Filip Krupa
Hubert Gancarczyk
Marcel Duda

1. Użytkownicy

1.1. Właściciel
1.2. Administrator systemu
1.3. Menadżer restauracji
1.4. Kelner/Kasjer
1.5. Magazynier
1.6. Klient indywidualny
1.7. Klient biznesowy

2. Funkcje

2.1. Właściciel
- 2.1.1. Ustalenie dostępnych produktów
- 2.1.2. Zmiana liczby stolików
- 2.1.3. Zmiana wartości Z1,K1,R1,K2,R2,D1,WK,WZ
- 2.1.4. Zmiana cen produktów

2.2. Menadżer
- 2.2.1. Ustalenie menu
- 2.2.2. Generowanie raportów dla klientów indywidualnych
- 2.2.3. Generowanie raportów o stolikach
- 2.2.4. Generowanie raportów o daniach
- 2.2.5. Generowanie raportów dla klientów biznesowych

2.3. Magazynier
- 2.3.1. Składanie zamówienia na owoce morza
- 2.3.2. Zamawianie składników
- 2.3.3. Wyświetlanie stanu magazynu
- 2.3.4. Przyjęcie na magazyn

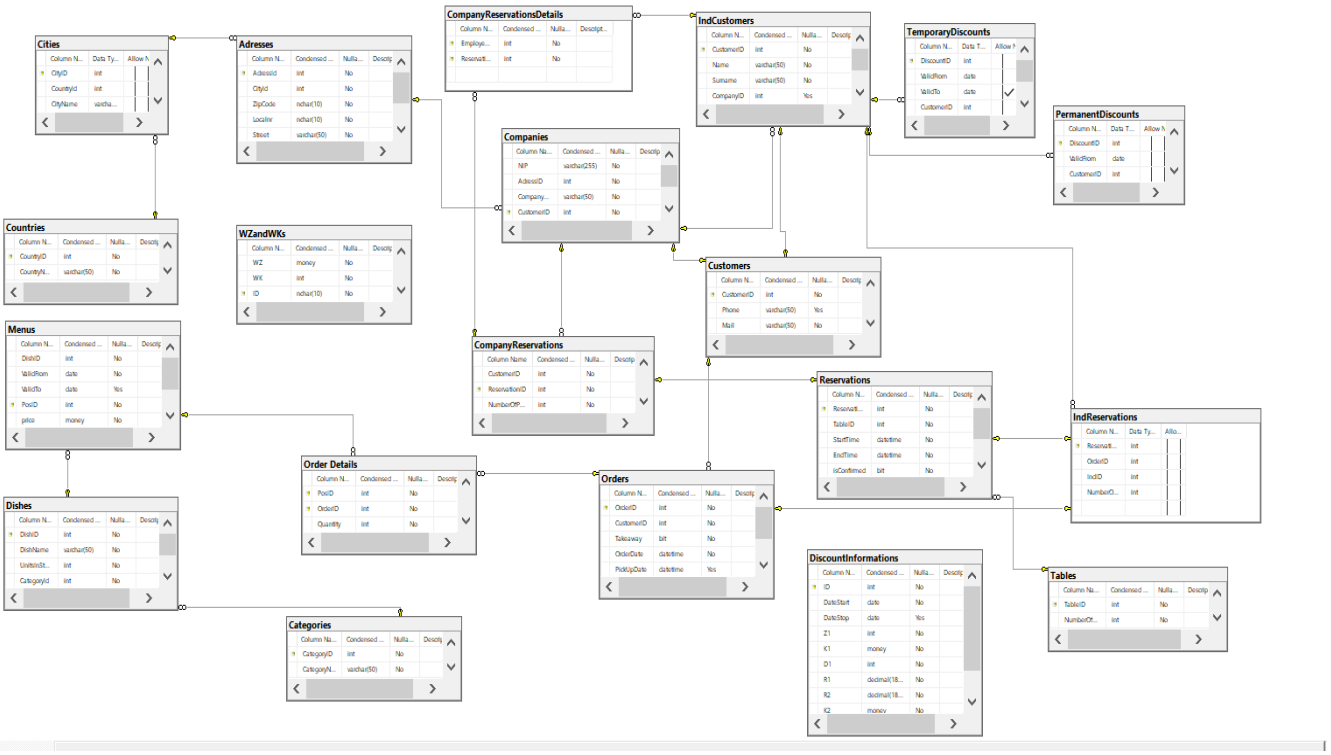
2.4. Kelner
- 2.4.1. Wyświetlanie danych o zamówieniu
- 2.4.2. Wyświetlanie rezerwacji stolików
- 2.4.3. Składanie zamówień na wynos na miejscu
- 2.4.4. Składanie zamówień na miejscu
- 2.4.5. Wyświetlanie (drukowanie) menu
- 2.4.6. Wyświetlanie listy klientów
- 2.4.7. Wyświetlanie liczby stolików
- 2.4.8. Potwierdzanie zamówień i rezerwacji klientów
- 2.4.9. Sprawdzanie czy w danym czasie jest dostępny stolik dla danej liczby osób
- 2.4.10. Wyświetlanie danych klienta
- 2.4.11. Realizacja zamówień firm

2.5. System
- 2.5.1. Przypominanie zmiany menu co 2 tygodnie
- 2.5.2. Wysyłanie informacji klientom o rabatach
- 2.5.3. Śledzenie warunków zamówienia i rabatów
- 2.5.4. Przypominanie o niedoborze składników

2.6. Klient indywidualny
- 2.6.1. Składanie zamówień na wynos przez stronę www
- 2.6.2. Rezerwacja stolika dla co najmniej dwóch osób

3. Schemat

3.1. Diagram



3.2. Kod Sql:

3.2.1. Tabela Adresses

```
USE [u_maduda]
GO

/****** Object:  Table [dbo].[Adresses]    Script Date: 11.12.2022 20:13:29 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Adresses](
    [AdressId] [int] NOT NULL,
    [CityId] [int] NOT NULL,
    [ZipCode] [nchar](10) NOT NULL,
    [Localnr] [nchar](10) NOT NULL,
    [Street] [varchar](50) NOT NULL,
 CONSTRAINT [PK_Adresses] PRIMARY KEY CLUSTERED
(
    [AdressId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Adresses]  WITH CHECK ADD  CONSTRAINT [FK_Adresses_Cities] FOREIGN KEY([CityId])
REFERENCES [dbo].[Cities] ([CityID])
GO

ALTER TABLE [dbo].[Adresses] CHECK CONSTRAINT [FK_Adresses_Cities]
GO

ALTER TABLE [dbo].[Adresses]  WITH CHECK ADD  CONSTRAINT [CK_Adresses] CHECK  ((isnumeric([ZipCode])=(1)))
GO

ALTER TABLE [dbo].[Adresses] CHECK CONSTRAINT [CK_Adresses]
GO
```

nazwa kolumny	typ	czy może być null	opis
AdressId	int	nie	Unikalne Id - klucz główny
CityId	int	nie	Id miasta
ZipCode	nchar(10)	nie	kod pocztowy
LocalNr	nchar(10)	nie	numer budynku/lokalu
Street	varchar(50)	nie	nazwa ulicy

Warunki integralnościowe:

- ZipCode składa się wyłącznie z cyfr
- AdressId nie jest nullem
- CityId nie jest nullem
- ZipCode nie jest nullem
- Localnr nie jest nullem
- Street nie jest nullem

3.2.2. Tabela Categories

```
USE [u_maduda]
GO

/****** Object:  Table [dbo].[Categories]    Script Date: 11.12.2022 20:21:30 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

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

nazwa kolumny	typ	czy może być null	opis
CategoryId	int	nie	Id kategorii - klucz główny
CategoryName	varchar(50)	nie	nazwa kategorii

Warunki integralnościowe:

- CategoryName jest unikalne
- CategoryID nie jest nullem
- CategoryName nie jest nulem

3.2.3. Tabela Cities

```
USE [u_maduda]
GO

/****** Object:  Table [dbo].[Cities]    Script Date: 11.12.2022 20:22:26 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Cities](
    [CityID] [int] NOT NULL,
    [CountryId] [int] NOT NULL,
    [CityName] [varchar](50) 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],
 CONSTRAINT [C_CityName] UNIQUE NONCLUSTERED
(
    [CityName] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Cities]  WITH CHECK ADD  CONSTRAINT [FK_Cities_Countries] FOREIGN KEY([CountryId])
REFERENCES [dbo].[Countries] ([CountryID])
GO

ALTER TABLE [dbo].[Cities] CHECK CONSTRAINT [FK_Cities_Countries]
GO
```

nazwa kolumny	typ	czy może być null	opis
CityId	int	nie	Unikalne Id - klucz główny
CountryId	int	nie	Id kraju, w którym leży miasto
CityName	varchar(50)	nie	Nazwa miasta

Warunki integralnościowe:

- CityName jest unikalne
- CountryId nie jest nullem
- CityName nie jest nullem

3.2.4. Tabela Companies

```
USE [u_maduda]
GO

/****** Object:  Table [dbo].[Companies]    Script Date: 11.12.2022 20:23:11 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Companies](
    [NIP] [varchar](255) NOT NULL,
    [AdressID] [int] NOT NULL,
    [CompanyName] [varchar](50) NOT NULL,
    [CustomerID] [int] NOT NULL,
 CONSTRAINT [PK_Companies] PRIMARY KEY CLUSTERED
(
    [CustomerID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [Co_CompanyName] UNIQUE NONCLUSTERED
(
    [CompanyName] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [Co_NIP] UNIQUE NONCLUSTERED
(
    [NIP] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Companies]  WITH CHECK ADD  CONSTRAINT [FK_Companies_Adresses] FOREIGN KEY([AdressID])
REFERENCES [dbo].[Adresses] ([AdressId])
GO

ALTER TABLE [dbo].[Companies] CHECK CONSTRAINT [FK_Companies_Adresses]
GO

ALTER TABLE [dbo].[Companies]  WITH CHECK ADD  CONSTRAINT [FK_Companies_Customers] FOREIGN KEY([CustomerID])
REFERENCES [dbo].[Customers] ([CustomerID])
GO

ALTER TABLE [dbo].[Companies] CHECK CONSTRAINT [FK_Companies_Customers]
GO
```

nazwa kolumny	typ	czy może być null	opis
NIP	varchar(255)	nie	Numer NIP firmy - klucz główny
AdressID	int	nie	ID adresu firmy
CompanyName	varchar(50)	nie	Nazwa firmy
CustomerID	int	nie	Numer klienta przypisany firmie

Warunki integralnościowe:

- CompanyName jest unikalne
- NIP jest unikalne
- NIP nie jest nullem
- AdressID nie jest nullem
- CompanyName nie jest nullem
- CustomerId nie jest nullem

3.2.5. Tabela CompanyReservations

```
USE [u_maduda]
GO

/****** Object:  Table [dbo].[CompanyReservations]    Script Date: 13.12.2022 17:37:18 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[CompanyReservations](
    [CustomerID] [int] NOT NULL,
    [ReservationID] [int] NOT NULL,
    [NumberOfPeople] [int] NOT NULL,
 CONSTRAINT [PK_CompanyReservations] PRIMARY KEY CLUSTERED
(
    [ReservationID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[CompanyReservations]  WITH CHECK ADD  CONSTRAINT [FK_CompanyReservations_Companies1] FOREIGN KEY([CustomerID])
REFERENCES [dbo].[Companies] ([CustomerID])
GO

ALTER TABLE [dbo].[CompanyReservations] CHECK CONSTRAINT [FK_CompanyReservations_Companies1]
GO

ALTER TABLE [dbo].[CompanyReservations]  WITH CHECK ADD  CONSTRAINT [FK_CompanyReservations_Reservations] FOREIGN KEY([ReservationID])
REFERENCES [dbo].[Reservations] ([ReservationID])
GO

ALTER TABLE [dbo].[CompanyReservations] CHECK CONSTRAINT [FK_CompanyReservations_Reservations]
GO

ALTER TABLE [dbo].[CompanyReservations]  WITH CHECK ADD  CONSTRAINT [CK_CompanyReservations] CHECK  (([NumberOfPeople]>(0)))
GO

ALTER TABLE [dbo].[CompanyReservations] CHECK CONSTRAINT [CK_CompanyReservations]
GO
```

nazwa kolumny	typ	czy może być null	opis
CustomerID	int	nie	ID klienta przypisane firmie
ReservationID	int	nie	ID rezerwacji
NumerOfPeople	int	nie	Liczba osób

Warunki integralnościowe:

- CustomerID nie jest nullem
- ReservationID nie jest nullem
- NumberOfPeople nie jest nullem i jest większe od 0.

3.2.6. Tabela CompanyReservationDetails

```
USE [u_maduda]
GO

/****** Object:  Table [dbo].[CompanyReservationsDetails]    Script Date: 17.12.2022 17:45:25 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[CompanyReservationsDetails](
    [EmployeeID] [int] NOT NULL,
    [ReservationID] [int] NOT NULL,
 CONSTRAINT [PK_Employees] PRIMARY KEY CLUSTERED
(
    [EmployeeID] ASC,
    [ReservationID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[CompanyReservationsDetails]  WITH CHECK ADD  CONSTRAINT [FK_Employees_CompanyReservations] FOREIGN KEY([ReservationID])
REFERENCES [dbo].[CompanyReservations] ([ReservationID])
GO

ALTER TABLE [dbo].[CompanyReservationsDetails] CHECK CONSTRAINT [FK_Employees_CompanyReservations]
GO

ALTER TABLE [dbo].[CompanyReservationsDetails]  WITH CHECK ADD  CONSTRAINT [FK_Employees_IndCustomers] FOREIGN KEY([EmployeeID])
REFERENCES [dbo].[IndCustomers] ([CustomerID])
GO

ALTER TABLE [dbo].[CompanyReservationsDetails] CHECK CONSTRAINT [FK_Employees_IndCustomers]
GO
```

nazwa kolumny	typ	czy może być null	opis
EmployeeID	int	nie	ID pracownika dla którego zostało złożone zamówienie firmowe imienne
ReservationID	int	nie	ID rezerwacji

3.2.7. Tabela Countries

```
USE [u_maduda]
GO

/****** Object:  Table [dbo].[Countries]    Script Date: 11.12.2022 20:24:37 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Countries](
    [CountryID] [int] NOT NULL,
    [CountryName] [varchar](50) NOT NULL,
 CONSTRAINT [PK_Countries] PRIMARY KEY CLUSTERED
(
    [CountryID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [Country_CountryName] UNIQUE NONCLUSTERED
(
    [CountryName] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

nazwa kolumny	typ	czy może być null	opis
CountryID	int	nie	Unikalne Id - klucz główny
CountryName	varchar(50)	nie	Nazwa kraju

Warunki integralnościowe:

- CountryName jest unikalne
- CountryId nie jest nullem
- CountryName nie jest nullem

3.2.8. Tabela Customers

```
USE [u_maduda]
GO

/****** Object:  Table [dbo].[Customers]    Script Date: 13.12.2022 17:38:49 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Customers](
    [CustomerID] [int] NOT NULL,
    [Phone] [varchar](50) NULL,
    [Mail] [varchar](50) 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],
 CONSTRAINT [Customers_UniqueMail] UNIQUE NONCLUSTERED
(
    [Mail] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Customers]  WITH CHECK ADD  CONSTRAINT [Customers_Mail] CHECK  (([Mail] like '%@%'))
GO

ALTER TABLE [dbo].[Customers] CHECK CONSTRAINT [Customers_Mail]
GO

ALTER TABLE [dbo].[Customers]  WITH CHECK ADD  CONSTRAINT [Customers_Phone] CHECK  (([Phone] like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'))
GO

ALTER TABLE [dbo].[Customers] CHECK CONSTRAINT [Customers_Phone]
GO
```

nazwa kolumny	typ	czy może być null	opis
CustomerID	int	nie	Unikalne Id - klucz główny
Phone	varchar(50)	tak	Numer telefonu klienta
Mail	varchar(50)	nie	Mail klienta

Warunki integralnościowe:

- Phone musi zawierać 9 cyfr
- CustomerId nie jest nullem
- Mail musi zawierać @, musi być unikalny oraz nie może być nullem.

3.2.9. Tabela DiscountInformations

```
CREATE TABLE [dbo].[DiscountInformations](
    [ID] [int] NOT NULL,
    [DateStart] [date] NOT NULL,
    [DateStop] [date] NULL,
    [Z1] [int] NOT NULL,
    [K1] [money] NOT NULL,
    [D1] [int] NOT NULL,
    [R1] [decimal](18, 0) NOT NULL,
    [R2] [decimal](18, 0) NOT NULL,
    [K2] [money] NOT NULL,
 CONSTRAINT [PK_DiscountTypes] PRIMARY KEY CLUSTERED
(
    [ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[DiscountInformations] ADD  CONSTRAINT [DF_DiscountInformations_DateStart]  DEFAULT (getdate()) FOR [DateStart]
GO

ALTER TABLE [dbo].[DiscountInformations] ADD  CONSTRAINT [DF_DiscountInformations_DateStop]  DEFAULT (NULL) FOR [DateStop]
GO

ALTER TABLE [dbo].[DiscountInformations]  WITH CHECK ADD  CONSTRAINT [CK_DiscountInformations] CHECK  (([DateStop]>[DateStart] OR [DateStop] IS NULL))
GO

ALTER TABLE [dbo].[DiscountInformations] CHECK CONSTRAINT [CK_DiscountInformations]
GO

ALTER TABLE [dbo].[DiscountInformations]  WITH CHECK ADD  CONSTRAINT [Discounts_D1] CHECK  (([D1]>(0)))
GO

ALTER TABLE [dbo].[DiscountInformations] CHECK CONSTRAINT [Discounts_D1]
GO

ALTER TABLE [dbo].[DiscountInformations]  WITH CHECK ADD  CONSTRAINT [Discounts_K1] CHECK  (([K1]>(0)))
GO

ALTER TABLE [dbo].[DiscountInformations] CHECK CONSTRAINT [Discounts_K1]
GO

ALTER TABLE [dbo].[DiscountInformations]  WITH CHECK ADD  CONSTRAINT [Discounts_K2] CHECK  (([K2]>(0)))
GO

ALTER TABLE [dbo].[DiscountInformations]  WITH CHECK ADD  CONSTRAINT [Discounts_K2] CHECK  (([K2]>(0)))
GO

ALTER TABLE [dbo].[DiscountInformations] CHECK CONSTRAINT [Discounts_K2]
GO

ALTER TABLE [dbo].[DiscountInformations]  WITH CHECK ADD  CONSTRAINT [Discounts_R1] CHECK  (([R1]>(0)))
GO

ALTER TABLE [dbo].[DiscountInformations] CHECK CONSTRAINT [Discounts_R1]
GO

ALTER TABLE [dbo].[DiscountInformations]  WITH CHECK ADD  CONSTRAINT [Discounts_R2] CHECK  (([R2]>(0)))
GO

ALTER TABLE [dbo].[DiscountInformations] CHECK CONSTRAINT [Discounts_R2]
GO

ALTER TABLE [dbo].[DiscountInformations]  WITH CHECK ADD  CONSTRAINT [Discounts_Z1] CHECK  (([Z1]>(0)))
GO

ALTER TABLE [dbo].[DiscountInformations] CHECK CONSTRAINT [Discounts_Z1]
GO
```

nazwa kolumny	typ	czy może być null	opis
ID	int	nie	ID wpisu - klucz główny
DateStart	date	nie	Data, od której obowiązywały parametry
DateStop	date	tak	Data, do której obowiązywały parametry lub null jeśli dalej obowiązują
Z1	int	nie	Minimalna liczba zamówień wymagana do przyznania zniżki na wszystkie zamówienia
K1	money	nie	Minimalna kwota, aby zamówienie przybliżało klienta do otrzymania zniżki na wszystkie zamówienia
D1	int	nie	Długość trwania jednorazowej zniżki tymczasowej w dniach
R1	decimal(18,0)	nie	Wysokość zniżki na wszystkie zamówienia wyrażona w procentach
R2	decimal(18,0)	nie	Wysokość jednorazowej zniżki tymczasowej wyrażona w procentach
K2	money	nie	Minimalna liczba zamówień wymagana do przyznania jednorazowej tymczasowej zniżki

Warunki integralnościowe:

- ID nie jest nullem
- DateStop jest później niż DateStart lub DateStop jest nullem (domyslny przypadek)
- DateStart nie jest nullem i domyslnie przyjmuje wartosc getdate().
- Z1 nie jest nullem i jest większe od 0.
- K1 nie jest nullem i jest większe od 0.
- D1 nie jest nullem i jest większe od 0.
- R1 nie jest nullem i jest większe od 0.
- R2 nie jest nullem i jest większe od 0.
- K2 nie jest nullem i jest większe od 0.

3.2.10. Tabela Dishes

```
USE [u_maduda]
GO

/****** Object:  Table [dbo].[Dishes]    Script Date: 11.12.2022 20:28:12 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Dishes](
    [DishID] [int] NOT NULL,
    [DishName] [varchar](50) NOT NULL,
    [UnitsInStock] [int] NOT NULL,
    [CategoryId] [int] NOT NULL,
 CONSTRAINT [PK_Dishes] PRIMARY KEY CLUSTERED
(
    [DishID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [IX_Dishes] UNIQUE NONCLUSTERED
(
    [DishName] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

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

ALTER TABLE [dbo].[Dishes] CHECK CONSTRAINT [FK_Dishes_Categories]
GO

ALTER TABLE [dbo].[Dishes]  WITH CHECK ADD  CONSTRAINT [CK_Dishes] CHECK  (([UnitsInStock]>=(0)))
GO

ALTER TABLE [dbo].[Dishes] CHECK CONSTRAINT [CK_Dishes]
GO
```

nazwa kolumny	typ	czy może być null	opis
DishID	int	nie	Unikalne Id dania - klucz główny
DishName	varchar(50)	nie	Nazwa potrawy
UnitsInStock	int	nie	Liczba możliwych porcji do przyrządzenia
CategoryId	int	nie	Id kategorii, do której należy danie

Warunki integralnościowe:

- UnitsInStock jest większe lub równe 0 i nie jest nullem.
- DishName jest unikalny i nie jest nullem.
- DishId nie jest nullem
- CategoryId nie jest nullem

3.2.11. Tabela IndCustomers

```
USE [u_maduda]
GO

/****** Object:  Table [dbo].[IndCustomers]    Script Date: 11.12.2022 20:28:56 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[IndCustomers](
    [CustomerID] [int] NOT NULL,
    [Name] [varchar](50) NOT NULL,
    [Surname] [varchar](50) NOT NULL,
    [CompanyID] [int] NULL,
 CONSTRAINT [PK_IndCustomers] PRIMARY KEY CLUSTERED
(
    [CustomerID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[IndCustomers]  WITH CHECK ADD  CONSTRAINT [FK_IndCustomers_Companies] FOREIGN KEY([CompanyID])
REFERENCES [dbo].[Companies] ([CustomerID])
GO

ALTER TABLE [dbo].[IndCustomers] CHECK CONSTRAINT [FK_IndCustomers_Companies]
GO

ALTER TABLE [dbo].[IndCustomers]  WITH CHECK ADD  CONSTRAINT [FK_IndCustomers_Customers] FOREIGN KEY([CustomerID])
REFERENCES [dbo].[Customers] ([CustomerID])
GO

ALTER TABLE [dbo].[IndCustomers] CHECK CONSTRAINT [FK_IndCustomers_Customers]
GO
```

nazwa kolumny	typ	czy może być null	opis
CustomerId	int	nie	Unikalne Id klienta- klucz główny
Name	varchar(50)	nie	Imię klienta
Surname	varchar(50)	nie	Nazwisko klienta
CompanyID	int	tak	ID firmy, w której pracuje lub null jeśli przyszedł prywatnie

Warunki integralnościowe:

- CustomerID nie jest nullem
- Name nie jest nullem
- Surname nie jest nullem

3.2.12. Tabela IndReservations

```
USE [u_maduda]
GO

/****** Object:  Table [dbo].[IndReservations]    Script Date: 17.12.2022 17:50:51 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[IndReservations](
    [ReservationID] [int] NOT NULL,
    [OrderID] [int] NOT NULL,
    [IndID] [int] NOT NULL,
    [NumberOfPeople] [int] NOT NULL,
 CONSTRAINT [PK_IndReservations] 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],
 CONSTRAINT [IX_IndReservations] UNIQUE NONCLUSTERED
(
    [OrderID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[IndReservations]  WITH CHECK ADD  CONSTRAINT [FK_IndReservations_IndCustomers] FOREIGN KEY([IndID])
REFERENCES [dbo].[IndCustomers] ([CustomerID])
GO

ALTER TABLE [dbo].[IndReservations] CHECK CONSTRAINT [FK_IndReservations_IndCustomers]
GO

ALTER TABLE [dbo].[IndReservations]  WITH CHECK ADD  CONSTRAINT [FK_IndReservations_Orders] FOREIGN KEY([OrderID])
REFERENCES [dbo].[Orders] ([OrderID])
GO

ALTER TABLE [dbo].[IndReservations] CHECK CONSTRAINT [FK_IndReservations_Orders]
GO

ALTER TABLE [dbo].[IndReservations]  WITH CHECK ADD  CONSTRAINT [CK_IndReservations] CHECK  (([NumberOfPeople]>(0)))
GO

ALTER TABLE [dbo].[IndReservations] CHECK CONSTRAINT [CK_IndReservations]
GO
```

nazwa kolumny	typ	czy może być null	opis
ReservationID	int	nie	Unikalne Id rezerwacji - klucz główny
OrderID	int	nie	Id zamówienia powiązanego z rezerwacją
IndID	int	nie	Id klienta
NumberOfPeople	int	nie	Liczba gości

Warunki integralnościowe:

- NumberOfPeople musi być większe od 0 i nie może być nullem.
- OrderID jest unikalne i nie może być nullem.
- ReservationID nie jest nullem
- IndID nie jest nullem

3.2.13. Tabela Menus

```
USE [u_maduda]
GO

/****** Object:  Table [dbo].[Menus]    Script Date: 13.12.2022 17:44:16 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Menus](
    [DishID] [int] NOT NULL,
    [ValidFrom] [date] NOT NULL,
    [ValidTo] [date] NULL,
    [PosID] [int] NOT NULL,
    [price] [money] NOT NULL,
 CONSTRAINT [PK_Menus] PRIMARY KEY CLUSTERED
(
    [PosID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Menus] ADD  CONSTRAINT [DF_Menus_ValidFrom]  DEFAULT (getdate()) FOR [ValidFrom]
GO

ALTER TABLE [dbo].[Menus]  WITH CHECK ADD  CONSTRAINT [FK_Menus_Dishes] FOREIGN KEY([DishID])
REFERENCES [dbo].[Dishes] ([DishID])
GO

ALTER TABLE [dbo].[Menus] CHECK CONSTRAINT [FK_Menus_Dishes]
GO

ALTER TABLE [dbo].[Menus]  WITH CHECK ADD  CONSTRAINT [CK_Menus] CHECK  (([ValidTo]>[ValidFrom] OR [ValidTo] IS NULL))
GO

ALTER TABLE [dbo].[Menus] CHECK CONSTRAINT [CK_Menus]
GO

ALTER TABLE [dbo].[Menus]  WITH CHECK ADD  CONSTRAINT [CK_Menus_1] CHECK  (([price]>(0)))
GO

ALTER TABLE [dbo].[Menus] CHECK CONSTRAINT [CK_Menus_1]
GO
```

nazwa kolumny	typ	czy może być null	opis
DishID	int	nie	ID potrawy
ValidFrom	date	nie	Od kiedy jest w menu
ValidTo	date	tak	Do kiedy było w menu lub null jeśli dalej je serwujemy
PosID	int	nie	Unikalne ID pozycji w menu -klucz główny
price	money	nie	Cena dania

Warunki integralnościowe:

- Price musi być większe od 0 i nie może być nullem.
- ValidTo musi być większe od ValidFrom lub ValidTo jest nullem
- DishID nie jest nullem
- ValidFrom nie jest nullem i domyślnie przyjmuje wartość getdate().
- PosID nie jest nullem

3.2.14. Tabela Order Details

```
USE [u_maduda]
GO

/****** Object:  Table [dbo].[Order Details]    Script Date: 11.12.2022 20:30:41 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Order Details](
    [PosID] [int] NOT NULL,
    [OrderID] [int] NOT NULL,
    [Quantity] [int] NOT NULL,
 CONSTRAINT [PK_Order Details_1] PRIMARY KEY CLUSTERED
(
    [PosID] ASC,
    [OrderID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Order Details]  WITH CHECK ADD  CONSTRAINT [FK_Order Details_Menus] FOREIGN KEY([PosID])
REFERENCES [dbo].[Menus] ([PosID])
GO

ALTER TABLE [dbo].[Order Details] CHECK CONSTRAINT [FK_Order Details_Menus]
GO

ALTER TABLE [dbo].[Order Details]  WITH CHECK ADD  CONSTRAINT [FK_Order Details_Orders] FOREIGN KEY([OrderID])
REFERENCES [dbo].[Orders] ([OrderID])
GO

ALTER TABLE [dbo].[Order Details] CHECK CONSTRAINT [FK_Order Details_Orders]
GO

ALTER TABLE [dbo].[Order Details]  WITH CHECK ADD  CONSTRAINT [CK_Order Details] CHECK  (([Quantity]>(0)))
GO

ALTER TABLE [dbo].[Order Details] CHECK CONSTRAINT [CK_Order Details]
GO
```

nazwa kolumny	typ	czy może być null	opis
PosID	int	nie	ID pozycji w menu
OrderID	int	nie	ID zamówienia - klucz główny
Quantity	int	nie	Liczba zamówionych jednostek dania w tym zamówieniu

Warunki integralnościowe:

- Quantity musi być większe od 0 i nie może być nullem.
- PosID nie jest nullem
- OrderID nie jest nullem

3.2.15. Tabela Orders

```
USE [u_maduda]
GO

/****** Object:  Table [dbo].[Orders]    Script Date: 13.12.2022 17:45:40 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Orders](
    [OrderID] [int] NOT NULL,
    [CustomerID] [int] NOT NULL,
    [Takeaway] [bit] NOT NULL,
    [OrderDate] [datetime] NOT NULL,
    [PickUpDate] [datetime] NULL,
 CONSTRAINT [PK_Orders] PRIMARY KEY CLUSTERED
(
    [OrderID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Orders] ADD  CONSTRAINT [DF_Orders_OrderDate]  DEFAULT (getdate()) FOR [OrderDate]
GO

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

ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [FK_Orders_Customers]
GO

ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT [CK_Orders] CHECK  (([PickUpDate]>[OrderDate]))
GO

ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [CK_Orders]
GO
```

nazwa kolumny	typ	czy może być null	opis
OrderID	int	nie	ID zamówienia - klucz główny
CustomerID	int	nie	ID klienta składającego zamówienie
Takeaway	bit	nie	Czy danie jest zamówione na wynos
OrderDate	datetime	nie	Data złożenia zamówienia
PickUpDate	datetime	tak	Data odbioru zamówienia na wynos lub null jeśli nie jest na wynos

Warunki integralnościowe:

- PickUpDate musi być później od OrderDate lub byc nullem
- OrderID nie jest nullem
- CustomerID nie jest nullem
- DiscountID nie jest nullem
- TakeAway nie jest nullem
- OrderDate nie jest nullem i domyślnie przyjmuje wartość
- getdate().

3.2.16. Tabela Reservations

```
USE [u_maduda]
GO

/****** Object:  Table [dbo].[Reservations]    Script Date: 11.12.2022 20:32:03 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Reservations](
    [ReservationID] [int] NOT NULL,
    [TableID] [int] NOT NULL,
    [StartTime] [datetime] NOT NULL,
    [EndTime] [datetime] NOT NULL,
    [isConfirmed] [bit] NOT NULL,
 CONSTRAINT [PK_Reservations] PRIMARY KEY CLUSTERED
(
    [ReservationID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Reservations]  WITH CHECK ADD  CONSTRAINT [FK_Reservations_IndReservetions] FOREIGN KEY([ReservationID])
REFERENCES [dbo].[IndReservetions] ([ReservationID])
GO

ALTER TABLE [dbo].[Reservations] CHECK CONSTRAINT [FK_Reservations_IndReservetions]
GO

ALTER TABLE [dbo].[Reservations]  WITH CHECK ADD  CONSTRAINT [FK_Reservations_Tables] FOREIGN KEY([TableID])
REFERENCES [dbo].[Tables] ([TableID])
GO

ALTER TABLE [dbo].[Reservations] CHECK CONSTRAINT [FK_Reservations_Tables]
GO

ALTER TABLE [dbo].[Reservations]  WITH CHECK ADD  CONSTRAINT [CK_Reservations] CHECK  (([EndTime]>[StartTime]))
GO

ALTER TABLE [dbo].[Reservations] CHECK CONSTRAINT [CK_Reservations]
GO
```

nazwa kolumny	typ	czy może być null	opis
ReservationID	int	nie	ID rezerwacji - klucz główny
TableID	int	nie	ID zarezerwowanego stolika
StartTime	datetime	nie	Data początku obowiązywania rezerwacji
EndTime	datetime	nie	Data końca obowiązywania rezerwacji
isConfirmed	bit	nie	Czy pracownik potwierdził rezerwacje

Warunki Integralnościowe:

- EndTime musi być później od StartTime i nie może być nullem.
- ReservationID nie jest nullem
- TableID nie jest nullem
- StartTime nie jest nullem
- isConfirmed nie jest nullem

3.2.17. Tabela Tables

```
USE [u_maduda]
GO

/****** Object:  Table [dbo].[Tables]    Script Date: 11.12.2022 20:32:51 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Tables](
    [TableID] [int] NOT NULL,
    [NumberOfChairs] [int] NOT NULL,
 CONSTRAINT [PK_Tables] PRIMARY KEY CLUSTERED
(
    [TableID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Tables]  WITH CHECK ADD  CONSTRAINT [CK_Tables] CHECK  (([TableID]>=(0)))
GO

ALTER TABLE [dbo].[Tables] CHECK CONSTRAINT [CK_Tables]
GO

ALTER TABLE [dbo].[Tables]  WITH CHECK ADD  CONSTRAINT [CK_Tables_1] CHECK  (([NumberOfChairs]>(0)))
GO

ALTER TABLE [dbo].[Tables] CHECK CONSTRAINT [CK_Tables_1]
GO
```

nazwa kolumny	typ	czy może być null	opis
TableID	int	nie	ID stolika
NumberOfChairs	int	nie	Liczba krzeseł przy stoliku

Warunki integralnościowe:

- TableID musi być większe od 0 i nie może być nullem.
- NumberOfChairs musi być większe od 0 i nie może być nullem.

3.2.18. Tabela WZandWKs

```
USE [u_maduda]
GO

/****** Object:  Table [dbo].[WZandWKs]    Script Date: 11.12.2022 20:33:24 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[WZandWKs](
    [WZ] [money] NOT NULL,
    [WK] [int] NOT NULL,
    [ID] [nchar](10) NOT NULL,
    [ValidFrom] [date] NOT NULL,
    [ValidTo] [date] NULL,
 CONSTRAINT [PK_WZandWKs] PRIMARY KEY CLUSTERED
(
    [ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[WZandWKs]  WITH CHECK ADD  CONSTRAINT [CK_WZandWKs] CHECK  (([WZ]>=(0)))
GO

ALTER TABLE [dbo].[WZandWKs] CHECK CONSTRAINT [CK_WZandWKs]
GO

ALTER TABLE [dbo].[WZandWKs]  WITH CHECK ADD  CONSTRAINT [CK_WZandWKs_1] CHECK  (([WK]>=(0)))
GO

ALTER TABLE [dbo].[WZandWKs] CHECK CONSTRAINT [CK_WZandWKs_1]
GO

ALTER TABLE [dbo].[WZandWKs]  WITH CHECK ADD  CONSTRAINT [CK_WZandWKs_2] CHECK  (([ValidTo]>[ValidFrom] OR [ValidTo] IS NULL))
GO

ALTER TABLE [dbo].[WZandWKs] CHECK CONSTRAINT [CK_WZandWKs_2]
GO
```

nazwa kolumny	typ	czy może być null	opis
WZ	money	nie	Minimalna wartość zamówienia, aby można je było złożyć przez internetowy formularz
WK	int	nie	Wymagana liczba wcześniej dokonanych zamówień, aby klient mógł składać zamówienie online
ID	nchar(10)	nie	Unikalne ID wpisu do tabeli - klucz główny

Warunki intergralnościowe:

- WZ musi być większe od 0 i nie może być nullem.
- WK musi być większe od 0 i nie może być nullem.
- ValidTo musi być później niż ValidFrom lub ValidTo jest nullem
- ID nie jest nullem
- ValidFrom nie jest nullem

3.2.19. Tabela TemporaryDiscounts

```
USE [u_maduda]
GO

/****** Object:  Table [dbo].[TemporaryDiscounts]    Script Date: 13.12.2022 17:50:42 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[TemporaryDiscounts](
    [DiscountID] [int] NOT NULL,
    [ValidFrom] [date] NOT NULL,
    [ValidTo] [date] NULL,
    [CustomerID] [int] NOT NULL,
 CONSTRAINT [PK_TemporaryDiscounts] PRIMARY KEY CLUSTERED
(
    [DiscountID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[TemporaryDiscounts] ADD  CONSTRAINT [DF_TemporaryDiscounts_ValidFrom]  DEFAULT (getdate()) FOR [ValidFrom]
GO

ALTER TABLE [dbo].[TemporaryDiscounts]  WITH CHECK ADD  CONSTRAINT [FK_TemporaryDiscounts_Customers] FOREIGN KEY([CustomerID])
REFERENCES [dbo].[Customers] ([CustomerID])
GO

ALTER TABLE [dbo].[TemporaryDiscounts] CHECK CONSTRAINT [FK_TemporaryDiscounts_Customers]
GO

ALTER TABLE [dbo].[TemporaryDiscounts]  WITH CHECK ADD  CONSTRAINT [TD_Dates] CHECK  (([ValidTo]>[ValidFrom] OR [ValidTo] IS NULL))
GO

ALTER TABLE [dbo].[TemporaryDiscounts] CHECK CONSTRAINT [TD_Dates]
GO
```

nazwa kolumny	typ	czy może być null	opis
DiscountID	int	nie	ID zniżki - klucz główny
ValidFrom	date	nie	Data aktywacji zniżki
ValidTo	date	tak	Data końca obowiązywania zniżki
CustomerID	int	nie	ID klienta dla którego przypisana jest zniżka

Warunki integralnościowe:
DiscountId nie jest nullem.
ValidFrom nie jest nullem i domyslnie przyjmuje wartosc getdate().
ValidTo jest pozniej niz ValidFrom lub jest nullem.
CustomerId nie jest nullem.

3.2.20. Tabela PermanentDiscounts

```
USE [u_maduda]
GO

/****** Object:  Table [dbo].[PermanentDiscounts]    Script Date: 13.12.2022 17:56:38 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[PermanentDiscounts](
    [DiscountID] [int] NOT NULL,
    [ValidFrom] [date] NOT NULL,
    [CustomerID] [int] NOT NULL,
 CONSTRAINT [PK_PermanentDiscounts] PRIMARY KEY CLUSTERED
(
    [DiscountID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[PermanentDiscounts] ADD  CONSTRAINT [DF_PermanentDiscounts_ValidFrom]  DEFAULT (getdate()) FOR [ValidFrom]
GO

ALTER TABLE [dbo].[PermanentDiscounts]  WITH CHECK ADD  CONSTRAINT [FK_PermanentDiscounts_Customers] FOREIGN KEY([CustomerID])
REFERENCES [dbo].[Customers] ([CustomerID])
GO

ALTER TABLE [dbo].[PermanentDiscounts] CHECK CONSTRAINT [FK_PermanentDiscounts_Customers]
GO
```

nazwa kolumny	typ	czy może być null	opis
DiscountID	int	nie	ID zniżki - klucz główny
ValidFrom	date	nie	Data aktywacji zniżki
CustomerID	int	nie	ID klienta dla którego przypisana jest zniżka

Warunki integralnościowe:

- DiscountId nie jest nullem.
- ValidFrom nie jest nullem i domyslnie przyjmuje wartosc getdate().
- CustomerId nie jest nullem.

4. Widoki

4.1.1. ClientActiveDiscounts - wyświetla aktywne zniżki klientów i ich wartość

```
create view [dbo].[clientActiveDiscounts] as
    Select CustomerID, MAX(R) as R
    from
    (
    Select I.CustomerID, T.DiscountID, D.R2 as R from IndCustomers I inner join TemporaryDiscounts T on I.CustomerID = T.CustomerID cross join DiscountInformations D where T.ValidTo > GETDATE() and D.DateStop is Null
    Union
    Select I.CustomerID, P.DiscountID, D.R1 as R from IndCustomers I inner join PermanentDiscounts P on I.CustomerID = P.CustomerID cross join DiscountInformations D where D.DateStop is Null
    ) Results
    group by CustomerID
GO
```

4.1.2. ClientDiscounts - wyświetla wszystkie zniżki klientów

```
Create view [dbo].[ClientDiscounts] as
    Select CustomerID, DiscountID, R
    from
    (
            Select I.CustomerID, T.DiscountID, D.R2 as R
            from IndCustomers I
            inner join TemporaryDiscounts T on I.CustomerID = T.CustomerID
            cross join DiscountInformations D
            where T.ValidTo > GETDATE() and D.DateStop is Null
        Union
            Select I.CustomerID, P.DiscountID, D.R1 as R
            from IndCustomers I
            inner join PermanentDiscounts P on I.CustomerID = P.CustomerID
            cross join DiscountInformations D
            where D.DateStop is Null
    ) results
GO
```

4.1.3. CurrentMenu - wyświetla aktywne menu

```
CREATE VIEW [dbo].[CurrentMenu]
AS
SELECT        dbo.Menus.DishID, dbo.Menus.ValidFrom, dbo.Menus.ValidTo, dbo.Menus.PosID, dbo.Menus.price, dbo.Dishes.DishName
FROM            dbo.Menus INNER JOIN
                         dbo.Dishes ON dbo.Menus.DishID = dbo.Dishes.DishID
WHERE        (dbo.Menus.ValidTo IS NULL) OR
                         (dbo.Menus.ValidTo > GETDATE())
GO
```

4.1.4. CurrentDiscountInfo - wyświetla aktualne parametry zniżek

```
CREATE VIEW [dbo].[CurrentDiscountsInfo]
AS
SELECT        ID, DateStart, DateStop, Z1, K1, D1, R1, R2, K2
FROM            dbo.DiscountInformations
WHERE        (DateStop IS NULL) OR
                         (DateStop > GETDATE())
GO
```

4.1.5. DiscountInfoHistory - wyświetla historię parametrów zniżek

```
CREATE VIEW [dbo].[DiscountInfoHistory]
AS
SELECT        dbo.DiscountInformations.*
FROM            dbo.DiscountInformations
GO
```

4.1.6. DishesInStock - wyświetla zawartość magazynu

```
Create view [dbo].[DishesInStock] as
    Select d.DishName, d.UnitsInStock from Dishes d where d.UnitsInStock > 0
GO
```

4.1.7. DishesRank - wyświetla ranking popularności dań z okresu dwóch tygodni

```
create view [dbo].[DishesRank] as
    Select dishName, Count(O.OrderID) as OrdersQuantity
    from Orders O
    join [Order Details] Od on O.OrderID = Od.OrderID
    join Menus M on M.PosID = Od.PosID
    join Dishes D on D.DishID = M.DishID
    where DATEDIFF(Day,Getdate(),OrderDate) <= 14
    group by DishName, M.DishID
GO
```

4.1.8. NumberOfOrders - wyświetla liczbę zamówień dla każdego klienta

```
create View [dbo].[NumberOfOrders] as
select C.CustomerID, IC.Name + ' ' +  IC.Surname as 'Name', COUNT(*) as 'Orders number' from
Customers as C
INNER JOIN Orders as O on O.CustomerID = C.CustomerID
INNER JOIN IndCustomers as IC on IC.CustomerID = C.CustomerID
GROUP BY C.CustomerID, IC.Name, IC.Surname
GO
```

4.1.9. OrderValuePreDiscount - wyświetla wartość każdego zamówienia przed nałożeniem zniżek

```
CREATE VIEW [dbo].[OrderValuePreDistcount]
AS
SELECT        dbo.Customers.CustomerID, dbo.Orders.OrderID, SUM(dbo.[Order Details].Quantity * dbo.Menus.price) AS Cost, dbo.Orders.OrderDate
FROM            dbo.Customers INNER JOIN
                         dbo.Orders ON dbo.Customers.CustomerID = dbo.Orders.CustomerID INNER JOIN
                         dbo.[Order Details] ON dbo.Orders.OrderID = dbo.[Order Details].OrderID INNER JOIN
                         dbo.Menus ON dbo.[Order Details].PosID = dbo.Menus.PosID
GROUP BY dbo.Customers.CustomerID, dbo.Orders.OrderID, dbo.Orders.OrderDate
GO
```

4.1.10. TodaysOrders - wyświetla dzisiejsze zamówienia

```
create view [dbo].[Todaysorders] as
    Select orderId from Orders o where o.PickUpDate = GETDATE()

GO
```

4.1.11. TodaysReservations - wyświetla dzisiejsze rezerwacje

```
create view [dbo].[TodaysReservations] as
    Select ReservationID, StartTime
    from Reservations R
    where CONVERT(date,StartTime) = GETDATE()

GO
```

4.1.12. UnconfirmedReservations - wyświetla niepotwierdzone rezerwacje

```
Create View [dbo].[UnconfirmedReservations] As
    select R.ReservationID
    from Reservations R
    where r.isConfirmed = 0
GO
```

4.1.13. WZWK - wyświela aktualne wartości WZ i WK

```
CREATE VIEW [dbo].[WZWK]
AS
SELECT        WZ, WK, ID
FROM            dbo.WZandWKs
GO
```

4.1.14. CompanyGuestList - wyświetla id rezerwacji, nazwe firmy oraz imię i nazwisko osób dla których jest rezerwacja

```
CREATE VIEW [dbo].[CompanyGuestList]
AS
    select R.ReservationID, C.CompanyName, IC.Name + ' ' + IC.Surname as 'Name'
    from CompanyReservationsDetails as CRD
    INNER JOIN IndCustomers as IC on IC.CustomerID = CRD.EmployeeID
    INNER JOIN CompanyReservations as CR on CRD.ReservationID = CR.ReservationID
    INNER JOIN Companies as C on C.CustomerID = CR.CustomerID
    INNER JOIN Reservations as R on R.ReservationID = CR.ReservationID
    INNER JOIN Tables as T on T.TableID = R.TableID
GO
```

4.1.15. TakeAwayOrders - wyświetla zamówienia złożone na wynos

```
create view [dbo].[TakeAwayOrders] as
select C.CustomerID, IC.Name + ' ' + IC.Surname as 'Name' from Orders as O
INNER JOIN Customers as C on C.CustomerID = O.CustomerID
INNER JOIN IndCustomers as IC on IC.CustomerID = C.CustomerID
WHERE O.Takeaway = 1
GO
```

4.1.16. NumberOfOrdersForIndCustomersAndCompanies - wyświetla łączną liczbę zamówień dla klientów indywidualnych i klientów firmowych

```
create View [dbo].[NumberOfOrdersForIndCustomersAndCompanies] as
select COUNT(*) as 'Number of Orders', 'Individual Customers' as 'Customer Type' from IndCustomers
INNER JOIN Customers on Customers.CustomerID = IndCustomers.CustomerID
INNER JOIN Orders on Orders.CustomerID = Customers.CustomerID
UNION
select COUNT(*) as 'Number of Orders', 'Companies' as 'Customer Type' from Companies
INNER JOIN Customers on Customers.CustomerID = Companies.CustomerID
INNER JOIN Orders on Orders.CustomerID = Customers.CustomerID
GO
```

4.1.17. CategoriesStatistics - wyświetla liczbę zamówień dań z danej kategorii oraz jaki przychód przyniosła

```
create View [dbo].[CategoriesStatistics] as
select C.CategoryName, COUNT(*) as 'Number of Orders', SUM(M.price) as 'Income' from Orders as O
INNER JOIN [Order Details] as OD on OD.OrderID = O.OrderID
INNER JOIN Menus as M on M.PosID = OD.PosID
INNER JOIN Dishes as D on D.DishID = M.DishID
INNER JOIN Categories as C on C.CategoryID = D.CategoryId
GROUP BY C.CategoryName
GO
```

4.1.18. CategoriesMonthlyStatistics - wyświetla liczbę zamówień dań z danej kategorii oraz jaki przychód przyniosła z podziałem na miesiące

```
create View [dbo].[CategoriesMonthlyStatistics] as
select C.CategoryName, Month(O.OrderDate) as 'Month', COUNT(*) as 'Number of Orders',
SUM(M.price) as 'Income'
from Orders as O
INNER JOIN [Order Details] as OD on OD.OrderID = O.OrderID
INNER JOIN Menus as M on M.PosID = OD.PosID
INNER JOIN Dishes as D on D.DishID = M.DishID
INNER JOIN Categories as C on C.CategoryID = D.CategoryId
GROUP BY C.CategoryName, MONTH(O.OrderDate)
GO
```

4.1.19. CategoriesYearlyStatistics - wyświetla liczbę zamówień dań z danej kategorii oraz przychód jaki przyniosła z podziałem na lata

```
create View [dbo].[CategoriesYearlyStatistics] as
select C.CategoryName, YEAR(O.OrderDate) as 'Year', COUNT(*) as 'Number of Orders',
SUM(M.price) as 'Income'
from Orders as O
INNER JOIN [Order Details] as OD on OD.OrderID = O.OrderID
INNER JOIN Menus as M on M.PosID = OD.PosID
INNER JOIN Dishes as D on D.DishID = M.DishID
INNER JOIN Categories as C on C.CategoryID = D.CategoryId
GROUP BY C.CategoryName, YEAR(O.OrderDate)
GO
```

4.1.20. CategoriesTotalStatistics - wyświetla liczbę zamówień dań z danej kategorii oraz uzyskany przychód z podziałem na lata i miesiące

```
create View [dbo].[CategoriesTotalStatistics] as
select C.CategoryName, YEAR(O.OrderDate) as 'Year', MONTH(O.OrderDate) as 'Month', COUNT(*) as 'Number of Orders',
SUM(M.price) as 'Income'
from Orders as O
INNER JOIN [Order Details] as OD on OD.OrderID = O.OrderID
INNER JOIN Menus as M on M.PosID = OD.PosID
INNER JOIN Dishes as D on D.DishID = M.DishID
INNER JOIN Categories as C on C.CategoryID = D.CategoryId
GROUP BY C.CategoryName, YEAR(O.OrderDate), MONTH(O.OrderDate)
GO
```

4.1.21. TablesStatistics - wyświetla informację ile razy dany stolik był rezerwowany

```
create View [dbo].[TablesStatistics] as
select T.TableID, COUNT(*) as 'Number of Reservations'
from Reservations as R
INNER JOIN Tables as T on T.TableID = R.TableID
GROUP BY T.TableID
GO
```

4.1.22. TablesMonthlyStatistics - wyświetla informację ile razy dany stolik był rezerwowany z podziałem na miesiące

```
create View [dbo].[TablesMonthlyStatistics] as
select T.TableID, MONTH(R.StartTime) as 'Month',
COUNT(*) as 'Number of Reservations'
from Reservations as R
INNER JOIN Tables as T on T.TableID = R.TableID
GROUP BY T.TableID, MONTH(R.StartTime)
GO
```

4.1.23. TablesYearlyStatistics - wyświetla informację ile razy dany stolik był rezerwowany z podziałem na lata

```
create View [dbo].[TablesYearlyStatistics] as
select T.TableID, YEAR(R.StartTime) as 'Year',
COUNT(*) as 'Number of Reservations'
from Reservations as R
INNER JOIN Tables as T on T.TableID = R.TableID
GROUP BY T.TableID, YEAR(R.StartTime)
GO
```

4.1.24. TablesTotalStatistics - wyświetla informację ile razy dany stolik był rezerwowany z podziałem na lata i miesiące

```
create View [dbo].[TablesTotalStatistics] as
select T.TableID, MONTH(R.StartTime) as 'Month', YEAR(R.StartTime) as 'Year',
COUNT(*) as 'Number of Reservations'
from Reservations as R
INNER JOIN Tables as T on T.TableID = R.TableID
GROUP BY T.TableID, MONTH(R.StartTime), YEAR(R.StartTime)
GO
```

4.1.25. ShowIndCustomersInformations - wyświetla informacje o klientach indywidualnych

```
create View [dbo].[ShowIndCustomersInformations] as
select IC.CustomerID, IC.Name + ' ' + IC.Surname as 'name', C.Mail, C.Phone
from IndCustomers as IC
INNER JOIN Customers as C on C.CustomerID = IC.CustomerID
GO
```

4.1.26. ShowCompaniesInformations - wyświetla informacje o firmach

```
create View [dbo].[ShowCompaniesInformations] as
select CO.CustomerID, CO.CompanyName, CO.NIP, CU.Mail, CU.Phone, COU.CountryName, CI.CityName, A.Street, A.Localnr, A.ZipCode from Companies as CO
INNER JOIN Customers as CU on CU.CustomerID  = CO.CustomerID
INNER JOIN Adresses as A on A.AdressId = CO.AdressID
INNER JOIN Cities as CI on CI.CityID = A.CityId
INNER JOIN Countries as COU on COU.CountryID = CI.CountryId
GO
```

4.1.27. CustomerStatistics - wyświetla całkowitą ilość pieniędzy wydaną przez każdego klienta

```
create View [dbo].[CustomersStatistics] as
select C.CustomerID, SUM(dbo.GetOrderPriceWithDiscount(O.OrderID)) as 'Price' from Customers as C
INNER JOIN Orders as O on O.CustomerID = C.CustomerID
GROUP BY C.CustomerID
GO
```

4.1.28. CustomerMonthlyStatistics - wyświetla całkowitą ilość pieniędzy wydaną przez każdego klienta z podziałem na miesiące

```
create View [dbo].[CustomersMonthlyStatistics] as
select C.CustomerID, MONTH(O.OrderDate) as 'Month', SUM(dbo.GetOrderPriceWithDiscount(O.OrderID)) as 'Price' from Customers as C
INNER JOIN Orders as O on O.CustomerID = C.CustomerID
GROUP BY C.CustomerID, MONTH(O.OrderDate)
GO
```

4.1.29. CustomerYearlyStatistics - wyświetla całkowitą ilość pieniędzy wydaną przez każdego klienta z podziałem na lata

```
create View [dbo].[CustomersYearlyStatistics] as
select C.CustomerID, YEAR(O.OrderDate) as 'Year', SUM(dbo.GetOrderPriceWithDiscount(O.OrderID)) as 'Price' from Customers as C
INNER JOIN Orders as O on O.CustomerID = C.CustomerID
GROUP BY C.CustomerID, YEAR(O.OrderDate)
GO
```

4.1.30. CustomerTotalStatistics - wyświetla całkowitą ilość pieniędzy wydaną przez każdego klienta z podziałem na miesiące i lata

```
create View [dbo].[CustomersTotalStatistics] as
select C.CustomerID, MONTH(O.OrderDate) as 'Month', YEAR(O.OrderDate) as 'Year', SUM(dbo.GetOrderPriceWithDiscount(O.OrderID)) as 'Price' from Customers as C
INNER JOIN Orders as O on O.CustomerID = C.CustomerID
GROUP BY C.CustomerID, MONTH(O.OrderDate), YEAR(O.OrderDate)
GO
```

4.1.31. DishesStatistics - wyświetla ilość zamówień poszczególnych dań oraz przychód jaki to danie przyniosło

```
create View [dbo].[DishesStatistics] as
select D.DishID, D.DishName, COUNT(*) as 'Number of Orders',
SUM(M.price * OD.Quantity) as 'Price'
from Orders as O
INNER JOIN [Order Details] as OD on OD.OrderID = O.OrderID
INNER JOIN Menus as M on M.PosID = OD.PosID
INNER JOIN Dishes as D on D.DishID = M.DishID
GROUP BY D.DishID, D.DishName
GO
```

4.1.32. DishesMonthlyStatistics - wyświetla ilość zamówień poszczególnych dań oraz przychód jaki to danie przyniosło z podziałem na miesiące

```
create View [dbo].[DishesMonthlyStatistics] as
select D.DishID, D.DishName, MONTH(O.OrderDate) as 'Month', COUNT(*) as 'Number of Orders',
SUM(M.price * OD.Quantity) as 'Price'
from Orders as O
INNER JOIN [Order Details] as OD on OD.OrderID = O.OrderID
INNER JOIN Menus as M on M.PosID = OD.PosID
INNER JOIN Dishes as D on D.DishID = M.DishID
GROUP BY D.DishID, D.DishName, MONTH(O.OrderDate)
GO
```

4.1.33. DishesYearlyStatistics - wyświetla ilość zamówień poszczególnych dań oraz przychód jaki to danie przyniosło z podziałem na miesiące

```
create View [dbo].[DishesYearlyStatistics] as
select D.DishID, D.DishName, YEAR(O.OrderDate) as 'Year', COUNT(*) as 'Number of Orders',
SUM(M.price * OD.Quantity) as 'Price'
from Orders as O
INNER JOIN [Order Details] as OD on OD.OrderID = O.OrderID
INNER JOIN Menus as M on M.PosID = OD.PosID
INNER JOIN Dishes as D on D.DishID = M.DishID
GROUP BY D.DishID, D.DishName, YEAR(O.OrderDate)
GO
```

4.1.34. DishesTotalStatistics - wyświetla ilość zamówień poszczególnych dań oraz przychód jaki to danie przyniosło z podziałem na lata oraz miesiące

```
create View [dbo].[DishesTotalStatistics] as
select D.DishID, D.DishName, MONTH(O.OrderDate) as 'Month', YEAR(O.OrderDate) as 'Year', COUNT(*) as 'Number of Orders',
SUM(M.price * OD.Quantity) as 'Price'
from Orders as O
INNER JOIN [Order Details] as OD on OD.OrderID = O.OrderID
INNER JOIN Menus as M on M.PosID = OD.PosID
INNER JOIN Dishes as D on D.DishID = M.DishID
GROUP BY D.DishID, D.DishName, MONTH(O.OrderDate), YEAR(O.OrderDate)
GO
```

4.1.35. ReservationsStatistics - wyświetla ilość rezerwacji w poszczególnych latach i miesiącach

```
CREATE VIEW ReservationsStatistics AS
select MONTH(StartTime) as 'Month', YEAR(StartTime) as 'Year', COUNT(*) as 'Number of Reservations' from Reservations
GROUP BY MONTH(StartTime), YEAR(StartTime)
```

4.1.36. ReservationsMonthlyStatistics - wyświetla ilość rezerwacji w danych miesiącach

```
CREATE VIEW ReservationsMonthlyStatistics as
select MONTH(StartTime) as 'Month', COUNT(*) as 'Number of Reservations' from Reservations
GROUP BY MONTH(StartTime)
```

4.1.37. ReservationsYearlyStatistics- wyświetla ilość rezerwacji w poszczególnych miesiącach

```
CREATE VIEW ReservationsYearlyStatistics as
select YEAR(StartTime) as 'Year', COUNT(*) as 'Number of Reservations' from Reservations
GROUP BY YEAR(StartTime)
```

4.1.38. OrdersStatistics - wyświetla łączny przychód za zamówienia z podziałem na lata i miesiące

```
CREATE VIEW [dbo].[OrdersStatistics] as
select MONTH(OrderDate) as 'Month', YEAR(OrderDate) as 'Year', SUM(dbo.GetOrderPriceWithDiscount(OrderID)) as 'Income' from Orders
GROUP BY MONTH(OrderDate), YEAR(OrderDate)
GO
```

4.1.39. OrdersMonthlyStatistics - wyświetla łączny przychód za zamówienia z podziałem na miesiące

```
CREATE VIEW [dbo].[OrdersMonthlyStatistics] as
select MONTH(OrderDate) as 'Month', SUM(dbo.GetOrderPriceWithDiscount(OrderID)) as 'Income' from Orders
GROUP BY MONTH(OrderDate)
GO
```

4.1.40. OrdersYearlyStatistics - wyświetla łączny przychód za zamówienia z podziałem na lata

```
CREATE VIEW [dbo].[OrdersYearlyStatistics] as
select YEAR(OrderDate) as 'Year', SUM(dbo.GetOrderPriceWithDiscount(OrderID)) as 'Income' from Orders
GROUP BY YEAR(OrderDate)
GO
```

5. Procedury

5.1.1. uspAddDish

Procedura dodaje nowe danie do tabeli Dishes

```
CREATE PROCEDURE [dbo].[uspAddDish](
@DishName varchar(50),
@CategoryName varchar(50),
@StartingUnitsInStock INT
)
AS BEGIN
    BEGIN TRY
        IF EXISTS (select * from Dishes WHERE @DishName = DishName)
        BEGIN
            ;THROW 51000, 'Dish already exist', 1;
        END

        IF NOT EXISTS(select * from Categories WHERE @CategoryName = CategoryName)
        BEGIN
            ;THROW 51000, 'This Category does not exist', 1;
        END

        DECLARE @CategoryId INT
        select @CategoryId = CategoryId from Categories WHERE @CategoryName = CategoryName

        DECLARE @DishID INT
        select @DishID = ISNULL(MAX(DishID), 0) + 1 from Dishes

        INSERT INTO Dishes(DishID, DishName, UnitsInStock, CategoryId)
        VALUES(@DishID, @DishName, @StartingUnitsInStock, @CategoryId)
    END TRY
    BEGIN CATCH
        select ERROR_MESSAGE() as ErrorMessage
    END CATCH
END
GO
```

5.1.2. uspAddOrder

Procedura dodaje nowe zamówienia do tabeli Orders

```
CREATE PROCEDURE [dbo].[uspAddOrder](
@CustomerID INT,
@Order varchar(MAX),
@Date DATETIME = NULL,
@Takeaway BIT = 0
)
AS BEGIN
    IF @Date IS NULL
    BEGIN
        SET @Date = GETDATE()
    END

    BEGIN TRY
        IF NOT EXISTS(select * from Customers WHERE @CustomerID = CustomerID)
        BEGIN
            ;THROW 51000, 'This customer does not exist', 1
        END

        DECLARE @OrderID INT
        select @OrderID = ISNULL(MAX(OrderID), 0) + 1 from Orders

        DECLARE @TableWithDishAndQuantityInString TABLE (
            ID INT,
            value varchar(MAX)
        )
        INSERT INTO @TableWithDishAndQuantityInString (ID, value)
        select ROW_NUMBER() OVER(ORDER BY value) as 'ID', value from string_split(@Order, ';')

        DECLARE @ConfiguredTable TABLE (
            ID INT,
            value varchar(MAX)
        )

        DECLARE @Counter INT
        DECLARE @Limit INT
        SET @Counter = 1
        select @Limit = COUNT(*) from @TableWithDishAndQuantityInString

        DECLARE @DAY varchar(50)
        SET @DAY = DATENAME(WEEKDAY, @Date)

        DECLARE @WholeString varchar(MAX)
        DECLARE @DishName varchar(50)
        DECLARE @Quantity INT
        DECLARE @CategoryName varchar(50)
```

```
        IF @DAY NOT IN ('Thursday', 'Friday', 'Saturday') OR GETDATE() > (SELECT DATEADD(wk, DATEDIFF(wk,0,@Date), 0))
        BEGIN
            WHILE @Counter <= @Limit
            BEGIN
                select @WholeString = value from @TableWithDishAndQuantityInString WHERE @Counter = ID

                INSERT INTO @ConfiguredTable (ID, value)
                select ROW_NUMBER() OVER(ORDER BY value) as 'ID', value from string_split(@WholeString, '_')

                select @Quantity = value from @ConfiguredTable WHERE ID = 1
                select @DishName = value from @ConfiguredTable WHERE ID = 2
                SET @CategoryName = dbo.GetDishCategory(@DishName)

                IF @CategoryName = 'Owoce morza'
                BEGIN
                    ;THROW 51000, 'Seafood can be order only for Thursday, Friday, Saturday and you habe to order till Monday', 1
                END

                SET @Counter = @Counter + 1
            END
        END

        SET @Counter = 1

        IF @Takeaway = 1
        BEGIN
            INSERT INTO Orders(OrderID, CustomerID, Takeaway, OrderDate, PickUpDate)
            VALUES(@OrderID, @CustomerID, @Takeaway, GETDATE(), @Date)
        END
        ELSE BEGIN
            INSERT INTO Orders(OrderID, CustomerID, Takeaway, OrderDate, PickUpDate)
            VALUES(@OrderID, @CustomerID, @Takeaway, GETDATE(), NULL)
        END

        WHILE @Counter <= @Limit
        BEGIN
            select @WholeString = value from @TableWithDishAndQuantityInString WHERE @Counter = ID

            INSERT INTO @ConfiguredTable (ID, value)
            select ROW_NUMBER() OVER(ORDER BY value) as 'ID', value from string_split(@WholeString, '_')

            select @Quantity = value from @ConfiguredTable WHERE ID = 1
            select @DishName = value from @ConfiguredTable WHERE ID = 2

            EXEC uspAddToOrderDetails @OrderID, @DishName, @Quantity

            SET @Counter = @Counter + 1
        END

        IF (select COUNT(OrderID) from [Order Details] WHERE OrderID = @OrderID) < @Limit
        BEGIN
            EXEC uspDeleteOrder @OrderID
            ;THROW 51000, 'Out of units in stock', 1
        END
    END TRY
    BEGIN CATCH
    select ERROR_MESSAGE() as ErrorMessage
    END CATCH
END
GO
```

5.1.3. uspAddPosToMenu

Procedura dodaje nową pozycję do aktualnego menu

```
CREATE Procedure [dbo].[uspAddPosToMenu](
@DishID INT,
@Price MONEY
)
AS BEGIN
    BEGIN TRY
        DECLARE @DishName varchar(50)
        select @DishName = DishName from Dishes WHERE @DishID = DishID

        IF @@ROWCOUNT = 0
        BEGIN
            ;THROW 51000, 'Dish with that ID does not exist', 1
        END

        DECLARE @PosID INT
        select @PosID = ISNULL(MAX(PosID), 0) + 1 from Menus

        INSERT INTO Menus(DishID, ValidFrom, ValidTo, PosID, price)
        VALUES(@DishID, GETDATE(), NULL, @PosID, @Price)
    END TRY
    BEGIN CATCH
        select ERROR_MESSAGE() as ErrorMessage
    END CATCH
END
GO
```

5.1.4. uspAddTables

Procedura dodaje nowy stolik do tabeli Tables

```
Create PROCEDURE [dbo].[uspAddTables](
@NumberOfChairs int
)
as begin
    begin try
        Declare @TableID INT;
        select @TableID = ISNULL(MAX(TableID), 0) + 1 from Tables

        insert into Tables(TableID, NumberOfChairs)
        values (@TableID, @NumberOfChairs)
    end try
    begin catch
        select ERROR_MESSAGE() as ErrorMessage
    end catch
end
GO
```

5.1.5. uspAddToOrderDetails

Procedura dodaje szczegółowe informacje o zamówieniu do tabeli Order Details.

```
CREATE PROCEDURE [dbo].[uspAddToOrderDetails](
@OrderID INT,
@DishName varchar(50),
@Quantity INT
)
AS BEGIN
    BEGIN TRY
        IF NOT EXISTS(select * from Orders WHERE @OrderID = OrderID)
        BEGIN
            ;THROW 51000, 'This order does not exist', 1
        END

        DECLARE @UnitsInStock INT
        select @UnitsInStock = UnitsInStock from Dishes WHERE @DishName = DishName

        IF @Quantity > @UnitsInStock
        BEGIN
            ;THROW 51000, 'There is not enough units in stock', 1
        END

        DECLARE @PosID INT
        select @PosID = PosID from Dishes as D
        INNER JOIN Menus as M on M.DishID = D.DishID
        WHERE @DishName = D.DishName AND M.ValidTo IS NULL

        IF @PosID IS NULL
        BEGIN
            ;THROW 51000, 'This dish is not in current menu', 1
        END

        DECLARE @DishID INT
        select @DishID = DishID from Dishes WHERE @DishName = DishName

        DECLARE @UpdatedUnitsInStock INT
        SET @UpdatedUnitsInStock = (@UnitsInStock - @Quantity)

        EXEC uspUpdateUnitsInStock @DishID, @UpdatedUnitsInStock

        INSERT INTO [Order Details](PosID, OrderID, Quantity)
        VALUES(@PosID, @OrderID, @Quantity)
    END TRY
    BEGIN CATCH
        select ERROR_MESSAGE() as ErrorMessage
    END CATCH
END
GO
```

5.1.6. uspChangeNumberOfChairs

Procedura aktualizuje liczbę miejsc przypisaną do danego stolika

```
Create Procedure [dbo].[uspChangeNumberOfChairs](
@NumberOfChairs INT,
@TableID INT
)
AS BEGIN
    BEGIN TRY
        IF NOT EXISTS(select * from Tables WHERE @TableID = TableID)
        BEGIN
            ;THROW 51000, 'Table with that ID do not exist', 1
        END

        UPDATE Tables
        SET NumberOfChairs = @NumberOfChairs
        WHERE TableID = @TableID
    END TRY
    BEGIN CATCH
        select ERROR_MESSAGE() as ErrorMessage
    END CATCH
END
GO
```

5.1.7. uspDeleteTable

Procedura usuwa dany stolik z tabeli Tables

```
CREATE PROCEDURE [dbo].[uspDeleteTable](
@TableID INT
)
AS BEGIN
    BEGIN TRY
        IF NOT EXISTS(select * from Tables WHERE @TableID = TableID)
        BEGIN
            ;THROW 51000, 'Table with this ID do not exist', 1
        END

        DELETE from Tables WHERE TableID = @TableID
    END TRY
    BEGIN CATCH
        select ERROR_MESSAGE() as ErrorMessage
    END CATCH
END
GO
```

5.1.8. uspRemoveDishFromCurrentMenu

Procedura ustawia daną pozycję w Menu jako nieaktualną.

```
CREATE PROCEDURE [dbo].[uspRemoveDishFromCurrentMenu](
@PosID INT
)
AS BEGIN
    BEGIN TRY
        IF NOT EXISTS(select * from Menus WHERE @PosID = PosID)
        BEGIN
            ;THROW 51000, 'There is not such position in menu', 1
        END

        UPDATE Menus SET ValidTo = GETDATE() WHERE PosID = @PosID
    END TRY
    BEGIN CATCH
        select ERROR_MESSAGE() as ErrorMessage
    END CATCH
END
GO
```

5.1.9. uspConfirmReservation

Procedura potwierdza rezerwację

```
CREATE PROCEDURE [dbo].[uspConfirmReservation](
@ReservationID INT
)
AS BEGIN
    BEGIN TRY
        IF NOT EXISTS(select * from Reservations WHERE @ReservationID = ReservationID)
        BEGIN
            ;THROW 51000, 'Reservation with that id do not exist', 1
        END

        DECLARE @CustomerID INT
        DECLARE @R1 DECIMAL(18,0)
        DECLARE @R2 DECIMAL(18,0)
        DECLARE @OrderID INT
        DECLARE @OrderDate DATETIME

        select @OrderID = OrderID from IndReservations WHERE @ReservationID = ReservationID
        select @OrderDate = OrderDate from Orders WHERE @OrderID = OrderID
        select @CustomerID = IndID from IndReservations
        select @R1 = R1 from dbo.GetPermanentDiscount(@OrderDate, @CustomerID)
        select @R2 = R2 from dbo.GetTemporaryDiscount(@OrderDate, @CustomerID)

        IF @R2 > @R1
        BEGIN
            DELETE FROM TemporaryDiscounts WHERE @CustomerID = CustomerID
        END

        DECLARE @TableID INT
        DECLARE @StartTime DATETIME
        DECLARE @EndTime DATETIME
        DECLARE @NumberOfChairs INT

        select @StartTime = StartTime, @EndTime = EndTime  from Reservations WHERE @ReservationID = ReservationID
        select @NumberOfChairs = NumberOfPeople from IndReservations WHERE @ReservationID = ReservationID
        SET @TableID = dbo.FreeTable(@StartTime, @EndTime, @NumberOfChairs)

        IF @TableID IS NULL
        BEGIN
            ;THROW 51000, 'There are no free tables', 1
        END

        UPDATE Reservations
        SET isConfirmed = 1, TableID = @TableID
        WHERE ReservationID = @ReservationID
    END TRY
    BEGIN CATCH
        select ERROR_MESSAGE() as ErrorMessage
    END CATCH
END
GO
```

5.1.10. uspUpdateWZandWK

Procedura aktualizuje wartości WZ oraz WK

```
CREATE PROCEDURE [dbo].[uspUpdateWZandWK](
@WZ INT,
@WK INT
)
AS BEGIN
    UPDATE WZandWKs SET WZ = @WZ, WK = @WK WHERE ID = 1
END
GO
```

5.1.11. uspUpdateDiscountParameters

Procedura ustawia nowe wartości dla parametrów Z1, K1, D1, R1, R2, K2

```
CREATE PROCEDURE [dbo].[uspUpdateDiscountParameters](
@Z1 INT,
@K1 MONEY,
@D1 INT,
@R1 DECIMAL(18, 0),
@R2 DECIMAL(18, 0),
@K2 MONEY
)
AS BEGIN
    BEGIN TRY
        DECLARE @CurrentActiveID INT
        select @CurrentActiveID = ID from DiscountInformations WHERE DateStop IS NULL

        IF @CurrentActiveID IS NOT NULL
        BEGIN
            UPDATE DiscountInformations SET DateStop = GETDATE() WHERE @CurrentActiveID = ID
        END

        DECLARE @ID INT
        select @ID = ISNULL(MAX(ID), 0) + 1 from DiscountInformations

        INSERT INTO DiscountInformations(ID, DateStart, DateStop, Z1, K1, D1, R1, R2, K2)
        VALUES(@ID, GETDATE(), NULL, @Z1, @K1, @D1, @R1, @R2, @K2)
    END TRY
    BEGIN CATCH
        select ERROR_MESSAGE() as ErrorMessage
    END CATCH
END
GO
```

5.1.12. uspAddNewCategory

Procedura dodaje nową kategorię do tabeli Categories

```
CREATE PROCEDURE [dbo].[uspAddNewCategory](
@CategoryName varchar(50)
)
AS BEGIN
    BEGIN TRY
        IF EXISTS(select * from Categories WHERE @CategoryName = CategoryName)
        BEGIN
            ;THROW 51000, 'This Category already exist', 1
        END

        DECLARE @CategoryID INT
        select @CategoryID = ISNULL(MAX(CategoryID), 0) + 1 from Categories

        INSERT INTO Categories(CategoryID, CategoryName)
        VALUES(@CategoryID, @CategoryName)
    END TRY
    BEGIN CATCH
        select ERROR_MESSAGE() as ErrorMessage
    END CATCH
END
GO
```

5.1.13. uspGrantPermanentDiscount

Procedura dodaje klientowi indywidualnemu zniżkę stała jeśli spełnia wymagania

```
CREATE PROCEDURE [dbo].[uspGrantPermanentDiscount](
@CustomerID INT
)
AS BEGIN
    BEGIN TRY
        IF NOT EXISTS(select * from IndCustomers WHERE @CustomerID = CustomerID)
        BEGIN
            ;THROW 51000, 'Customer do not exist', 1
        END

        IF EXISTS(select * from PermanentDiscounts WHERE @CustomerID = CustomerID)
        BEGIN
            ;THROW 51000, 'The customer has already been granted a discount', 1
        END

        DECLARE @Z1 INT
        DECLARE @K1 MONEY
        DECLARE @CurrentZ1 INT
        DECLARE @CurrentK1 MONEY

        select @Z1 = CustomerID from NumberOfOrders WHERE @CustomerID = CustomerID
        select @K1 = MIN(Cost) from OrderValuePreDistcount WHERE @CustomerID = CustomerID
        select @CurrentZ1 = Z1, @CurrentK1 = K1 from CurrentDiscountsInfo

        IF @Z1 < @CurrentZ1 OR @K1 < @CurrentK1
        BEGIN
            ;THROW 51000, 'The customer does not meet the requirements', 1
        END

        DECLARE @DiscountID INT
        select @DiscountID = ISNULL(MAX(DiscountID), 0) + 1 from PermanentDiscounts

        INSERT INTO PermanentDiscounts(DiscountID, ValidFrom, CustomerID)
        VALUES(@DiscountID, GETDATE(), @CustomerID)
    END TRY
    BEGIN CATCH
        select ERROR_MESSAGE() as ErrorMessagwe
    END CATCH
END
GO
```

5.1.14. uspGrantTemporaryDiscount

Procedura dodaje klientowi indywidualnemu zniżkę tymczasową jeśli spełnia wymagania

```
CREATE PROCEDURE [dbo].[uspGrantTemporaryDiscount](
@CustomerID INT
)
AS BEGIN
    BEGIN TRY
        IF NOT EXISTS(select * from IndCustomers WHERE @CustomerID = CustomerID)
        BEGIN
            ;THROW 51000, 'Customer do not exist', 1
        END

        DECLARE @K2 INT
        DECLARE @CurrentK2 INT
        DECLARE @CurrentD1 INT

        select @K2 = Price from CustomersStatistics WHERE @CustomerID = CustomerID
        select @CurrentD1 = D1, @CurrentK2 = K2 from CurrentDiscountsInfo

        IF @K2 < @CurrentK2
        BEGIN
            ;THROW 51000, 'The customer does not meet the requirements', 1
        END

        DECLARE @DiscountID INT
        select @DiscountID = ISNULL(MAX(DiscountID), 0) + 1 from TemporaryDiscounts

        INSERT INTO TemporaryDiscounts(DiscountID, ValidFrom, ValidTo, CustomerID)
        VALUES(@DiscountID, GETDATE(), DATEADD(DAY, @CurrentD1, GETDATE()), @CustomerID)
    END TRY
    BEGIN CATCH
        select ERROR_MESSAGE() as ErrorMessage
    END CATCH
END
GO
```

5.1.15. uspAddCompanyReservationDetails

Procedura dodaje szczegóły dla rezerwacji firmowej

```
CREATE PROCEDURE [dbo].[uspAddCompanyReservationDetails](
@Employees varchar(MAX),
@ReservationID INT,
@CompanyID INT
)
AS BEGIN
    BEGIN TRY
        IF NOT EXISTS(select * from Customers WHERE @CompanyID = CustomerID)
        BEGIN
            ;THROW 51000, 'This company does not exist', 1
        END

        DECLARE @TableWithNonConfiguresStrings TABLE (
            ID INT,
            value varchar(MAX)
        )
        INSERT INTO @TableWithNonConfiguresStrings (ID, value)
        select ROW_NUMBER() OVER(ORDER BY value) as 'ID', value from string_split(@Employees, ';')

        DECLARE @ConfiguredTable TABLE (
            ID INT,
            value varchar(MAX)
        )

        DECLARE @Counter INT
        DECLARE @Limit INT
        SET @Counter = 1
        select @Limit = COUNT(*) from @TableWithNonConfiguresStrings

        DECLARE @WholeString varchar(MAX)
        DECLARE @Name varchar(50)
        DECLARE @Surname varchar(50)
        DECLARE @Mail varchar(50)
        DECLARE @CustomerID INT

        WHILE @Counter <= @Limit
        BEGIN
            select @WholeString = value from @TableWithNonConfiguresStrings WHERE @Counter = ID

            INSERT INTO @ConfiguredTable (ID, value)
            select ROW_NUMBER() OVER(ORDER BY (select 1)) as ID, value from string_split(@WholeString, '_')

            select @Mail = value from @ConfiguredTable WHERE ID = 3
            select @Surname = value from @ConfiguredTable WHERE ID = 2
            select @Name = value from @ConfiguredTable WHERE ID = 1

            EXEC uspAddIndCustomer @Name, @Surname, @Mail, @CompanyID

            select @CustomerID = ISNULL(MAX(CustomerID), 0) from IndCustomers
            INSERT INTO CompanyReservationsDetails(EmployeeID, ReservationID)
            VALUES(@CustomerID, @ReservationID)

            SET @Counter = @Counter + 1
        END
    END TRY
    BEGIN CATCH
        select ERROR_MESSAGE() as ErrorMessage
    END CATCH
END
GO
```

5.1.16. uspAddCompanyReservationWithEmployeesNames

```
CREATE PROCEDURE [dbo].[uspAddCompanyReservationWithEmployeesNames](
@CustomerID INT,
@NumberOfPeople INT,
@StartTime DATETIME,
@EndTime DATETIME,
@Employees varchar(MAX)
)
AS BEGIN
    BEGIN TRY
        IF NOT EXISTS(select * from Companies WHERE @CustomerID = CustomerID)
        BEGIN
            ;THROW 51000, 'Customer does not exist', 1
        END

        IF @NumberOfPeople < 2
        BEGIN
            ;THROW 51000, 'Minimum number of people for reservation is 2', 1
        END

        DECLARE @COUNTER INT

        select @COUNTER = COUNT(*) from string_split(@Employees, ';')

        IF @COUNTER != @NumberOfPeople
        BEGIN
            ;THROW 51000, 'Number of people must equals number of employees', 1
        END

        DECLARE @ReservationID INT
        select @ReservationID = ISNULL(MAX(ReservationID), 0) + 1 from Reservations

        INSERT INTO Reservations(ReservationID, TableID, StartTime, EndTime, isConfirmed)
        VALUES(@ReservationID, NULL, @StartTime, @EndTime, 0)

        INSERT INTO CompanyReservations(CustomerID, ReservationID, NumberOfPeople)
        VALUES(@CustomerID, @ReservationID, @NumberOfPeople)

        EXEC uspAddCompanyReservationDetails @Employees, @ReservationID, @CustomerID
    END TRY
    BEGIN CATCH
        select ERROR_MESSAGE() as ErrorMessage
    END CATCH
END
GO
```

5.1.17. uspAddCountry

Procedura dodaje nowy kraj do tabeli Countries

```
CREATE PROCEDURE [dbo].[uspAddCountry](
@CountryName varchar(50)
)
AS BEGIN
    BEGIN TRY
        IF EXISTS(select * from Countries WHERE @CountryName = CountryName)
        BEGIN
            ;THROW 51000, 'This country already exist', 1
        END

        DECLARE @CountryID INT
        select @CountryID = ISNULL(MAX(CountryID), 0) + 1 from Countries

        INSERT INTO Countries(CountryID, CountryName)
        VALUES(@CountryID, @CountryName)
    END TRY
    BEGIN CATCH
        select ERROR_MESSAGE() as ErrorMessage
    END CATCH
END
GO
```

5.1.18. uspAddCity

Procedura dodaje miasto do tabeli Cities

```
CREATE PROCEDURE [dbo].[uspAddCity](
@CityName varchar(50),
@CountryName varchar(50)
)
AS BEGIN
    BEGIN TRY
        IF NOT EXISTS(select * from Countries WHERE @CountryName = CountryName)
        BEGIN
            EXEC uspAddCountry @CountryName
        END

        IF EXISTS(select * from Cities WHERE @CityName = CityName)
        BEGIN
            ;THROW 51000, 'This city already exist', 1
        END

        DECLARE @CityID INT
        DECLARE @CountryId INT

        select @CityID = ISNULL(MAX(CityID), 0) + 1 from Cities
        select @CountryId = CountryID from Countries WHERE @CountryName = CountryName

        INSERT INTO Cities(CityID, CountryId, CityName)
        VALUES(@CityID, @CountryId, @CityName)
    END TRY
    BEGIN CATCH
        select ERROR_MESSAGE() as ErrorMessage
    END CATCH
END
GO
```

5.1.19. uspAddAddress

Procedura dodaje nowy adres do tabeli Addresses

```
CREATE PROCEDURE [dbo].[uspAddAddress](
@CityName varchar(50),
@CountryName varchar(50),
@ZipCode nchar(10),
@Localnr nchar(10),
@Street varchar(50)
)
AS BEGIN
    BEGIN TRY
        IF NOT EXISTS(select * from Cities WHERE @CityName = CityName)
        BEGIN
            EXEC uspAddCity @CityName, @CountryName
        END

        DECLARE @CityId INT
        DECLARE @AdressID INT

        select @CityId = CityID from Cities WHERE @CityName = CityName
        select @AdressID = ISNULL(MAX(AdressId), 0) + 1 from Adresses

        INSERT INTO Adresses(AdressId, CityId, ZipCode, Localnr, Street)
        VALUES(@AdressID, @CityId, @ZipCode, @Localnr, @Street)
    END TRY
    BEGIN CATCH
        select ERROR_MESSAGE() as ErrorMessage
    END CATCH
END
GO
```

5.1.20. uspAddIndCustomer

Procedura dodaje nowego klienta indywidualnego

```
CREATE PROCEDURE [dbo].[uspAddIndCustomer](
@Name varchar(50),
@Surname varchar(50),
@Mail varchar(50),
@Phone varchar(50) = NULL,
@CompanyID INT = NULL
)
AS BEGIN
    BEGIN TRY
        DECLARE @CustomerID INT
        select @CustomerID = ISNULL(MAX(CustomerID), 0) + 1 from Customers

        INSERT INTO Customers(CustomerID, Phone, Mail)
        VALUES(@CustomerID, @Phone, @Mail)

        INSERT INTO IndCustomers(CustomerID, Name, Surname, CompanyID)
        VALUES(@CustomerID, @Name, @Surname, @CompanyID)
    END TRY
    BEGIN CATCH
        select ERROR_MESSAGE() as ErrorMessage
    END CATCH
END
GO
```

5.1.21. uspAddCompanyCustomer

Procedura dodaje firmę do tabeli Companies

```
CREATE PROCEDURE [dbo].[uspAddCompanyCustomer](
@CompanyName varchar(50),
@NIP varchar(255),
@Mail varchar(50),
@CityName varchar(50),
@CountryName varchar(50),
@ZipCode nchar(10),
@Localnr nchar(10),
@Street varchar(50),
@Phone varchar(50) = NULL
)
AS BEGIN
    BEGIN TRY
        IF EXISTS(select * from Companies WHERE @CompanyName = CompanyName)
        BEGIN
            ;THROW 51000, 'This company already exists', 1
        END

        EXEC uspAddAddress @CityName, @CountryName, @ZipCode, @Localnr, @Street

        DECLARE @AdressID INT
        select @AdressID = MAX(AdressId) from Adresses

        DECLARE @CustomerID INT
        select @CustomerID = ISNULL(MAX(CustomerID), 0) + 1 from Customers

        INSERT INTO Customers(CustomerID, Phone, Mail)
        VALUES(@CustomerID, @Phone, @Mail)

        INSERT INTO Companies(NIP, AdressID, CompanyName, CustomerID)
        VALUES(@NIP, @AdressID, @CompanyName, @CustomerID)
    END TRY
    BEGIN CATCH
        select ERROR_MESSAGE() as ErrorMessage
    END CATCH
END
GO
```

5.1.22. uspAddIndReservation

Procedura dodaje rezerwację dla klienta indywidualnego

```
CREATE PROCEDURE [dbo].[uspAddIndReservation](
@NumberOfPeople INT,
@CustomerID INT,
@Order VARCHAR(MAX),
@StartTime DATETIME,
@EndTime DATETIME
)

AS BEGIN
    BEGIN TRY
        IF NOT EXISTS(select * from IndCustomers WHERE @CustomerID = CustomerID)
        BEGIN
            ;THROW 51000, 'Customer does not exist',1
        END

        IF @NumberOfPeople < 2
        BEGIN
            ;THROW 51000, 'Minimum number of people for reservation is 2', 1
        END

        DECLARE @NumberOfOrders INT
        DECLARE @CurrentWZ MONEY
        DECLARE @CurrentWK INT
        DECLARE @OrderCost MONEY
        DECLARE @ReservationID INT
        DECLARE @R1 DECIMAL(18,0)
        DECLARE @R2 DECIMAL(18,0)
        DECLARE @DiscountValue DECIMAL(18,0)

        select @NumberOfOrders = [Orders number] from NumberOfOrders WHERE @CustomerID = CustomerID
        select @CurrentWZ = WZ, @CurrentWK = WK from WZandWKs
        select @R1 = R1 from dbo.GetPermanentDiscount(GETDATE(), @CustomerID)
        select @R2 = R2 from dbo.GetTemporaryDiscount(GETDATE(), @CustomerID)

        IF @R1 > @R2
        BEGIN
            SET @DiscountValue = @R1
        END
        ELSE
        BEGIN
            SET @DiscountValue = @R2
        END

        DECLARE @PreOrderID INT
        select @PreOrderID = ISNULL(MAX(OrderID), 0) from Orders
```

```
        EXEC uspAddOrder @CustomerID, @Order, @StartTime
        DECLARE @OrderID INT
        select @OrderID = ISNULL(MAX(OrderID), 0) from Orders

        IF @PreOrderID = @OrderID
        BEGIN
            ;THROW 51000, 'Cannot add that order', 1
        END

        select @OrderCost = Cost from OrderValuePreDistcount WHERE @OrderID = OrderID
        SET @OrderCost = @OrderCost * (100-@DiscountValue)/100

        IF @OrderCost < @CurrentWZ OR @NumberOfOrders < @CurrentWK
        BEGIN
            EXEC uspDeleteOrder @OrderID
            ;THROW 51000, 'Customer doesnt meet the requirements', 1
        END

        select @ReservationID = ISNULL(MAX(ReservationID),0) + 1 from Reservations

        INSERT INTO Reservations(ReservationID, TableID, StartTime, EndTime, isConfirmed)
        VALUES (@ReservationID, NULL, @StartTime, @EndTime,0)

        INSERT INTO IndReservations(ReservationID, OrderID, IndID, NumberOfPeople)
        VALUES(@ReservationID, @OrderID, @CustomerID, @NumberOfPeople)

    END TRY
    BEGIN CATCH
        select ERROR_MESSAGE() as ErrorMessage
    END CATCH
END
GO
```

5.1.23. uspAddCompanyReservation

Procedura dodaje rezerwację dla klienta firmowego

```
CREATE PROCEDURE [dbo].[uspAddCompanyReservation](
@CustomerID INT,
@NumberOfPeople INT,
@StartTime DATETIME,
@EndTime DATETIME
)
AS BEGIN
    BEGIN TRY
        IF NOT EXISTS(select * from Companies WHERE @CustomerID = CustomerID)
        BEGIN
            ;THROW 51000, 'Customer does not exist', 1
        END

        IF @NumberOfPeople < 2
        BEGIN
            ;THROW 51000, 'Minimum number of people for reservation is 2', 1
        END

        DECLARE @ReservationID INT
        select @ReservationID = ISNULL(MAX(ReservationID), 0) + 1 from Reservations

        INSERT INTO Reservations(ReservationID, TableID, StartTime, EndTime, isConfirmed)
        VALUES(@ReservationID, NULL, @StartTime, @EndTime, 0)

        INSERT INTO CompanyReservations(CustomerID, ReservationID, NumberOfPeople)
        VALUES(@CustomerID, @ReservationID, @NumberOfPeople)
    END TRY
    BEGIN CATCH
        select ERROR_MESSAGE() as ErrorMessage
    END CATCH
END
GO
```

5.1.24. uspUpdateUnitsInStock

Procedura aktualizuje ilość określonego dania w magazynie

```
CREATE PROCEDURE [dbo].[uspUpdateUnitsInStock](
@DishID INT,
@UnitsInStock INT
)
AS BEGIN
    BEGIN TRY
        IF NOT EXISTS(select * from Dishes WHERE @DishID = DishID)
        BEGIN
            ;THROW 51000, 'Dish with that ID does not exist', 1
        END

        UPDATE Dishes
        SET UnitsInStock = @UnitsInStock
        WHERE @DishID = DishID
    END TRY
    BEGIN CATCH
        select ERROR_MESSAGE() as ErrorMessage
    END CATCH
END
GO
```

5.1.25. uspDeleteOrder

Procedura usuwa zamówienie i połączone z nim szczegóły zamówienia oraz rezerwację

```
CREATE PROCEDURE [dbo].[uspDeleteOrder](
@OrderID INT
)
AS BEGIN
    BEGIN TRY
        DECLARE @CustomerID INT
        select @CustomerID = CustomerID from Orders WHERE @OrderID = OrderID
        DELETE FROM [Order Details] WHERE @OrderID = OrderID

        IF EXISTS(select * from IndReservations WHERE @CustomerID = IndID AND @OrderID = OrderID)
        BEGIN
            DECLARE @ReservationID INT
            select @ReservationID = ReservationID from IndReservations WHERE @OrderID = OrderID

            DELETE FROM IndReservations WHERE @OrderID = OrderID
            DELETE FROM Reservations WHERE @ReservationID = ReservationID
        END

        DELETE FROM Orders WHERE @OrderID = OrderID
    END TRY
    BEGIN CATCH
        select ERROR_MESSAGE() as ErrorMessage
    END CATCH
END
GO
```

6. Funkcje

6.1.1. TakenTables

Funkcja zwraca tabelę ID stolików które są zajęte w danym przedziale czasowym

```
CREATE FUNCTION [dbo].[TakenTables](
@StartTime DATETIME,
@EndTime DATETIME
)
returns table as return
select Tables.TableID from Tables
INNER JOIN Reservations ON Reservations.TableID = Tables.TableID
WHERE
((@StartTime >= StartTime AND @EndTime <= EndTime) OR
(@StartTime <= StartTime AND @EndTime <= EndTime AND StartTime < @EndTime) OR
(@StartTime >= StartTime AND @EndTime >= EndTime AND @StartTime < EndTime) OR
(@StartTime < StartTime AND @EndTime > EndTime))
GO
```

6.1.2. FreeTables

Funkcja zwraca tabelę ID wolnych stolików w danym przedziale czasowym

```
CREATE FUNCTION [dbo].[FreeTables](
@StartTime DATETIME,
@EndTime DATETIME,
@NumberOfChairs INT
)
returns table as return
select TableID, NumberOfChairs from Tables
WHERE @NumberOfChairs <= NumberOfChairs AND
TableID NOT IN (select * from TakenTables(@StartTime, @EndTime))
GO
```

6.1.3. FreeTable

Funkcja zwraca ID wolnego stolika w danym przedziale czasowym

```
CREATE FUNCTION [dbo].[FreeTable](
@StartTime DATETIME,
@EndTime DATETIME,
@NumberOfChairs INT
)
returns int as begin
return (select TOP 1 TableID from FreeTables(@StartTime, @EndTime, @NumberOfChairs) ORDER BY NumberOfChairs ASC)
END
GO
```

6.1.4. GetIncome

Funkcja zwraca łączny przychód w danym roku i miesiącu

```
CREATE FUNCTION [dbo].[GetIncome](
@Year INT,
@Month INT
)
returns money as begin
return (select Income from OrdersStatistics
WHERE @Year = Year AND @Month = Month)
END
GO
```

6.1.5. GetTableNumberOfReservations

Funkcja zwraca ilość zamówień dla danego stolika w określonym roku i miesiącu

```
CREATE FUNCTION [dbo].[GetTableNumberOfReservations](
@Year INT,
@Month INT,
@TableID INT
)
returns int as begin
return (select [Number of Reservations] from TablesTotalStatistics
WHERE @Year = Year AND @Month = Month AND @TableID = TableID)
END
GO
```

6.1.6. GetOrderDetails

Funkcja zwraca identyfikator klienta, nazwę produktu i jej ilość na podany numer zamówienia

```
CREATE FUNCTION [dbo].[GetOrderDetails](
@OrderID INT
)
returns table as return
select O.OrderID, CustomerID, OrderDate, DishName, Quantity from Orders as O
INNER JOIN [Order Details] as OD on OD.OrderID = O.OrderID
INNER JOIN Menus as M on M.PosID = OD.PosID
INNER JOIN Dishes as D on D.DishID = M.DishID
WHERE @OrderID = O.OrderID
GO
```

6.1.7. GetNumberOfDishorders

Funkcja zwraca ilość zamówień danego dania w określonym roku i miesiącu

```
CREATE FUNCTION [dbo].[GetNumberOfDishOrders](
@DishName varchar(50),
@Year INT,
@Month INT
)
returns int as begin
return (select [Number of Orders] from DishesTotalStatistics
WHERE @DishName = DishName AND @Year = Year AND @Month = Month)
END
GO
```

6.1.8. GetCustomerOrders

Funkcja zwraca wszystkie zamówienia wraz z ceną złożone przez określonego klienta

```
CREATE FUNCTION [dbo].[GetCustomerOrders](
@CustomerID INT
)
returns table as return
select CustomerID, OrderID, Cost, OrderDate from OrderValuePreDistcount
WHERE @CustomerID = CustomerID
GO
```

6.1.9. GetTemporaryDiscount

Funkcja zwraca zniżkę tymczasową dla określonego klienta

```
CREATE FUNCTION [dbo].[GetTemporaryDiscount](
@Date DATE,
@CustomerID INT
)
returns table as return
select
*,
(select TOP 1 R2 from DiscountInformations as DI
WHERE @Date >= DI.DateStart AND (DI.DateStop IS NULL OR @Date <= DI.DateStop)
ORDER BY R2 DESC) as 'R2'
from TemporaryDiscounts as TD
WHERE @CustomerID = CustomerID AND @Date >= TD.ValidFrom AND (TD.ValidTo IS NULL OR @Date <= TD.ValidTo)
GO
```

6.1.10. GetPermanentDiscount

Funkcja zwraca zniżkę stałą dla określonego klienta

```
CREATE FUNCTION [dbo].[GetPermanentDiscount](
@Date DATE,
@CustomerID INT
)
returns table as return
select
*,
(select TOP 1 R1 from DiscountInformations as DI
WHERE @Date >= DI.DateStart AND (DI.DateStop IS NULL OR @Date <= DI.DateStop)
ORDER BY R1 DESC) as 'R1'
from PermanentDiscounts as PD
WHERE @CustomerID = CustomerID AND @Date >= PD.ValidFrom
GO
```

6.1.11. GetOrderPriceWithDiscount

Funkcja zwraca wartość określonego zamówienia po nałożeniu zniżki

```
CREATE FUNCTION [dbo].[GetOrderPriceWithDiscount](
@OrderID INT
)
returns float as begin
    DECLARE @R1 DECIMAL(18,0)
    DECLARE @R2 DECIMAL(18,0)
    DECLARE @CustomerID INT
    DECLARE @Price MONEY
    DECLARE @OrderDate DATETIME

    select @CustomerID = CustomerID, @OrderDate = OrderDate from Orders WHERE @OrderID = OrderID
    select @R1 = R1 from dbo.GetPermanentDiscount(@OrderDate, @CustomerID)
    select @R2 = R2 from dbo.GetTemporaryDiscount(@OrderDate, @CustomerID)
    select @Price = Cost from OrderValuePreDistcount WHERE @OrderID = OrderID

    IF @R1 IS NULL AND @R2 IS NULL
    BEGIN
        return @Price
    END

    IF ISNULL(@R1, 0) > ISNULL(@R2, 0)
    BEGIN
        return @Price * (100 - ISNULL(@R1, 0)) / 100
    END

    return @Price * (100 - ISNULL(@R2, 0)) / 100
END
GO
```

6.1.12. GenerateInvoice

Funkcja generuje fakturę dla określonego zamówienia

```
CREATE FUNCTION [dbo].[GenerateInvoice](
@OrderID INT
)
returns table as return
select
*,
(select Cost from OrderValuePreDistcount WHERE @OrderID = OrderID) as 'Total price without discount',
(select dbo.GetOrderPriceWithDiscount(@OrderID)) as 'Total price with discount'
from GetOrderDetails(@OrderID)
GO
```

6.1.13. isMenuValidate

Funkcja zwraca 0 jeśli trzeba poprawić menu lub 1 jeśli przynajmniej połowa pozycji w Menu została już zamieniona

```
CREATE FUNCTION [dbo].[isMenuValidate]()
returns BIT as BEGIN
    DECLARE @MenuSize INT
    SET @MenuSize = (select COUNT(*) from Menus WHERE ValidTo IS NULL)
    DECLARE @FreshPositions INT
    SET @FreshPositions = (select COUNT(*) from Menus WHERE ValidTo IS NULL AND DATEDIFF(DAY, ValidFrom, GETDATE()) < 14)

    IF @FreshPositions < @MenuSize / 2
    BEGIN
        RETURN 0
    END

    RETURN 1
END
GO
```

6.1.14. GetDishCategory

```
CREATE FUNCTION [dbo].[GetDishCategory](
@DishName varchar(50)
)
returns varchar(50) as BEGIN
return (select C.CategoryName from Dishes as D
INNER JOIN Categories as C on C.CategoryID = D.CategoryId
WHERE @DishName = D.DishName)
END
GO
```

6.1.15. GetInformationAboutIndCustomer

Funkcja zwraca informacje o określonym kliencie indywidualnym

```
CREATE FUNCTION [dbo].[GetInformationAboutIndCustomer](
@CustomerID INT
)
returns table as return
select * from ShowIndCustomersInformations
WHERE @CustomerID = CustomerID
GO
```

6.1.16. GetInformationAboutCompany

Funkcja zwraca informacje o określonym kliencie firmowym

```
CREATE FUNCTION [dbo].[GetInformationAboutCompany](
@CustomerID INT
)
returns table as return
select * from ShowCompaniesInformations
WHERE @CustomerID = CustomerID
GO
```

7. Indeksy

7.1. ix_customers_name

```
create index ix_customers_name
on IndCustomers(surname,name)
```

7.2. ix_orders_dates

```
create index ix_orders_dates on Orders(orderdate,pickupdate)
```

7.3. ix_menu_dates

```
create index ix_menu_dates on Menus(validFrom,validTo)
```

7.4. ix_menu_prices

```
create index ix_menu_prices on Menus(price)
```

7.5. ix_tempdiscounts_dates

```
create index ix_tempdiscounts_dates on TemporaryDiscounts(validFrom,validTo)
```

7.6. ix_permdiscounts_dates

```
create index ix_permdiscounts_dates on TemporaryDiscounts(validFrom)
```

7.7. ix_discountInformations_dates

```
create index ix_discountInformations_dates on DiscountInformations(dateStart,dateStop)
```

7.8. ix_reservations_times

```
create index ix_reservations_times on Reservations(startTime,endTime)
```

7.9. ix_dishes_names

```
create index ix_dishes_names on Dishes(dishName)
```

7.10. ix_categories_names

```
create index ix_categories_names on Categories(categoryName)
```

7.11. ix_tables_nrOfChairs

```
create index ix_tables_nrOfChairs on Tables(numberofchairs)
```

7.12. ix_orders_customerId

```
create index ix_Orders_customerId on Orders(customerId)
```

8. Triggery

8.1. TR_ValidDiscountInformationsValues

Trigger sprawdza czy dodawane wartości dla zniżek są poprawne

```
CREATE TRIGGER [dbo].[TR_ValidDiscountInformationsValues] ON [dbo].[DiscountInformations]
FOR INSERT AS BEGIN
    IF (select Z1 from inserted) < 0 OR
       (select K1 from inserted) < 0 OR
       (select D1 from inserted) < 0 OR
       (select R1 from inserted) < 0 OR
       (select R2 from inserted) < 0 OR
       (select K2 from inserted) < 0
    BEGIN
        RAISERROR ('Every Discount parameter should be greater than 0', 17, -1)
    END
END
GO

ALTER TABLE [dbo].[DiscountInformations] ENABLE TRIGGER [TR_ValidDiscountInformationsValues]
GO
```

8.2. TR_ValidCustomerForDiscount

Trigger sprawdza czy zniżka przyznawana jest dla klienta indywidualnego

```
CREATE TRIGGER [dbo].[TR_ValidCustomerForDiscount] on [dbo].[TemporaryDiscounts]
AFTER INSERT, UPDATE AS BEGIN
    SET NOCOUNT ON;
    IF (select I.CustomerID from inserted I
        INNER JOIN IndCustomers as IC on IC.CustomerID = I.CustomerID
        WHERE IC.CustomerID = I.CustomerID) IS NULL
    BEGIN
        ROLLBACK;
        RAISERROR('Discounts are only for individual customers', 17, -1);
    END
END
GO
```

8.3. TR_ValidMenuInsert

Trigger sprawdza czy dodane danie nie znajduje się w aktualnym menu

```
CREATE TRIGGER [dbo].[TR_ValidMenuInsert] ON [dbo].[Menus]
AFTER INSERT, UPDATE AS BEGIN
    IF (select I.PosID from inserted as I
        INNER JOIN Menus as M on M.DishID = I.DishID AND M.PosID != I.PosID
        WHERE M.ValidTo IS NULL AND I.ValidTo IS NULL) IS NOT NULL
    BEGIN
        ROLLBACK;
        RAISERROR ('This dish already exist in current menu', 16, -1)
    END
END
GO
```

8.4. TR_ChangeMenuDatesValidity

Trigger sprawdza czy danie które chcemy usunąć z aktualnego menu nie jest w trakcie niezrealizowanego zamówienia

```
CREATE TRIGGER [dbo].[TR_ChangeMenuDatesValidity] ON [dbo].[Menus]
AFTER UPDATE AS BEGIN
    SET NOCOUNT ON;

    IF (select I.DishID from inserted as I
        INNER JOIN [Order Details] as OD on OD.PosID = I.PosID
        INNER JOIN Orders as O on O.OrderID = OD.OrderID
        WHERE O.OrderDate > GETDATE()) IS NOT NULL
    BEGIN
        ROLLBACK;
        RAISERROR ('You cannot remove a dish from the menu because there is an unfulfilled order on it', 17, -1)
    END
END
GO

ALTER TABLE [dbo].[Menus] ENABLE TRIGGER [TR_ChangeMenuDatesValidity]
GO
```

8.5. TR_UpdateUnitsInStock

Trigger poprawia ilość dań w magazynie po usunięciu zamówienia

```
CREATE TRIGGER [dbo].[TR_UpdateUnitsInStock] ON [dbo].[Order Details]
AFTER DELETE AS BEGIN
    DECLARE @PrevQuantity INT
    DECLARE @QuantityToAdd INT
    DECLARE @DishID INT
    DECLARE @NewUnitsInStock INT

    select @PrevQuantity = D.UnitsInStock, @QuantityToAdd = DEL.Quantity, @DishID = D.DishID from deleted as DEL
    INNER JOIN Menus as M on M.PosID = DEL.PosID
    INNER JOIN Dishes as D on D.DishID = M.DishID

    SET @NewUnitsInStock = @PrevQuantity + @QuantityToAdd

    UPDATE Dishes
    SET UnitsInStock = @NewUnitsInStock WHERE @DishID = DishID
END
GO

ALTER TABLE [dbo].[Order Details] ENABLE TRIGGER [TR_UpdateUnitsInStock]
GO
```

8.6. TR_CheckDeleteTable

Trigger sprawdza czy usuwany stolik nie jest przypisany do rezerwacji która jeszcze się nie odbyła

```
CREATE TRIGGER [dbo].[TR_CheckDeleteTable] ON [dbo].[Tables]
AFTER DELETE AS BEGIN
    IF (select I.TableID from inserted as I
        INNER JOIN Reservations as R on R.TableID = I.TableID
        WHERE R.StartTime > GETDATE()) IS NOT NULL
    BEGIN
        ROLLBACK;
        RAISERROR ('You cannot remove table, because there is unfulfilled reservation on it', 17, -1)
    END
END
GO
```

8.7. TR_ValidWZWKValues

Trigger sprawdza poprawność parametrów WZ i WK

```
CREATE TRIGGER [dbo].[TR_ValidWZWKValues] ON [dbo].[WZandWKs]
AFTER UPDATE AS BEGIN
    IF (select WZ from inserted) <= 0 OR
       (select WK from inserted) <= 0
    BEGIN
        ROLLBACK;
        RAISERROR ('WZ and WK should be greater than 0', 17, -1)
    END
END
GO
```

9. Uprawnienia

9.1 Rola StoreMan

```
create Role Storeman
grant select on DishesInStock to StoreMan
```

9.2 Rola IndCustomer

```
create Role IndCustomer
grant select on CurrentMenu to IndCustomer
grant execute on uspAddOrder to IndCustomer
grant execute on uspAddIndReservation to IndCustomer
grant select on FreeTables to IndCustomer
grant select on GetTemporaryDiscount to IndCustomer
grant select on GetPermanentDiscount to IndCustomer
grant select on GetInformationAboutIndCustomer to IndCustomer
grant execute on GetOrderPriceWithDiscount to IndCustomer
```

9.3 Rola CompanyCustomer

```
create Role CompanyCustomer
grant select on CurrentMenu to CompanyCustomer
grant execute on uspAddOrder to CompanyCustomer
grant execute on uspAddCompanyReservation to CompanyCustomer
grant select on FreeTables to CompanyCustomer
grant select on GetInformationAboutCompany to CompanyCustomer
```

9.4 Rola Manager

```
create Role Manager
grant select on DiscountInfoHistory to Manager
grant select on DishesRank to Manager
grant select on NumberOfOrders to Manager
grant select on WZWK to Manager
grant select on CategoriesStatistics to Manager
grant select on CategoriesMonthlyStatistics to Manager
grant select on CategoriesYearlyStatistics to Manager
grant select on CategoriesTotalStatistics to Manager
grant select on TablesStatistics to Manager
grant select on TablesMonthlyStatistics to Manager
grant select on TablesYearlyStatistics to Manager
grant select on TablesTotalStatistics to Manager
grant select on CustomerMonthlyStatistics to Manager
grant select on CustomerYearltStatistics to Manager
grant select on DishesStatistics to Manager
grant select on DishesMonthlyStatistics to Manager
grant select on DishesYearlyStatistics to Manager
grant select on DishesTotalStatistics to Manager
grant select on ReservationsStatistics to Manager
grant select on ReservationsMonthlyStatistics to Manager
grant select on ReservationsYearlyStatistics to Manager
grant select on OrdersStatistics to Manager
grant select on OrdersMonthlyStatistics to Manager
grant select on OrdersYearlyStatistics to Manager
grant select on isMenuValidate to Manager
grant execute on uspAddPosToMenu to Manager
grant execute on uspRemoveDishFromCurrentMenu to Manager
grant select on GetIncome to Manager
grant select on GetTableNumberOfReservations to Manager
grant select on GetNumberOfDishOrders to Manager
grant select on GenerateInvoice to Manager
```

9.5 Rola Waiter

```
create Role Waiter
grant select on ClientActiveDiscounts to Waiter
grant select on ClientDiscounts to Waiter
grant select on CurrentMenu to Waiter
grant select on CurrentDiscountInfo to Waiter
grant select on TodaysOrders to Waiter
grant select on TodaysReservations to Waiter
grant select on UnconfirmedReservations to Waiter
grant select on CompanyGuestList to Waiter
grant select on TakeAwayOrders to Waiter
grant select on ShowIndCustomersInformations to Waiter
grant select on ShowCompaniesInformations to Waiter
grant select on CustomerStatistics to Waiter
grant execute on uspAddOrder to Waiter
grant execute on uspUpdateReservation to Waiter
grant execute on uspGrantPermanentDiscount to Waiter
grant execute on uspGrantTemporaryDiscount to Waiter
grant execute on uspAddIndCustomer to Waiter
grant execute on uspAddCompanyCustomer to Waiter
grant select on FreeTables to Waiter
grant select on TakenTables to Waiter
grant select on GetOrderDetails to Waiter
grant select on GetCustomerOrders to Waiter
grant select on GetTemporaryDiscount to Waiter
grant select on Permanent to Waiter
grant select on GetOrderPriceWithDiscount to Waiter
grant select on GetDishCategory to Waiter
```

9.6 Rola RestaurantOwner

```
create Role RestaurantOwner
grant execute on uspAddDish to RestaurantOwner
grant execute on uspAddTables to RestaurantOwner
grant execute on uspChangeNumberOfChairs to RestaurantOwner
grant execute on uspDeleteTable to RestaurantOwner
grant execute on uspUpdateWZandWK to RestaurantOwner
grant execute on uspUpdateDiscountParameters to RestaurantOwner
grant execute on uspAddNewCategory to RestaurantOwner
```