiscountSetDetails] AS DSD
      INNER JOIN [DiscountsSet] AS DS
     ON DSD.[SetID] = DS.[SetID]
      WHERE DS.[SetName] = 'R'
     AND DSD.[DiscountID] = @DiscountIDFT)
END
ELSE
BEGIN
      SET @Discount =
      (SELECT [Value] FROM [DiscountSetDetails] AS DSD
      INNER JOIN [DiscountsSet] AS DS
```

```
            ON DSD.[SetI] = DS.[SetID]
            WHERE DS.[SetName] = 'R'
            AND DSD.[DiscountID] = @DiscountIDST)
      END
END
ELSE
BEGIN
      SET @Discount = 0
END

IF EXISTS
(
      SELECT [DishesHistoryID] FROM [DishesHistory] AS DH
      INNER JOIN [Dishes] AS D ON DH.[DishID] = D.[DishID]
      INNER JOIN Categories AS C
      ON D.[CategoryID] = C.[CategoryID]
      WHERE DH.[DishesHistoryID] = @DishesHistoryID
      AND C.[CategoryName] = 'Owoce morza'
)
BEGIN
      IF NOT
      (
            DATEPART(WEEKDAY, @ReceiveDate) BETWEEN 4 AND 6
      )
      BEGIN
      ;THROW 52000, 'Seafood can be order only between
      thursday and saturday.', 1
      END

      IF
      (
            DATEDIFF(WEEK, @OrderDate, @ReceiveDate) = 0
      )
      AND
      (
            DATEPART(WEEKDAY, @ReceiveDate) NOT LIKE 1
      )
      BEGIN
      ;THROW 52000, 'Seafood must be ordered before
      Tuesday.', 1
      END
END
```

```
        UPDATE [DishesHistory]
        SET [UnitsInStock] = @UnitsInStock - @Quantity
        WHERE [DishesHistoryID] = @DishesHistoryID
       INSERT INTO [OrderDetails]([OrderID], [DishesHistoryID],
       [Quantity], [DishPrice])
        VALUES (@OrderID, @DishesHistoryID, @Quantity,
       (@BasicDishPrice * (100 - @Discount) ) / 100)

    END TRY
    BEGIN CATCH
    DELETE FROM [OrderDetails]
       WHERE [OrderID] = @OrderID
    DELETE FROM [Orders]
       WHERE [OrderID] = @OrderID
    DECLARE @errorMsg nvarchar(2048) = 'Cannot add dish to order.
    Order is removed. Error message: '
    +ERROR_MESSAGE();
    THROW 52000, @errorMsg, 1;
    END CATCH
END
GO
```

7. **AddFirmCustomer** – dodaje klienta indywidualnego

```
CREATE PROCEDURE [AddFirmCustomer]
    @CompanyName varchar(50),
    @NIP nchar(10),
    @Address varchar(50),
    @PostalCode varchar(50),
    @Phone nchar(9),
    @CityName varchar(50)
AS
BEGIN
    BEGIN TRY

    IF NOT EXISTS
    (
    SELECT [CityID] FROM [Cities] WHERE [CityName] = @CityName
    )
    BEGIN
      ;THROW 52000, 'City does not exist.', 1
    END

    DECLARE @CityID int;
    SET @CityID = (
    SELECT [CityID] FROM [Cities] WHERE [CityName] = @CityName
    );


    INSERT INTO [Customers]([Phone])
    VALUES (@Phone)

    DECLARE @CustomerID int;
    SELECT @CustomerID = SCOPE_IDENTITY();

    INSERT INTO [CustomerFirms]([CustomerID], [CityID],
    [PostalCode], [Address], [CompanyName], [NIP])
    VALUES (@CustomerID, @CityID, @PostalCode, @Address,
    @CompanyName, @NIP)
```

```
    END TRY
    BEGIN CATCH
    DELETE FROM [Customers]
      WHERE [CustomerID] = @CustomerID
    DELETE FROM [CustomerFirms]
      WHERE [CustomerID] = @CustomerID
    DECLARE @errorMsg nvarchar(2048) = 'Cannot add new Firm
    Customer. Error message: '
    +ERROR_MESSAGE();
    THROW 52000, @errorMsg, 1;
    END CATCH
END
GO
```

8. **AddFirmEmployee** – dodaje pracownika klienta firmowego

```
CREATE PROCEDURE [AddFirmEmployee]
    @FirstName varchar(50),
    @LastName varchar(50),
    @NIP nchar(10),
    @Phone nchar(9)
AS
BEGIN
    BEGIN TRY
    DECLARE @CustomerID int;
    DECLARE @FirmID int;
    IF NOT EXISTS
    (
       SELECT * FROM [Customers] WHERE [Phone] = @Phone
    )
    BEGIN
       INSERT INTO [Customers]([Phone])
       VALUES (@Phone)
       SELECT @CustomerID = SCOPE_IDENTITY()
       INSERT INTO [CustomerIndividuals]([CustomerID], [FirstName],
      [LastName])
       VALUES (@CustomerID, @FirstName, @LastName)
    END
    ELSE
    BEGIN
       SET @CustomerID = (
             SELECT [CustomerID] FROM [Customers]
             WHERE [Phone] = @Phone
             );
    END

    SET @FirmID = (
             SELECT [CustomerID] FROM [CustomerFirms]
             WHERE [NIP] = @NIP
             );

    IF NOT EXISTS
             (
             SELECT [CustomerID] FROM [CustomerFirms]
             WHERE [NIP] = @NIP
             )
```

```
            BEGIN
                  ;THROW 52000, 'Firm does not exist.', 1
            END

    INSERT INTO [CustomerFirmsEmployees]([CustomerID], [FirmID])
    VALUES (@CustomerID, @FirmID)

    END TRY
    BEGIN CATCH
    IF NOT EXISTS
    (
      SELECT * FROM [Customers] WHERE [Phone] = @Phone
    )
    BEGIN
      DELETE FROM [Customers]
            WHERE [CustomerID] = @CustomerID
      DELETE FROM [CustomerIndividuals]
            WHERE [CustomerID] = @CustomerID
    END
    DECLARE @errorMsg nvarchar(2048) = 'Cannot add new Firm
    Employee. Error message: '
    +ERROR_MESSAGE();
    THROW 52000, @errorMsg, 1;
    END CATCH
END
GO
```

9. **AddIndividualCustomer** – dodaje klienta indywidualnego

```
CREATE PROCEDURE [AddIndividualCustomer]
    @FirstName varchar(50),
    @LastName varchar(50),
    @Phone nchar(9)
AS
BEGIN
    BEGIN TRY

    INSERT INTO [Customers]([Phone])
    VALUES (@Phone)
    DECLARE @CustomerID int;
    SELECT @CustomerID = SCOPE_IDENTITY();
    INSERT INTO [CustomerIndividuals]([CustomerID], [FirstName],
    [LastName])
    VALUES (@CustomerID, @FirstName, @LastName)

    END TRY
    BEGIN CATCH
    DELETE FROM [Customers]
      WHERE [CustomerID] = @CustomerID
    DELETE FROM [CustomerIndividuals]
      WHERE [CustomerID] = @CustomerID
    DECLARE @errorMsg nvarchar(2048)= 'Cannot add new Individual
    Customer. Error message: '
    +ERROR_MESSAGE();
    THROW 52000, @errorMsg, 1;
    END CATCH
END
GO
```

10. **AddOrder** – dodaje nowe zamówienie

```
CREATE PROCEDURE [AddOrder]
    @Phone nchar(9),
    @EmployeeID int = NULL,
    @PaymentName varchar(50),
    @ReceiveDate datetime
AS
BEGIN
    BEGIN TRY
      DECLARE @OrderID int;
      DECLARE @CustomerID int;
      DECLARE @PaymentTypeID int;

      IF NOT EXISTS
      (
            SELECT [CustomerID] FROM [Customers]
            WHERE [Phone] = @Phone
      )
      BEGIN
            ;THROW 52000, 'Customer does not exist.', 1
      END

      SET @CustomerID = (
            SELECT [CustomerID] FROM [Customers]
            WHERE [Phone] = @Phone
      );


      IF NOT EXISTS
      (
            SELECT [PaymentTypeID] FROM [PaymentType]
            WHERE [PaymentName] = @PaymentName
      )
      BEGIN
            ;THROW 52000, 'Payment type does not exist.', 1
      END

      SET @PaymentTypeID = (
            SELECT [PaymentTypeID] FROM [PaymentType]
            WHERE [PaymentName] = @PaymentName
      );
```

```
      IF
      (
      @EmployeeID IS NOT NULL
      )
      BEGIN
            IF NOT EXISTS
            (
                  SELECT [EmployeeID] FROM [Employees]
                  WHERE [EmployeeID] = @EmployeeID
            )
            BEGIN
                  ;THROW 52000, 'Employee does not exist.', 1
            END
      END

      IF
      (
            DATEDIFF(DAY, GETDATE(), @ReceiveDate) < 0
      )
      BEGIN
            ;THROW 52000, 'Receive date can not be before order
            date', 1
      END

      INSERT INTO [Orders]([CustomerID], [EmployeeID],
      [OrderDate], [ReceiveDate], [PaymentTypeID])
      VALUES (@CustomerID, @EmployeeID, GETDATE(), @ReceiveDate,
      @PaymentTypeID)
      SELECT @OrderID = SCOPE_IDENTITY()
    END TRY
    BEGIN CATCH
    DELETE FROM [Orders]
      WHERE [OrderID] = @OrderID
    DECLARE @errorMsg nvarchar(2048)= 'Cannot add new order. Error
    message: '
    +ERROR_MESSAGE();
    THROW 52000, @errorMsg, 1;
    END CATCH
    SELECT @OrderID AS 'OrderID'
END
GO
```

11. **AddPaymentType** – dodaje nowy typ płatności

```
CREATE PROCEDURE [AddPaymentType]
    @PaymentName varchar(50)
AS
BEGIN
    BEGIN TRY
      INSERT INTO [PaymentType]([PaymentName])
      VALUES (@PaymentName)
      DECLARE  @PaymentTypeID int
      SELECT @PaymentTypeID = SCOPE_IDENTITY()
    END TRY
    BEGIN CATCH
    DELETE FROM [PaymentType]
      WHERE [PaymentTypeID] = @PaymentTypeID

    DECLARE @errorMsg nvarchar(2048) = 'Cannot add new Payment
    Type. Error message: '
    +ERROR_MESSAGE();
    THROW 52000, @errorMsg, 1;
    END CATCH
END
GO
```

12. **AddReservationForFirm** – dodaje rezerwację dla klienta firmowego

```
CREATE PROCEDURE [AddReservationForFirm]
    @FirmID int,
    @ReservationDate date
AS
BEGIN
    BEGIN TRY

      IF NOT EXISTS
      (
            (SELECT [CustomerID] FROM [CustomerFirms]
            WHERE [CustomerID] = @FirmID)
      )
      BEGIN
            ;THROW 52000, 'Firm does not exist.', 1
      END

      IF
      (
      DATEDIFF(DAY, GETDATE(), @ReservationDate) <= 0
      )
      BEGIN
            ;THROW 52000, 'Invalid reservation date.', 1
      END

      INSERT INTO [Reservations]([ReservationDate])
      VALUES (@ReservationDate)
      DECLARE  @ReservationID int
      SELECT @ReservationID = SCOPE_IDENTITY()

      INSERT INTO [ReservationsFirms]([ReservationID], [FirmID])
      VALUES (@ReservationID, @FirmID)

    END TRY
    BEGIN CATCH

    DELETE FROM [Reservations]
      WHERE [ReservationID] = @ReservationID
    DELETE FROM [ReservationsFirms]
      WHERE [ReservationID] = @ReservationID
```

```
    DECLARE @errorMsg nvarchar(2048) = 'Cannot add reservation.
    Error message: '
    +ERROR_MESSAGE();
    THROW 52000, @errorMsg, 1;
    END CATCH
END
GO
```

13. **AddReservationForFirmAnonymous** – dodaje rezerwację dla klienta firmowego bez wskazywania pracowników

```
CREATE PROCEDURE [AddReservationForFirmAnonymous]
    @FirmID int,
    @ReservationID int,
    @PeopleCount int
AS
BEGIN
    BEGIN TRY

      IF NOT EXISTS
      (
            (SELECT [CustomerID] FROM [CustomerFirms]
            WHERE [CustomerID] = @FirmID)
      )
      BEGIN
            ;THROW 52000, 'Firm does not exist.', 1
      END

      IF NOT EXISTS
      (
            (SELECT [FirmID] FROM [ReservationsFirms]
            WHERE [FirmID] = @FirmID)
      )
      BEGIN
            ;THROW 52000, 'Reservation does not exist.', 1
      END

      INSERT INTO [ReservationsFirmsDetails]([ReservationID],
      [PeopleCount])
      VALUES (@ReservationID, @PeopleCount)

    END TRY
    BEGIN CATCH

    DELETE FROM [ReservationsFirmsDetails]
      WHERE [ReservationID] = @ReservationID
    DECLARE @errorMsg nvarchar(2048) = 'Cannot add reservation.
    Error message: '
```

```
    +ERROR_MESSAGE();
    THROW 52000, @errorMsg, 1;
    END CATCH
END
GO
```

14. **AddReservationForFirmEmployee** – dodaje pracownika do rezerwacji dla klienta firmowego

```
CREATE PROCEDURE [AddReservationForFirmEmployee]
    @FirmID int,
    @PeopleCount int,
    @Phone nchar(9),
    @ReservationID int
AS
BEGIN
    BEGIN TRY
      DECLARE @CustomerID int;


      IF NOT EXISTS
      (
            (SELECT [CustomerID] FROM [CustomerFirms]
            WHERE [CustomerID] = @FirmID)
      )
      BEGIN
            ;THROW 52000, 'Firm does not exist.', 1
      END

      IF NOT EXISTS
      (
            (SELECT [CustomerID] FROM [Customers]
            WHERE [Phone] = @Phone)
      )
      BEGIN
            ;THROW 52000, 'Firm employee does not exist.', 1
      END

      SET @CustomerID =
      (
            SELECT [CustomerID] FROM [Customers]
            WHERE [Phone] = @Phone
      );

      IF NOT EXISTS
      (
            (SELECT [CustomerID] FROM [CustomerIndividuals]
            WHERE [CustomerID] = @CustomerID)
```

```
        )
        BEGIN
              ;THROW 52000, 'Firm employee does not exist.', 1
        END

        IF NOT EXISTS
        (
              (SELECT [CustomerID] FROM [CustomerFirmsEmployees]
               WHERE [CustomerID] = @CustomerID
               AND [FirmID] = @FirmID)
        )
        BEGIN
              ;THROW 52000, 'This person is not employee of this
              Firm', 1
        END



        INSERT INTO [ReservationsFirmsEmployees]([ReservationID],
        [EmployeeID], [PeopleCount])
        VALUES (@ReservationID, @CustomerID, @PeopleCount)

    END TRY
    BEGIN CATCH

    DELETE FROM [ReservationsFirmsEmployees]
       WHERE [ReservationID] = @ReservationID
      AND [EmployeeID] = @CustomerID
    DECLARE @errorMsg nvarchar(2048) = 'Cannot add reservation.
    Error message: '
    +ERROR_MESSAGE();
    THROW 52000, @errorMsg, 1;
    END CATCH
END
GO
```

15. **AddReservationForIndividual** – dodaje rezerwację dla klienta indywidualnego

```
CREATE PROCEDURE [AddReservationForIndividual]
    @CustomerID int,
    @OrderID int,
    @PeopleCount int,
    @ReservationDate date
AS
BEGIN
    BEGIN TRY

      IF NOT EXISTS
      (
            (SELECT [OrderID] FROM [Orders]
            WHERE [OrderID] = @OrderID)
      )
      BEGIN
            ;THROW 52000, 'Order does not exist.', 1
      END

      IF NOT EXISTS
      (
            (SELECT [CustomerID] FROM [CustomerIndividuals]
            WHERE [CustomerID] = @CustomerID)
      )
      BEGIN
            ;THROW 52000, 'Customer does not exist.', 1
      END

      IF
      (
      DATEDIFF(DAY, GETDATE(), @ReservationDate) <= 0
      )
      BEGIN
            ;THROW 52000, 'Invalid reservaton date.', 1
      END

      IF
      (
            (SELECT SUM([DishPrice] * [Quantity])
```

```
            FROM [OrderDetails] AS OD
            WHERE OD.[OrderID] = @OrderID GROUP BY OD.[OrderID])
            <
            (SELECT [WZValue] FROM [ReservationRequirements])
      )
      BEGIN
            ;THROW 52000, 'Value of order is to small.', 1
      END

      IF
      (
      (SELECT COUNT(*) FROM [Orders]
       WHERE [OrderID] = @OrderID GROUP BY [OrderID])
            <
      (SELECT [WKValue] FROM [ReservationRequirements])
      )
      BEGIN
            ;THROW 52000, 'Number of orders is to small.', 1
      END

      INSERT INTO [Reservations](ReservationDate)
      VALUES (@ReservationDate)
      DECLARE  @ReservationID int
      SELECT @ReservationID = SCOPE_IDENTITY()

      INSERT INTO [ReservationsIndividuals]([ReservationID],
     [CustomerID], [OrderID], [PeopleCount])
      VALUES (@ReservationID, @CustomerID, @OrderID, @PeopleCount)

   END TRY
   BEGIN CATCH
   DELETE FROM [ReservationsIndividuals]
     WHERE [ReservationID] = @ReservationID
   DELETE FROM [Reservations]
     WHERE [ReservationID] = @ReservationID
   DECLARE @errorMsg nvarchar(2048) = 'Cannot add reservation.
   Error message: '
   +ERROR_MESSAGE();
   THROW 52000, @errorMsg, 1;
   END CATCH
END
GO
```

16. **AddRestaurantEmployee** – dodaje pracownika restauracji

```
CREATE PROCEDURE [AddRestaurantEmployee]
    @FirstName varchar(50),
    @LastName varchar(50),
    @ManagerID int = NULL,
    @Phone nchar(9),
    @CityName varchar(50),
    @PostalCode varchar(50),
    @Address varchar(50)
AS
BEGIN
    BEGIN TRY
      DECLARE @CityID int;

      IF NOT EXISTS
            (
            SELECT [CityID] FROM [Cities]
            WHERE [CityName] = @CityName
            )
            BEGIN
                  ;THROW 52000, 'City does not exist.', 1
            END

      SET @CityID = (
      SELECT [CityID] FROM [Cities] WHERE [CityName] = @CityName
      );
      INSERT INTO [Employees]([FirstName], [LastName],
      [ManagerID], [Phone], [CityID], [PostalCode], [Address])
      VALUES (@FirstName, @LastName, @ManagerID, @Phone, @CityID,
      @PostalCode, @Address)
      DECLARE  @EmployeeID int
      SELECT @EmployeeID = SCOPE_IDENTITY()
    END TRY
    BEGIN CATCH
    DELETE FROM [Employees]
      WHERE [EmployeeID] = @EmployeeID

    DECLARE @errorMsg nvarchar(2048) = 'Cannot add new Firm
    Employee. Error message: '
    +ERROR_MESSAGE();
    THROW 52000, @errorMsg, 1;
```

```
    END CATCH
END
GO
```

17. **AddTable** – dodaje nowy stolik

```
CREATE PROCEDURE [AddTable]
    @Places int
AS
BEGIN
    BEGIN TRY
           INSERT INTO [Tables]([Places])
           VALUES (@Places)
           DECLARE @TableID int
           SELECT @TableID = SCOPE_IDENTITY()
    END TRY
    BEGIN CATCH
           DELETE FROM [Tables] WHERE [TableID] = @TableID
           DECLARE @errorMsg nvarchar(2048) = 'Cannot add new
Table. Error message: '
           +ERROR_MESSAGE();
           THROW 52000, @errorMsg, 1;
    END CATCH
END
GO
```

18. **AddValueForDiscount** – dodaje nową wartość do zniżki

```
CREATE PROCEDURE [AddValueForDiscount]
    @DiscountID int,
    @Type varchar(50),
    @Value int
AS
BEGIN
    DECLARE @SetID INT;
    BEGIN TRY
       IF
       (
            (@Value <= 0)
       )
       BEGIN
            ;THROW 52000, 'Value must be greater than 0.', 1
       END

       IF NOT EXISTS
       (
            (SELECT [SetID] FROM [DiscountsSet]
            WHERE [SetName] = @Type)
       )
       BEGIN
            INSERT INTO [DiscountsSet]([SetName])
            VALUES (@Type)
            SELECT @SetID = SCOPE_IDENTITY()
       END
       ELSE
       BEGIN
            SET @SetID=
            (
                 (SELECT [SetID] FROM [DiscountsSet]
                  WHERE [SetName] = @Type)
            )
       END

       INSERT INTO [DiscountSetDetails]([DiscountID], [SetID],
      [Value])
       VALUES (@DiscountID, @SetID, @Value)
    END TRY
    BEGIN CATCH
```

```
    DELETE FROM [DiscountSetDetails]
      WHERE [DiscountID] = @DiscountID AND [SetID] = @SetID
    DECLARE @errorMsg nvarchar(2048) = 'Cannot add new discount
    value. Error message: '
    +ERROR_MESSAGE();
    THROW 52000, @errorMsg, 1;
    END CATCH
END
GO
```

19. **ChangeMenu** – dokonuje aktualizacji menu na kolejny dzień

```
CREATE PROCEDURE [ChangeMenu]
AS
BEGIN
      BEGIN TRY
      DECLARE @InMenu int
      DECLARE @AllDishes int
      DECLARE @Counter int
      DECLARE @NewDish int
      SET @InMenu =
      (
            SELECT COUNT(*) FROM [DishesHistory] AS DH
            INNER JOIN [Dishes] AS D ON DH.[DishID] = D.[DishID]
            INNER JOIN [Categories] AS C
            ON D.[CategoryID] = C.[CategoryID]
            WHERE [OutMenuDate] IS NULL AND
            C.[CategoryName] != 'Owoce morza'
      )
      SET @AllDishes =
            (
                  SELECT COUNT(*) FROM (
                        SELECT ROW_NUMBER() OVER ( ORDER BY
Dish.[DishID]) AS RowDish, Dish.[DishID] FROM
                        (
                        SELECT DISTINCT D.[DishID], ROW_NUMBER()
OVER (PARTITION BY D.[DishID] ORDER BY D.[DishID]) AS Row
                        FROM [Dishes] as D
                        INNER JOIN [Categories] AS C
                        ON C.[CategoryID] = D.[CategoryID]
                        LEFT JOIN [DishesHistory] AS DH
                        ON DH.[DishID] = D.[DishID]
                        WHERE C.[CategoryName] != 'Owoce morza'
                        AND (([OutMenuDate] IS NOT NULL
                        AND DATEDIFF(DAY, GETDATE(), [OutMenuDate])
                        < 1) OR DH.[DishesHistoryID] IS NULL)
                        AND D.[DishID] NOT IN
                        (SELECT D1.[DishID] FROM [Dishes] as D1
                        INNER JOIN [DishesHistory] AS DH1
                        ON DH1.[DishID] = D1.[DishID]
                        WHERE DH1.[OutMenuDate] IS NULL OR
```

```
                    DATEDIFF(DAY, GETDATE(), DH1.[OutMenuDate])
                    >= 1)
                    )
                    [Dish] WHERE Row = 1
                    )
                    FinallDish
      )

IF(@AllDishes = 0)
BEGIN
      ;THROW 52000, 'There is no dishes to add.', 1
END



SET @Counter = (
      CEILING(@InMenu / 2)
)

IF(@Counter > @AllDishes)
BEGIN
SET @Counter = @AllDishes
END

DECLARE @MenuCursor CURSOR;
      DECLARE @DishIDInMenu int;
      SET @MenuCursor = CURSOR FOR
            SELECT TOP 100 PERCENT [DishesHistoryID] FROM
            [DishesHistory] AS DH
            INNER JOIN [Dishes] AS D
            ON D.[DishID] = DH.[DishID]
            INNER JOIN [Categories] AS C ON
            C.[CategoryID] = D.[CategoryID]
            WHERE [OutMenuDate] IS NULL
            AND C.[CategoryName] != 'Owoce morza'
            AND DATEDIFF(DAY, GETDATE(), DH.[InMenuDate]) < 0
            ORDER BY [InMenuDate]

      BEGIN
            OPEN @MenuCursor
            FETCH NEXT FROM @MenuCursor
            INTO @DishIDInMenu
```

```
        WHILE @@FETCH_STATUS = 0
        BEGIN
              IF(@Counter > 0)
                    BEGIN
                    UPDATE [DishesHistory]
                    SET [OutMenuDate] = DATEADD(DAY, 1,
                    GETDATE()) WHERE
                    [DishesHistoryID] = @DishIDInMenu
                    SET @Counter = (@Counter - 1)
                    END
              FETCH NEXT FROM @MenuCursor
              INTO @DishIDInMenu
        END
        CLOSE @MenuCursor
        DEALLOCATE @MenuCursor
END

SET @Counter = (
      CEILING(@InMenu / 2)
)


SET @AllDishes =
(
      SELECT COUNT(*) FROM (
            SELECT ROW_NUMBER() OVER (
            ORDER BY Dish.[DishID]) AS
            RowDish, Dish.[DishID] FROM
            (
                  SELECT DISTINCT D.[DishID],
                  ROW_NUMBER() OVER (PARTITION BY
                  D.[DishID] ORDER BY D.[DishID]) AS Row
                  FROM [Dishes] AS D
                  INNER JOIN [Categories] AS C
                  ON C.[CategoryID] = D.[CategoryID]
                  LEFT JOIN [DishesHistory] AS DH
                  ON DH.[DishID] = D.[DishID]
                  WHERE C.[CategoryName] != 'Owoce morza'
                  AND (([OutMenuDate] IS NOT NULL
                  AND DATEDIFF(DAY, GETDATE(),
                  [OutMenuDate]) < 1)
                  OR DH.[DishesHistoryID] IS NULL)
```

```
                    AND D.[DishID] NOT IN
                    (SELECT D1.[DishID] FROM [Dishes] AS D1
                    INNER JOIN [DishesHistory] AS DH1
                    ON DH1.[DishID] = D1.[DishID]
                    WHERE DH1.[OutMenuDate] IS NULL
                    OR DATEDIFF(DAY, GETDATE(),
                    DH1.[OutMenuDate]) >= 1)
               )
               Dish WHERE Row = 1
               )
          FinallDish
)

IF(@AllDishes = 0)
BEGIN
     ;THROW 52000, 'There is no dishes to add.', 1
END

WHILE @Counter > 0 AND @AllDishes > 0
     BEGIN

          SET @NewDish = (
          SELECT FinallDish.[DishID] FROM (
               SELECT ROW_NUMBER() OVER (
                ORDER BY Dish.[DishID]) AS
                RowDish, Dish.[DishID] FROM
               (
               SELECT DISTINCT D.[DishID],
               ROW_NUMBER() OVER (PARTITION BY
               D.[DishID] ORDER BY D.[DishID]) AS Row
               FROM [Dishes] as D
               INNER JOIN [Categories] AS C
               ON C.[CategoryID] = D.[CategoryID]
               LEFT JOIN [DishesHistory] AS DH
               ON DH.[DishID] = D.[DishID]
               WHERE C.[CategoryName] != 'Owoce morza'
               AND (([OutMenuDate] IS NOT NULL
               AND DATEDIFF(DAY, GETDATE(),
               [OutMenuDate]) < 1)
               OR DH.[DishesHistoryID] IS NULL)
               AND D.[DishID] NOT IN
               (SELECT D1.[DishID] FROM [Dishes] AS D1
```

```
                        INNER JOIN [DishesHistory] AS DH1
                        ON DH1.[DishID] = D1.[DishID]
                        WHERE DH1.[OutMenuDate] IS NULL
                        OR DATEDIFF(DAY, GETDATE(),
                        DH1.[OutMenuDate]) >= 1)
                        )
                        Dish WHERE Row = 1
                        )
                        FinallDish WHERE RowDish =
                        FLOOR(RAND() * (@AllDishes) + 1)
                    )


                INSERT INTO  [DishesHistory]([DishPrice],
                [InMenuDate], [UnitsInStock], [DishID])
                VALUES (
                (SELECT [BasicDishPrice] FROM [Dishes]
                 WHERE [DishID] = @NewDish),
                  DATEADD(DAY, 1, GETDATE()),
                (SELECT [StartUnits] FROM [Dishes]
                WHERE [DishID] = @NewDish),
                @NewDish
                )

                SET @AllDishes = (@AllDishes - 1)
                SET @Counter = (@Counter - 1)

            END

    END TRY
    BEGIN CATCH

    DECLARE @errorMsg nvarchar(2048) = 'Cannot update menu. Error
    message: '
    +ERROR_MESSAGE();
    THROW 52000, @errorMsg, 1;
    END CATCH
END
GO
```

20. **ChangeUnitsInStockValueForDish** – zmienia ilość sztuk dania na stanie

```
CREATE PROCEDURE [ChangeUnitsInStockValueForDish]
    @DishName varchar(50),
    @UnitsInStock int
AS
BEGIN
    BEGIN TRY
    DECLARE @DishID int;
    DECLARE @DishesHistoryID int;

      IF NOT EXISTS
      (
            SELECT [DishID] FROM [Dishes]
            WHERE [DishName] = @DishName
      )
      BEGIN
            ;THROW 52000, 'Dish does not exist.', 1
      END

      SET @DishID = (
            SELECT [DishID] FROM [Dishes]
            WHERE [DishName] = @DishName
      );

      IF NOT EXISTS
      (
            SELECT [DishesHistoryID] FROM [DishesHistory]
            WHERE [DishID] = @DishID AND [OutMenuDate] IS NULL
      )
      BEGIN
            ;THROW 52000, 'Dish is not in Menu.', 1
      END

      SET @DishesHistoryID = (
            SELECT [DishesHistoryID] FROM [DishesHistory]
            WHERE [DishID] = @DishID AND [OutMenuDate] IS NULL
      );


      IF NOT EXISTS
```

```
      (
            SELECT [DishID] FROM [Dishes]
            WHERE [DishName] = @DishName
            AND @UnitsInStock >= [MinStockValue]
      )
      BEGIN
            ;THROW 52000, 'New value is smaller than minimal value
            for this dish.', 1
      END

      UPDATE [DishesHistory] SET [UnitsInStock] = @UnitsInStock
     WHERE [DishesHistoryID] = @DishesHistoryID

   END TRY
   BEGIN CATCH
   DECLARE @errorMsg nvarchar(2048) = 'Cannot change units in
   stock value. Error message: '
   +ERROR_MESSAGE();
   THROW 52000, @errorMsg, 1;
   END CATCH
END
GO
```

21. **CheckMenu** – Sprawdza czy obecne menu spełnia wymagania co do aktualności dań

```
CREATE PROCEDURE [CheckMenu]
AS
BEGIN
      BEGIN TRY
      DECLARE @InMenu int
      DECLARE @Valid int
      SET @InMenu =
      (
            SELECT COUNT(*) FROM [DishesHistory] AS DH
            INNER JOIN [Dishes] AS D ON DH.[DishID] = D.[DishID]
            INNER JOIN [Categories] AS C
            ON D.[CategoryID] = C.[CategoryID]
            WHERE [OutMenuDate] IS NULL
            AND C.[CategoryName] != 'Owoce morza'
      )
      SET @Valid =
      (
            SELECT COUNT(*) FROM [DishesHistory] AS DH
            INNER JOIN [Dishes] AS D ON DH.[DishID] = D.[DishID]
            INNER JOIN [Categories] AS C
            ON D.[CategoryID] = C.[CategoryID]
            WHERE [OutMenuDate] IS NULL
            AND C.[CategoryName] != 'Owoce morza'
            AND DATEDIFF(DAY, [InMenuDate], GETDATE()) < 14
      )
      IF 2 * @Valid > @InMenu
      BEGIN
            ;THROW 52000, 'More than half of dishes in menu have
            been there for longer than two weeks', 1
      END
      END TRY
      BEGIN CATCH
      DECLARE @errorMsg nvarchar(2048) = 'Menu is invalid. Error
     message: '
      +ERROR_MESSAGE();
      THROW 52000, @errorMsg, 1;
      END CATCH
END
```

22. **ConfirmReservation** – potwierdza rezerwację i przydziela stolik

```
CREATE PROCEDURE [ConfirmReservation]
    @EmployeeID int,
    @ReservationID int
AS
BEGIN
    BEGIN TRY
      DECLARE @PeopleCount int;
      DECLARE @ReservationDate date;
      DECLARE @TableID int;

      IF NOT EXISTS
      (
            (SELECT [ReservationID] FROM [Reservations] WHERE
[ReservationID] = @ReservationID)
      )
      BEGIN
            ;THROW 52000, 'Reservation does not exist.', 1
      END

      SET @ReservationDate =
      (
            (SELECT [ReservationDate] FROM [Reservations]
            WHERE [ReservationID] = @ReservationID)
      );


      IF EXISTS
      (
            (SELECT [ReservationID] FROM [ReservationsIndividuals]
            WHERE [ReservationID] = @ReservationID)
      )
      BEGIN
            SET @PeopleCount =
            (
                  (SELECT [PeopleCount] FROM
                  [ReservationsIndividuals]
                  WHERE [ReservationID] = @ReservationID)
            )
```

```
        IF NOT EXISTS
        (
              SELECT * FROM [FreeTables](@ReservationDate)
              WHERE [Places] >= @PeopleCount
        )
        BEGIN
              ;THROW 52000, 'There is not free table.', 1
        END

        SET @TableID =
        (
              SELECT TOP 1 [TableID] FROM
              [FreeTables](@ReservationDate)
              WHERE [Places] >= @PeopleCount ORDER BY [Places]
        )
        UPDATE [ReservationsIndividuals]
        SET [TableID] = @TableID
        WHERE [ReservationID] = @ReservationID

  END

  IF EXISTS
  (
        (SELECT [ReservationID] FROM [ReservationsFirms]
        WHERE [ReservationID] = @ReservationID)
  )
  BEGIN


        DECLARE @RFDIDCursor CURSOR;
        DECLARE @RFDID int;
        SET @RFDIDCursor = CURSOR FOR (SELECT [RFDID]
        FROM [ReservationsFirmsDetails]
        WHERE [ReservationID] = @ReservationID)

        BEGIN
              OPEN @RFDIDCursor
              FETCH NEXT FROM @RFDIDCursor
              INTO @RFDID
              WHILE @@FETCH_STATUS = 0
              BEGIN
```

```
                SET @PeopleCount =
                (
                      (SELECT [PeopleCount]
                      FROM [ReservationsFirmsDetails]
                      WHERE [RFDID] = @RFDID)
                )
                IF NOT EXISTS
                (
                      SELECT * FROM
                      [FreeTables](@ReservationDate)
                      WHERE [Places] >= @PeopleCount
                )
                BEGIN
                      ;THROW 52000, 'There is not free
                       table.', 1
                END

                SET @TableID =
                (
                      SELECT TOP 1 [TableID]
                       FROM [FreeTables](@ReservationDate)
                       WHERE [Places] >= @PeopleCount
                       ORDER BY [Places]
                )

                UPDATE [ReservationsFirmsDetails]
                SET [TableID] = @TableID
                WHERE [RFDID] = @RFDID

                FETCH NEXT FROM @RFDIDCursor
                INTO @RFDID

          END
          CLOSE @RFDIDCursor
          DEALLOCATE @RFDIDCursor
    END

    DECLARE @FirmEmployeeIDCursor CURSOR;
    DECLARE @FirmEmployeeID int;
    SET @FirmEmployeeIDCursor = CURSOR FOR
    (SELECT [EmployeeID] FROM [ReservationsFirmsEmployees]
    WHERE [ReservationID] = @ReservationID)
```

```
BEGIN
      OPEN @FirmEmployeeIDCursor
      FETCH NEXT FROM @FirmEmployeeIDCursor
      INTO @FirmEmployeeID
      WHILE @@FETCH_STATUS = 0
      BEGIN

            SET @PeopleCount =
            (
                  (SELECT [PeopleCount]
                  FROM [ReservationsFirmsEmployees]
                  WHERE [ReservationID] = @ReservationID
                  AND [EmployeeID] = @FirmEmployeeID)
            )

            IF NOT EXISTS
            (
                  SELECT * FROM
                  [FreeTables](@ReservationDate)
                  WHERE [Places] >= @PeopleCount
            )
            BEGIN
                  ;THROW 52000, 'There is not free
                   table.', 1
            END

            SET @TableID =
            (
                  SELECT TOP 1 [TableID]
                  FROM [FreeTables](@ReservationDate)
                  WHERE [Places] >= @PeopleCount
                  ORDER BY [Places]
            )

            UPDATE [ReservationsFirmsEmployees] SET
            [TableID] = @TableID
            WHERE [ReservationID] = @ReservationID
            AND [EmployeeID] = @FirmEmployeeID

            FETCH NEXT FROM @FirmEmployeeIDCursor
            INTO @FirmEmployeeID
```

```
                END
                CLOSE @FirmEmployeeIDCursor
                DEALLOCATE @FirmEmployeeIDCursor
        END
    END


UPDATE [Reservations] SET [EmployeeID] = @EmployeeID
WHERE [ReservationID] = @ReservationID
END TRY
BEGIN CATCH

        IF EXISTS
        (
              (SELECT [ReservationID]
               FROM [ReservationsIndividuals]
               WHERE [ReservationID] = @ReservationID)
        )
        BEGIN
              UPDATE [ReservationsIndividuals]
             SET [TableID] = NULL
             WHERE [ReservationID] = @ReservationID
        END

        SET @RFDIDCursor = CURSOR FOR (SELECT [RFDID]
        FROM [ReservationsFirmsDetails]
        WHERE [ReservationID] = @ReservationID)
        BEGIN
              OPEN @FirmEmployeeIDCursor
              FETCH NEXT FROM @FirmEmployeeIDCursor
              INTO @FirmEmployeeID
              WHILE @@FETCH_STATUS = 0
              BEGIN

                    UPDATE [ReservationsFirmsEmployees]
                    SET [TableID] = NULL
                    WHERE [ReservationID] = @ReservationID
                    AND [EmployeeID] = @FirmEmployeeID

                    FETCH NEXT FROM @FirmEmployeeIDCursor
                    INTO @FirmEmployeeID
```

```
                END
                CLOSE @FirmEmployeeIDCursor
                DEALLOCATE @FirmEmployeeIDCursor
            END

            SET @FirmEmployeeIDCursor = CURSOR FOR
            (SELECT [EmployeeID] FROM [ReservationsFirmsEmployees]
            WHERE [ReservationID] = @ReservationID)
            BEGIN
                OPEN @RFDIDCursor
                FETCH NEXT FROM @RFDIDCursor
                INTO @RFDID
                WHILE @@FETCH_STATUS = 0
                BEGIN

                    UPDATE [ReservationsFirmsDetails]
                    SET [TableID] = NULL WHERE [RFDID] = @RFDID

                    FETCH NEXT FROM @RFDIDCursor
                    INTO @RFDID

                END
                CLOSE @RFDIDCursor
                DEALLOCATE @RFDIDCursor
            END

            UPDATE [Reservations] SET [EmployeeID] = NULL
            WHERE [ReservationID] = @ReservationID

    DECLARE @errorMsg nvarchar(2048) = 'Cannot add confirm
    reservation. Error message: '
    +ERROR_MESSAGE();
    THROW 52000, @errorMsg, 1;
    END CATCH
END
GO
```

23. **RemoveDishFromMenu** – usuwa danie z aktualnego menu

```
CREATE PROCEDURE [RemoveDishFromMenu]
    @DishName varchar(50)
AS
BEGIN
    BEGIN TRY
    DECLARE @DishID int;
    DECLARE @DishesHistoryID int;
       IF NOT EXISTS
       (
            SELECT [DishID] FROM [Dishes]
            WHERE [DishName] = @DishName
       )
       BEGIN
            ;THROW 52000, 'Dish does not exist.', 1
       END

       SET @DishID = (
            SELECT [DishID] FROM [Dishes]
            WHERE [DishName] = @DishName
       );

       IF NOT EXISTS
       (
            SELECT [DishID] FROM [DishesHistory]
            WHERE [DishID] = @DishID AND [OutMenuDate] IS NULL
       )
       BEGIN
            ;THROW 52000, 'This dish is not in menu.', 1
       END

       SET @DishesHistoryID = (
            SELECT [DishesHistoryID] FROM [DishesHistory]
            WHERE [DishID] = @DishID AND [OutMenuDate] IS NULL
       );

       UPDATE [DishesHistory] SET [OutMenuDate] = GETDATE()
      WHERE [DishesHistoryID] = @DishesHistoryID
    END TRY
    BEGIN CATCH
```

```
    DECLARE @errorMsg nvarchar(2048) = 'Cannot delete dish from
    Menu. Error message: '
    +ERROR_MESSAGE();
    THROW 52000, @errorMsg, 1;
    END CATCH
END
GO
```

24. **UpdateCategory**– aktualizuje informacje o kategorii dania

```
CREATE PROCEDURE [UpdateCategory]
    @CategoryName varchar(50),
    @NewCategoryName varchar(50),
    @Description varchar(500)
AS
BEGIN
    DECLARE @CategoryID int
    BEGIN TRY

    IF NOT EXISTS
    (
    (SELECT [CategoryID] FROM [Categories]
    WHERE [CategoryName] = @CategoryName)
    )
    BEGIN
       ;THROW 52000, 'Category does not exist.', 1
    END

    SET @CategoryID =
    (
       (SELECT [CategoryID] FROM [Categories]
        WHERE [CategoryName] = @CategoryName)
    )
    UPDATE [Categories] SET [CategoryName] = @NewCategoryName,
    [Description] = @Description WHERE [CategoryID] = @CategoryID

    END TRY
    BEGIN CATCH

    DECLARE @errorMsg nvarchar(2048) = 'Cannot update category.
Error message: '
    +ERROR_MESSAGE();
    THROW 52000, @errorMsg, 1;
    END CATCH
END
GO
```

25. **UpdateDish**– aktualizuje parametry i informacje o daniu

```
CREATE PROCEDURE [UpdateDish]
    @DishName varchar(50),
    @NewDishName varchar(50),
    @Description varchar(500),
    @MinStockValue int,
    @StartUnits int
AS
BEGIN
    DECLARE @DishID int
    BEGIN TRY

    IF NOT EXISTS
    (
    (SELECT [DishID] FROM [Dishes] WHERE [DishName] = @DishName)
    )
    BEGIN
       ;THROW 52000, 'Dish does not exist.', 1
    END

    SET @DishID =
    (
       (SELECT [[DishID] FROM [Dishes]
        WHERE [DishName] = @DishName)
    )
    UPDATE [Dishes] SET [DishName] = @NewDishName,
    [Description] = @Description,
    [MinStockValue] = @MinStockValue, [StartUnits] = @StartUnits
    WHERE [DishID] = @DishID

    END TRY
    BEGIN CATCH

    DECLARE @errorMsg nvarchar(2048) = 'Cannot update dish. Error
    message: '
    +ERROR_MESSAGE();
    THROW 52000, @errorMsg, 1;
    END CATCH
END
GO
```

26. **UpdateReservationRequirements**– aktualizuje wymagania dotyczące możliwości składania rezerwacji

```
CREATE PROCEDURE [UpdateReservationRequirements]
    @WZValue int,
    @WKValue int
AS
BEGIN
    BEGIN TRY

    IF
    (@WZValue) < 0 OR (@WKValue) < 0
    BEGIN
      ;THROW 52000, 'Value must be greater than 0.', 1
    END

    UPDATE [ReservationRequirements]
    SET [WKValue] = @WKValue, [WZValue] = @WZValue

    END TRY
    BEGIN CATCH

    DECLARE @errorMsg nvarchar(2048) = 'Cannot update reservation
    requirements. Error message: '
    +ERROR_MESSAGE();
    THROW 52000, @errorMsg, 1;
    END CATCH
END
GO
```

7. Funkcje zwracające tabele (Widoki parametryzowane)

1. **CustomerDiscountSecondTypeHistory** – historia otrzymanych zniżek drugiego typu dla danego klienta

```
CREATE FUNCTION [CustomerDiscountSecondTypeHistory]
(
      @CustomerID int
)
RETURNS TABLE
AS
      RETURN
      (
      SELECT DS.[SetName], DSD.[Value], CDST.[ReceivedDate],
     CDST.[UseDate]
      FROM [CustomerDiscountsST] AS CDST
      INNER JOIN [Discounts] AS D
     ON CDST.[DiscountID] = D.[DiscountID]
      INNER JOIN [DiscountSetDetails] AS DSD
      ON D.[DiscountID] = DSD.[DiscountID]
      INNER JOIN [DiscountsSet] AS DS
     ON DSD.[SetID] = DS.[SetID]
      WHERE CDST.[CustomerID] = @CustomerID
      )
GO
```

2. **CustomerOrderHistory** – historia zamówień konkretnego klienta

```
CREATE FUNCTION [CustomerOrderHistory]
(
      @CustomerID int
)
RETURNS TABLE
AS
      RETURN
      (
      SELECT *,
            (
            SELECT SUM(OD.[DishPrice] * OD.[Quantity])
            FROM [OrderDetails] AS OD
            WHERE OD.[OrderID] = O.[OrderID]
            GROUP BY OD.[OrderID]
            ) AS 'Income'
            FROM [Orders] AS O
      WHERE O.[CustomerID] = @CustomerID
      )
GO
```

3. **CustomerOrderHistoryDetails** – szczegółowa historia zamówień konkretnego klienta

```
CREATE FUNCTION [CustomerOrderHistoryDetails]
(
      @CustomerID int
)
RETURNS TABLE
AS
      RETURN
      (
      SELECT O.[OrderID], O.[OrderDate], D.[DishID], D.[DishName],
       OD.[Quantity], OD.[DishPrice], OD.[Quantity] *
       OD.[DishPrice] AS 'Total' FROM [Orders] AS O
      INNER JOIN [OrderDetails] AS OD ON O.[OrderID] = OD.[OrderID]
      INNER JOIN [DishesHistory] AS DH
     ON OD.[DishesHistoryID] = DH.[DishesHistoryID]
      INNER JOIN [Dishes] AS D ON DH.[DishID] = D.[DishID]
      WHERE OD.[OrderID] = O.[OrderID]
     AND O.[CustomerID] = @CustomerID
      GROUP BY OD.[OrderID], D.[DishID], D.[DishName],
     OD.[Quantity], OD.[DishPrice], O.[OrderID], O.[OrderDate]
      )
GO
```

4. **FreeTables** – wolne stoliki na dany dzień

```
CREATE FUNCTION [FreeTables]
(
	@Date date
)
RETURNS TABLE
AS
	RETURN
	(
	SELECT T.[TableID], T.[Places] FROM [Reservations] AS R
	INNER JOIN [ReservationsFirms] AS RF
	ON R.[ReservationID] = RF.[ReservationID]
	INNER JOIN [ReservationsFirmsEmployees] AS RFE
	ON RF.[ReservationID] = RFE.[ReservationID]
	RIGHT OUTER JOIN [Tables] AS T
	ON RFE.[TableID] = T.[TableID]
	AND DATEDIFF(DAY, @Date, R.[ReservationDate]) = 0
	WHERE R.[ReservationID] IS NULL
	INTERSECT
	SELECT T.[TableID], T.[Places] FROM [Reservations] AS R
	INNER JOIN [ReservationsFirms] AS RF
	ON R.[ReservationID] = RF.[ReservationID]
	INNER JOIN [ReservationsFirmsDetails] AS RFD
	ON RF.[ReservationID] = RFD.[ReservationID]
	RIGHT OUTER JOIN [Tables] AS T
	ON RFD.[TableID] = T.[TableID]
	AND DATEDIFF(DAY, @Date, R.[ReservationDate]) = 0
	WHERE R.[ReservationID] IS NULL
	INTERSECT
	SELECT T.[TableID], T.[Places] FROM [Reservations] AS R
	INNER JOIN [ReservationsIndividuals] AS RI
	ON R.[ReservationID] = RI.[ReservationID]
	RIGHT OUTER JOIN [Tables] AS T
	ON RI.[TableID] = T.[TableID]
	AND DATEDIFF(DAY, @Date, R.[ReservationDate]) = 0
	WHERE R.[ReservationID] IS NULL
	)
GO
```

5. **OrdersForDay** – zamówienia złożone na dany dzień

```
CREATE FUNCTION [OrdersForDay]
(
	@ReceiveDate date
)
RETURNS TABLE
AS
	RETURN
	(
	SELECT * FROM [Orders]
	WHERE DATEDIFF(DAY, @ReceiveDate, [ReceiveDate]) = 0
	)
GO
```

6. **ReservationsForDay** – rezerwacje złożone na dany dzień

```
CREATE FUNCTION [ReservationsForDay]
(
	@ReservationDate date
)
RETURNS TABLE
AS
	RETURN
	(
	SELECT * FROM [Reservations]
	WHERE DATEDIFF(DAY, @ReservationDate, [ReservationDate]) = 0
	)
GO
```

7. **ReservationsForFirmCustomer** – historia rezerwacji złożonych przez klienta firmowego

```
CREATE FUNCTION [ReservationsForFirmCustomer]
(
      @CustomerID int
)
RETURNS TABLE
AS
      RETURN
      (
      SELECT R.[ReservationID], R.[ReservationDate], NULL
     AS 'EmployeeID', RFD.[PeopleCount], RFD.[TableID]
      FROM [Reservations] AS R
      INNER JOIN [ReservationsFirms] AS RF
     ON R.[ReservationID] = RF.[ReservationID]
      INNER JOIN [ReservationsFirmsDetails] AS RFD
     ON RF.[ReservationID] = RFD.[ReservationID]
      WHERE RF.FirmID = @CustomerID

      UNION

      SELECT R.[ReservationID], R.[ReservationDate],
     RFE.[EmployeeID], RFE.[PeopleCount], RFE.[TableID]
      FROM [Reservations] AS R
      INNER JOIN [ReservationsFirms] AS RF
     ON R.[ReservationID] = RF.[ReservationID]
      INNER JOIN [ReservationsFirmsEmployees] AS RFE
     ON RF.[ReservationID] = RFE.[ReservationID]
      WHERE RF.FirmID = @CustomerID
      )
GO
```

8. **ReservationsForIndividualCustomer** – historia rezerwacji złożonych przez klienta indywidualnego

```
CREATE FUNCTION [ReservationsForIndividualCustomer]
(
      @CustomerID int
)
RETURNS TABLE
AS
      RETURN
      (
      SELECT R.[ReservationID], R.[ReservationDate],
     RI.[PeopleCount], RI.[TableID], RI.[OrderID]
      FROM [Reservations] AS R
      INNER JOIN [ReservationsIndividuals] AS RI
     ON R.[ReservationID] = RI.[ReservationID]
      WHERE RI.CustomerID = @CustomerID
      )
GO
```

8. Funkcje zwracające wartości skalarne

1. **CustomerDiscountForOrder** – wartość zniżki dla danego zamówienia

```
CREATE FUNCTION [CustomerDiscountForOrder]
(
	@OrderID int
)
RETURNS FLOAT
AS
	BEGIN
		DECLARE @SumTotal FLOAT
		DECLARE @SumDiscounted FLOAT
		SET @SumTotal =
		(SELECT SUM(DH.[DishPrice] * OD.[Quantity])
		FROM Orders AS O
			INNER JOIN [OrderDetails] AS OD
			ON O.[OrderID] = OD.[OrderID]
			INNER JOIN [DishesHistory] AS DH
			ON OD.[DishesHistoryID] = DH.[DishesHistoryID]
			WHERE O.[OrderID] = @OrderID)
		SET @SumDiscounted =
		(SELECT SUM(OD.[DishPrice] * OD.[Quantity])
		FROM [Orders] AS O
			INNER JOIN [OrderDetails] AS OD
			ON O.[OrderID] = OD.[OrderID]
			WHERE O.[OrderID] = @OrderID)
		RETURN (
			SELECT(@SumTotal - @SumDiscounted)
		)
	END
GO
```

2. **CustomerOrdersMinValueCountSince** – Ilość zrealizowanych zamówień o minimalnej wartości dla klienta od konkretnej daty

```
CREATE FUNCTION [CustomerOrdersMinValueCountSince]
(
      @CustomerID int,
      @ReceiveDate datetime = NULL,
      @Value int = 0
)
RETURNS int
AS
      BEGIN
      IF @ReceiveDate IS NULL
      SET @ReceiveDate = CAST('1753-1-1' AS DATETIME)
      RETURN
      (
      SELECT COUNT(*)
      FROM [Orders] AS O
      WHERE O.[CustomerID] = @CustomerID
     AND O.[ReceiveDate] > @ReceiveDate
      AND (SELECT SUM(OD.[DishPrice] * OD.[Quantity])
            FROM [OrderDetails] AS OD
           WHERE OD.[OrderID] = O.[OrderID]) > @Value
      )
END
GO
```

3. **CustomerOrdersValueSince** – wartość zrealizowanych zamówień dla klienta od konkretnej daty

```
CREATE FUNCTION [CustomerOrdersValueSince]
(
      @CustomerID int,
      @ReceiveDate datetime = NULL
)
RETURNS int
AS
      BEGIN
      IF @ReceiveDate IS NULL
      SET @ReceiveDate = CAST('1753-1-1' AS DATETIME)
      RETURN
      (
      SELECT SUM(OD.[DishPrice] * OD.[Quantity])
      FROM [Orders] AS O
      INNER JOIN [OrderDetails] AS OD ON O.[OrderID] = OD.[OrderID]
      WHERE O.[CustomerID] = @CustomerID
       AND O.[ReceiveDate] > @ReceiveDate
      )
END
GO
```

4. **NumberDiscountFTReceived** – ilość klientów, którzy otrzymali zniżkę pierwszego typu

```
CREATE FUNCTION [NumberDiscountFTReceived]
()
RETURNS INT
AS
      BEGIN
      RETURN ISNULL((
            SELECT COUNT(*) FROM [CustomerIndividuals] AS CI
            INNER JOIN [CustomerDiscountFT] AS CDFT
            ON CO.[CustomerID] = CDFT.[CustomerID]
      ), 0)

      END
GO
```

5. **NumberDiscountSTCurrent** – ilość klientów, którzy mają aktywną zniżkę drugiego typu

```
CREATE FUNCTION [NumberDiscountSTCurrent]
()
RETURNS INT
AS
      BEGIN
      RETURN ISNULL((
            SELECT COUNT(*) FROM [CustomerIndividuals] AS CI
            INNER JOIN [CustomerDiscountsST] AS CDST
            ON CI.[CustomerID] = CDST.[CustomerID]
            INNER JOIN [Discounts] AS D
            ON CDST.[DiscountID] = D.[DiscountID]
            INNER JOIN [DiscountSetDetails] AS DSD
            ON D.[DiscountID] = DSD.[DiscountID]
            INNER JOIN [DiscountsSe]t AS DS
            ON DSD.[SetID] = DS.[SetID]
            WHERE DS.[SetName] = 'D'
            AND DATEDIFF(DAY, CDSD.[ReceivedDate], GETDATE()) <=
            DSD.[Value]
      ),0)

      END
GO
```

6. **OrderCost** – Całkowita wartość zamówienia

```
CREATE FUNCTION [OrderCost]
(
      @OrderID int
)
RETURNS FLOAT
AS
      BEGIN
      RETURN
      (
      SELECT SUM(OD.[DishPrice] * OD.[Quantity])
      FROM [Orders] AS O
      INNER JOIN [OrderDetails] AS OD ON O.[OrderID] = OD.[OrderID]
      WHERE O.[OrderID] = @OrderID
      )
END
GO
```

7. **OrderCostNoDiscount** – Całkowita wartość zamówienia bez uwzględniania zniżek

```
CREATE FUNCTION [OrderCostNoDiscount]
(
      @OrderID int
)
RETURNS FLOAT
AS
      BEGIN
      RETURN
      (
      SELECT SUM(DH.[DishPrice] * OD.[Quantity])
      FROM [Orders] AS O
      INNER JOIN [OrderDetails] AS OD ON O.[OrderID] = OD.[OrderID]
      INNER JOIN [DishesHistory] AS DH
     ON OD.[DishesHistoryID] = DH.[DishesHistoryID]
      WHERE O.[OrderID] = @OrderID
      )
END
GO
```

8. **ReservationsCount** – ilość złożonych przez danego klienta

```
CREATE FUNCTION [ReservationsCount]
(
	@CustomerID int
)
RETURNS int
AS
	BEGIN
	DECLARE @CountIndividual int
	DECLARE @CountFirm int
	SET @CountIndividual =
		(
		SELECT COUNT(*) FROM [Reservations] AS R
		INNER JOIN [ReservationsIndividuals] AS RI
		ON R.[ReservationID] = RI.[ReservationID]
		WHERE RI.[CustomerID] = @CustomerID
		)
	SET @CountFirm =
		(
		SELECT COUNT(*) FROM [Reservations] AS R
		INNER JOIN [ReservationsFirms] AS RF
		ON R.[ReservationID] = RF.[ReservationID]
		WHERE RF.[FirmID] = @CustomerID
		)
	RETURN (
		@CountIndividual + @CountFirm
	)
END
GO
```

9. Triggery

1. **DishCountCheck** – po każdym dodaniu dania do zamówienia sprawdza, czy jego dostępna ilość nie jest mniejsza od minimalnej i w razie potrzeby usuwa danie z menu

```
CREATE TRIGGER [DishCountCheck] ON [DishesHistory] AFTER UPDATE
AS
BEGIN
	SET NOCOUNT ON
	DECLARE @DishName varchar(50)
	SET @DishName =
		(SELECT D.[DishName] FROM [Inserted]
		INNER JOIN [Dishes] AS D
		ON [Inserted].[DishID] = D.[DishID])
	IF EXISTS
	(
	SELECT * FROM [Inserted]
	INNER JOIN [Dishes] AS D ON [Inserted].[DishID] = D.[DishID]
	WHERE
	[Inserted].[UnitsInStock] < D.[MinStockValue]
	)
	BEGIN;
		EXECUTE [RemoveDishFromMenu] @DishName;
	END
END
GO

ALTER TABLE [DishesHistory] ENABLE TRIGGER [DishCountCheck]
```

2. **UpdateUserDiscounts** – po każdym zamówieniu sprawdza czy klient nabył prawa do zniżki

```
CREATE TRIGGER [UpdateUserDiscounts] ON [OrderDetails]
AFTER INSERT
AS
BEGIN
  SET NOCOUNT ON

  DECLARE @CustomerID int
  DECLARE @DiscountFTID int
  DECLARE @DiscountSTID int
  DECLARE @LastDiscountDate date
  DECLARE @Z1 decimal(10,2)
  DECLARE @K1 decimal(10,2)
  DECLARE @K2 decimal(10,2)


     SET @CustomerID =
    (SELECT O.[CustomerID] FROM [Inserted]
    INNER JOIN [Orders] AS O
    ON [Inserted].[OrderID] = O.[OrderID])

     IF EXISTS
     (
          SELECT [CustomerID] FROM [CustomerIndividuals]
          WHERE [CustomerID] = @CustomerID
     )
     BEGIN

          IF NOT EXISTS
               (
               SELECT [DiscountID] FROM [CustomerDiscountFT]
               WHERE [CustomerI = @CustomerID
               )
               BEGIN

                    SET @DiscountFTID =
                    (
                         SELECT [DiscountID] FROM [Discounts]
                         WHERE [EndDate] IS NULL AND
                         [DiscountType] = 1
```

```
            )
            SET @Z1 = (
                  SELECT DSD.[Value] FROM
                  [DiscountSetDetails] AS DSD
                  INNER JOIN [DiscountsSet] AS DS
                  ON DS.[SetID] = DSD.[SetID]
                  WHERE DSD.[DiscountID] = @DiscountFTID
                  AND [SetName] = 'Z1'
                  )
            SET @K1 = (
                  SELECT DSD.[Value] FROM
                  [DiscountSetDetails] AS DSD
                  INNER JOIN [DiscountsSet] AS DS
                  ON DS.[SetID] = DSD.[SetID]
                  WHERE DSD.[DiscountID] = @DiscountFTID
                  AND [SetName] =  'K1'
                  )

            IF
            (
                  (SELECT COUNT(*) FROM [Orders] AS O
                  WHERE O.[CustomerID] = @CustomerID AND
                  (SELECT SUM([DishPrice] * [Quantity])
                   FROM [OrderDetails]
                   WHERE [OrderID] = O.[OrderID]
                  GROUP BY [OrderID]) > @Z1
                  GROUP BY O.[CustomerID])
                  >= @Z1
            )
            BEGIN
                  INSERT INTO
                  [CustomerDiscountFT]([CustomerID],
                  [DiscountID])
                  VALUES (@CustomerID, @DiscountFTID)
            END
      END

      UPDATE [CustomerDiscountsST]
       SET [UseDate] = GETDATE() WHERE [UseDate] IS NULL
      AND EXISTS
      (SELECT DSD.[DiscountID] FROM [DiscountSetDetails]
      AS DSD
```

```
INNER JOIN [DiscountsSet] AS DS ON
DS.[SetID] = DSD.[SetID]
WHERE DSD.[DiscountID] = [DiscountID]
AND [SetName] = 'D1' AND
DATEDIFF(DAY, [ReceivedDate], GETDATE()) > [Value]
)

IF NOT EXISTS
(
      SELECT [DiscountSTID]
      FROM [CustomerDiscountsST]
      WHERE [CustomerID] = @CustomerID
      AND [UseDate] IS NULL
)
BEGIN

      SET @LastDiscountDate=(
            SELECT TOP 1 [UseDate]
            FROM [CustomerDiscountsST]
            WHERE [CustomerID] = 2
            AND [UseDate] IS NOT NULL
            ORDER BY [UseDate] DESC
      )
      SET @DiscountSTID =
      (
            SELECT [DiscountID] FROM [Discounts]
            WHERE [EndDate] IS NULL
            AND [DiscountType] = 2
      )
      SET @K2 = (
            SELECT DSD.[Value]
            FROM [DiscountSetDetails] AS DSD
            INNER JOIN [DiscountsSet] AS DS
            ON DS.[SetID] = DSD.[SetID]
            WHERE DSD.[DiscountID] = @DiscountFTID
            AND [SetName] = 'K2'
            )



      IF
      (
```

```
                        (SELECT SUM(Sumy.suma) FROM (
                        SELECT (SELECT
                        SUM(OD.[DishPrice] * OD.[Quantity])
                        FROM [OrderDetails] AS OD
                        WHERE OD.[OrderID] = O.[OrderID]
                        GROUP BY OD.[OrderID]) AS suma,
                        O.[CustomerID] AS Customer
                        FROM [Orders] AS O
                        WHERE O.[CustomerID] = 2 AND
                        O.[OrderDate] >
                        ISNULL('2021-04-04', '1980-01-01')
                        GROUP BY O.[CustomerID], O.[OrderID])
                        AS Sumy
                        GROUP BY Sumy.[Customer]) > @K2

                    )
                    BEGIN
                        INSERT INTO
                        [CustomerDiscountsST]([CustomerID],
                        [DiscountID])
                        VALUES (@CustomerID, @DiscountSTID)
                    END
                END
    END

END
GO

ALTER TABLE [OrderDetails] ENABLE TRIGGER [UpdateUserDiscounts]
```

10. Indeksy

1. **Categories** – [CategoryName]

```
CREATE NONCLUSTERED INDEX [Category_Index] ON [Categories]
([CategoryName] ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
```

2. **Cities** – [CityName]

```
CREATE NONCLUSTERED INDEX [Cities_Index] ON [Cities]
([CityName] ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
```

3. **CustomerDiscountFT** – [ReceivedDate]

```
CREATE NONCLUSTERED INDEX [CustomerDiscountFT_Index]
ON [CustomerDiscountFT]
([ReceivedDate] ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
```

4. **CustomerDiscountST** – [ReceivedDate], [UseDate], [CustomerID]

```
CREATE NONCLUSTERED INDEX [CustomerDiscountsST_Index]
ON [CustomerDiscounsST]
([ReceivedDate] ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
```

```
CREATE NONCLUSTERED INDEX [CustomerDiscountsST_Index_1]
ON [CustomerDiscounsST]
([UseDate] ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
```

```
CREATE NONCLUSTERED INDEX [CustomerDiscountsST_Index_2]
ON [CustomerDiscounsST]
([CustomerID] ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
```

5. **CustomerFirms** – [NIP], [CompanyName]

```
CREATE NONCLUSTERED INDEX [CustomerFirms_Index] ON [CustomerFirms]
([NIP] ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
```

```
CREATE NONCLUSTERED INDEX [CustomerFirms_Index_1]
ON [CustomerFirms]
([CompanyName] ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
```

6. **CustomerFirmsEmployees** – [FirmID]

```
CREATE NONCLUSTERED INDEX [CustomerFirmsEmployees_Index]
ON [CustomerFirmsEmployees]
([FirmID] ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
```

7. **Customers** – [Phone]

```
CREATE NONCLUSTERED INDEX [Customers_Index] ON [Customers]
([Phone] ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
```

8. **DiscountsSet** – [SetName]

```
CREATE NONCLUSTERED INDEX [DiscountsSet_Index] ON [Discounts]
([SetName] ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
```

9. **Dishes** – [DishName], [CategoryID]

```
CREATE NONCLUSTERED INDEX [Dishes_Index] ON [Dishes]
([DishName] ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
```

```
CREATE NONCLUSTERED INDEX [Dishes_Index_1] ON [Dishes]
([CategoryID] ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
```

10. **DishesHistory** – [DishID], [InMenuDate], [OutMenuDate]

```
CREATE NONCLUSTERED INDEX [DishesHistory_Index] ON [DishesHistory]
([DishID] ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
```

```
CREATE NONCLUSTERED INDEX [DishesHistory_Index_1]
ON [DishesHistory]
([InMenuDate] ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
```

```
CREATE NONCLUSTERED INDEX [DishesHistory_Index_2]
ON [DishesHistory]
([OutMenuDate] ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
```

11. **Employees** – [Phone]

```
CREATE NONCLUSTERED INDEX [Employees_Index] ON [Employees]
([Phone] ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
```

12. **Orders** – [CustomerID], [ReceiveDate], [OrderDate]

```
CREATE NONCLUSTERED INDEX [Orders_Index] ON [Orders]
([CustomerID] ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
```

```
CREATE NONCLUSTERED INDEX [Orders_Index_1] ON [Orders]
([ReceiveDate] ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
```

```
CREATE NONCLUSTERED INDEX [Orders_Index_2] ON [Orders]
([OrderDate] ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
```

13. **PaymentType** – [PaymentName]

```
CREATE NONCLUSTERED INDEX [PaymentType_Index] ON [PaymentType]
([PaymentName] ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
```

14. **Reservations** – [ReservationDate]

```
CREATE NONCLUSTERED INDEX [Reservations_Index] ON [Reservations]
([ReservationDate] ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
```

15. **ReservationsFirms** – [FirmID]

```
CREATE NONCLUSTERED INDEX [ReservationsFirms_Index]
ON [ReservationsFirms]
([FirmID] ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
```

16. **ReservationsFirmsDetails** – [ReservationID], [TableID]

```
CREATE NONCLUSTERED INDEX [ReservationsFirmsDetails_Index]
ON [ReservationsFirmsDetails]
([ReservationID] ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
```

```
CREATE NONCLUSTERED INDEX [ReservationsFirmsDetails_Index_1]
ON [ReservationsFirmsDetails]
([TableID] ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
```

17. **ReservationsFirmsEmployees** – [TableID]

```
CREATE NONCLUSTERED INDEX [ReservationsFirmsEmployees_Index]
ON [ReservationsFirmsEmployees]
([TableID] ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
```

18. **ReservationsIndividuals** – [TableID], [CustomerID], [OrderID]

```
CREATE NONCLUSTERED INDEX [ReservationsIndividuals_Index]
ON [ReservationsIndividuals]
([TableID] ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
```

```
CREATE NONCLUSTERED INDEX [ReservationsIndividuals_Index_1]
ON [ReservationsIndividuals]
([CustomerID] ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
```

```
CREATE NONCLUSTERED INDEX [ReservationsIndividuals_Index_2]
ON [ReservationsIndividuals]
([OrderID] ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
```

11. Uprawnienia użytkowników:

1. Administrator systemu
    - Dostęp do wszystkich danych
    - Dodawanie/usuwanie/edytowanie danych
2. Menadżer restauracji
    - Generowanie raportów
        - Dotyczących rezerwacji stolików
        - Dotyczących zamawianych dań
        - Dotyczących pory zamawiania dań
        - Dotyczących firm/klientów indywidualnych
        - Dotyczących przyznawanych zniżek
    - Dodawanie pracowników
    - Ustawianie progów rabatowych
    - Dodawanie i edytowanie dań w bazie
    - Dodawanie i edytowanie kategorii dań
    - Zarządzanie menu
        - Dodawanie/usuwanie pozycji w menu
        - Aktualizacja menu
        - Aktualizacja cen dań
3. Pracownik restauracji
    - Przyjmowanie zamówień na miejscu
    - Przyjmowanie płatności
    - Wydawanie zamówień
    - Akceptowanie rezerwacji i przydzielanie stolików
    - Dostęp do podglądu zarezerwowanych stolików
    - Możliwość założenia konta klientowi

4. Klient firmowy
    - Składanie zamówień online
    - Przeglądanie menu
    - Składanie rezerwacji
        - Dla całej firmy
        - Dla konkretnych pracowników
5. Klient prywatny (klient indywidualny)
    - Tworzenie konta online
    - Składanie zamówień online
    - Składanie rezerwacji (po spełnieniu odpowiednich warunków)
    - Przeglądanie menu
6. System
    - Naliczanie rabatów
    - Usuwanie i dodawanie dania z menu.
    - Aktualizacja daty ostatniego pobytu dania w menu
    - Generowanie backupu
    - Blokada możliwości zamawiania dań z owocami morza po upływie poniedziałku poprzedzającego dni czwartek, piątek, sobota tego samego tygodnia
    - Monitorowanie i aktualizacja stanu magazynu
    - Generowanie wiadomości do klientów i pracowników