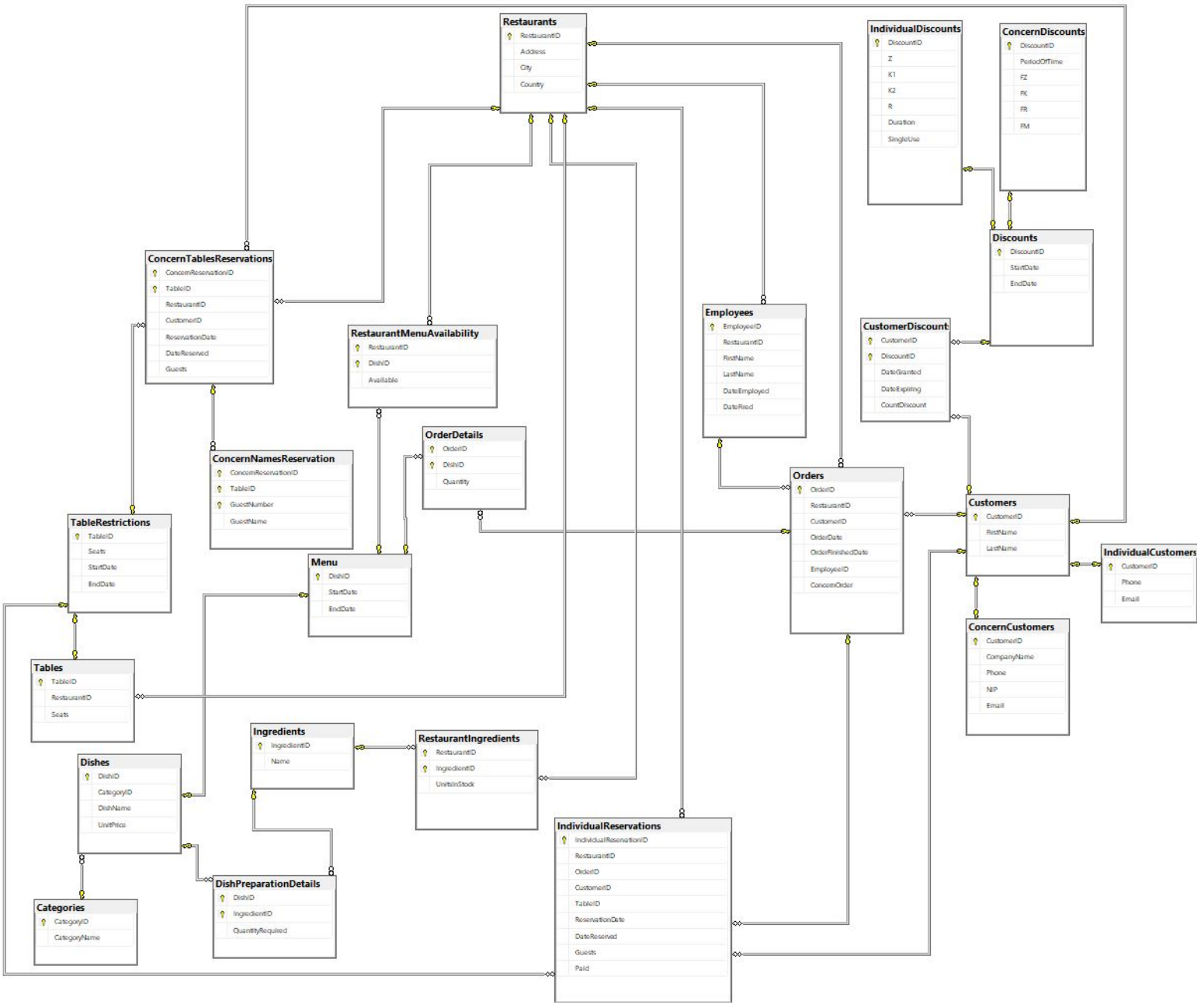
Dokumentacja

24.01.2021 r.
Paweł Dymara
Norbert Wolniak
Grupa Poniedziałek 14.40

1. Schemat bazy danych :



2. Opisy tabel

1. Orders

Przechowuje informacje o zamówieniach, klucz główny to OrderID.
Zawiera klucze:

-OrderID -identyfikator zamówienia [typ int]
-RestaurantID - identyfikator restauracji której to zamówienie dotyczy [typ int]
-CustomerID - identyfikator klienta który złożył zamówienie [typ int]
-OrderDate - data złożenia zamówienia [typ datetime]
-OrderFinishedDate - data zakończenia zamówienia (dopóki zamówienie nie zostanie zrealizowane to pole może być nullem) [typ datetime]
-EmployeeID - identyfikator pracownika, który przyjął zamówienie [typ int]
-ConcernOrder - wartość logiczna: prawda jeżeli zamówienie ma być traktowane jako złożone przez firmę, fałsz jeżeli ma być traktowane jak złożone przez klienta indywidualnego (jest to konieczne, ponieważ klient może reprezentować firmę, ale może być też prywatnym klientem a posiada jedno CustomerID) [typ bit]

Warunki integralności:

-OrderFinishedDate jest późniejszą datą niż OrderDate lub jest nullem.

```
--START Orders
CREATE TABLE [dbo].[Orders](
	[OrderID] [int] NOT NULL,
	[RestaurantID] [int] NOT NULL,
	[CustomerID] [int] NOT NULL,
	[OrderDate] [datetime] NOT NULL,
	[OrderFinishedDate] [datetime] NULL,
	[EmployeeID] [int] NOT NULL,
	[ConcernOrder] [bit] NOT NULL,
 CONSTRAINT [PK_Orders] PRIMARY KEY CLUSTERED
(
	[OrderID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

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

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

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

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

ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT
[FK_Orders_Restaurants] FOREIGN KEY([RestaurantID])
REFERENCES [dbo].[Restaurants] ([RestaurantID])
GO

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

ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT
[CK_Orders_OrderDate] CHECK  (([OrderDate]<=getdate()))
GO

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

ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT
[CK_Orders_OrderFinishedDate] CHECK  (([OrderFinishedDate]>=[OrderDate]
AND [OrderFinishedDate]<=getdate() OR [OrderFinishedDate] IS NULL))
GO

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


--END Orders
```

2. OrderDetails

Przechowuje informacje o zawartości zamówienia, klucz główny to para OrderID-DishID.
Zawiera klucze:

-OrderID - identyfikator zamówienia [typ int]
-DishID - identyfikator dania wchodzącego w skład tego zamówienia [typ int]
-Quantity - ilość porcji tego dania wchodzących w skład zamówienia [typ int]

Warunki integralności:

-Quantity jest liczbą większą od zera

```
--START OrderDetails
CREATE TABLE [dbo].[OrderDetails](
      [OrderID] [int] NOT NULL,
      [DishID] [int] NOT NULL,
      [Quantity] [int] NOT NULL,
 CONSTRAINT [PK_OrderDetails] PRIMARY KEY CLUSTERED
(
      [OrderID] ASC,
      [DishID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

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

ALTER TABLE [dbo].[OrderDetails] CHECK CONSTRAINT [FK_Order
Details_Orders]
GO

ALTER TABLE [dbo].[OrderDetails]  WITH CHECK ADD  CONSTRAINT
[FK_OrderDetails_Menu] FOREIGN KEY([DishID])
REFERENCES [dbo].[Menu] ([DishID])
GO

ALTER TABLE [dbo].[OrderDetails] CHECK CONSTRAINT [FK_OrderDetails_Menu]
GO

ALTER TABLE [dbo].[OrderDetails]  WITH CHECK ADD  CONSTRAINT
[CK_OrderDetails_Quantity] CHECK  (([Quantity]>(0)))
GO

ALTER TABLE [dbo].[OrderDetails] CHECK CONSTRAINT
[CK_OrderDetails_Quantity]
GO
--END OrderDetails
```

3. Customers

Tabela zawierająca podstawowe informacje o wszystkich klientach, klucz główny to CustomerID.
Zawiera klucze:

-CustomerID - identyfikator klienta [typ int]
-FirstName - imię klienta [typ nvarchar(50)]
-LastName - nazwisko klienta [typ nvarchar(50)]

```
--START Customers
CREATE TABLE [dbo].[Customers](
	[CustomerID] [int] NOT NULL,
	[FirstName] [nvarchar](50) NOT NULL,
	[LastName] [nvarchar](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]
) ON [PRIMARY]
GO
--END Customers
```

4. IndividualCustomers

Tabela zawierająca szczegółowe informacje o klientach indywidualnych, klucz główny to CustomerID.
Zawiera klucze:

-CustomerID - identyfikator klienta [typ int]
-Phone - numer telefonu klienta (prywatny) [typ varchar(20)]
-Email - adres email klienta (prywatny) [typ nvarchar[50]]

Warunki integralności:

-Email zawiera znak ‘@’
-Email jest unikalny
-Phone zawiera tylko cyfry (i ewentualnie znak ‘+’ na pierwszym miejscu)

```
--START IndividualCustomers
CREATE TABLE [dbo].[IndividualCustomers](
	[CustomerID] [int] NOT NULL,
	[Phone] [varchar](20) NOT NULL,
	[Email] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_IndividualCustomers] 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 [UQ_IndividualCustomers] UNIQUE NONCLUSTERED
(
      [Email] 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].[IndividualCustomers]  WITH CHECK ADD  CONSTRAINT
[FK_IndividualCustomers_Customers] FOREIGN KEY([CustomerID])
REFERENCES [dbo].[Customers] ([CustomerID])
GO

ALTER TABLE [dbo].[IndividualCustomers] CHECK CONSTRAINT
[FK_IndividualCustomers_Customers]
GO

ALTER TABLE [dbo].[IndividualCustomers]  WITH CHECK ADD  CONSTRAINT
[CK_IndividualCustomer_Email] CHECK  (([Email] like '%@%'))
GO

ALTER TABLE [dbo].[IndividualCustomers] CHECK CONSTRAINT
[CK_IndividualCustomer_Email]
GO

ALTER TABLE [dbo].[IndividualCustomers]  WITH CHECK ADD  CONSTRAINT
[CK_IndividualCustomer_Phone] CHECK  ((NOT [Phone] like '%[^0-9]%' OR
NOT [Phone] like '^+%[^0-9]%'))
GO

ALTER TABLE [dbo].[IndividualCustomers] CHECK CONSTRAINT
[CK_IndividualCustomer_Phone]
GO

--END IndividualCustomers
```

5. ConcernCustomers

Tabela zawierająca szczegółowe informacje o klientach firmowych, klucz główny to CustomerID.

Zawiera klucze:

-CustomerID - identyfikator klienta [typ int]

-CompanyName - nazwa firmy, którą reprezentuje [nvarchar(50)]
-Phone - numer telefonu klienta (służbowy) [typ varchar(20)]
-NIP - NIP firmy [typ char(10)]
-Email - adres email klienta (służbowy) [typ nvarchar(50)]

Warunki integralności:

-NIP jest unikalny, zawiera tylko liczby.
-Email zawiera znak '@'.
-Email jest unikalny
-Phone zawiera tylko cyfry (i ewentualnie znak '+' na pierwszym miejscu).

```
--START ConcernCustomers
CREATE TABLE [dbo].[ConcernCustomers](
	[CustomerID] [int] NOT NULL,
	[CompanyName] [nvarchar](50) NOT NULL,
	[Phone] [varchar](20) NOT NULL,
	[NIP] [char](10) NOT NULL,
	[Email] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_ConcernCustomers] 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 [UQ_ConcernCustomers_Email] UNIQUE NONCLUSTERED
(
	[Email] 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 [UQ_ConcernCustomers_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].[ConcernCustomers]  WITH CHECK ADD  CONSTRAINT
[FK_ConcernCustomers_Customers] FOREIGN KEY([CustomerID])
REFERENCES [dbo].[Customers] ([CustomerID])
GO

ALTER TABLE [dbo].[ConcernCustomers] CHECK CONSTRAINT
[FK_ConcernCustomers_Customers]
GO
```

```
ALTER TABLE [dbo].[ConcernCustomers]  WITH CHECK ADD  CONSTRAINT
[CK_ConcernCustomers_Email] CHECK  (([Email] like '%@%'))
GO

ALTER TABLE [dbo].[ConcernCustomers] CHECK CONSTRAINT
[CK_ConcernCustomers_Email]
GO

ALTER TABLE [dbo].[ConcernCustomers]  WITH CHECK ADD  CONSTRAINT
[CK_ConcernCustomers_NIP] CHECK  ((NOT [NIP] like '%[^0-9]%'))
GO

ALTER TABLE [dbo].[ConcernCustomers] CHECK CONSTRAINT
[CK_ConcernCustomers_NIP]
GO

--END ConcernCustomers
```

6. Menu

Tabela zawierająca informacje o daniach wchodzących w skład danego menu (menu jest globalne dla wszystkich restauracji), klucz główny to DishID.

Zawiera klucze:

-DishID - identyfikator dania [typ int]

-StartDate - data początku obowiązywania danego dania w menu [typ datetime]

-EndDate - data końca obowiązywania danego dania w menu [typ datetime]

Warunki integralności:

-EndDate jest datą późniejszą niż StartDate.

```
--START Menu
CREATE TABLE [dbo].[Menu](
      [DishID] [int] NOT NULL,
      [StartDate] [datetime] NOT NULL,
      [EndDate] [datetime] NOT NULL,
 CONSTRAINT [PK_Menu_1] PRIMARY KEY CLUSTERED
(
      [DishID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

```
ALTER TABLE [dbo].[Menu]  WITH CHECK ADD  CONSTRAINT [FK_Menu_Dishes]
FOREIGN KEY([DishID])
REFERENCES [dbo].[Dishes] ([DishID])
GO

ALTER TABLE [dbo].[Menu] CHECK CONSTRAINT [FK_Menu_Dishes]
GO

ALTER TABLE [dbo].[Menu]  WITH CHECK ADD  CONSTRAINT [CK_Menu] CHECK
(([EndDate]>[StartDate]))
GO

ALTER TABLE [dbo].[Menu] CHECK CONSTRAINT [CK_Menu]
GO
--END Menu
```

7. RestaurantMenuAvailability

Tabela zawierająca informację o dostępności poszczególnych dań w menu dla każdej z restauracji. Klucz główny to para RestaurantID-DishID.
Zawiera klucze:

-RestaurantID - identyfikator restauracji [typ int]
-DishID - identyfikator dania [typ int]
-Available - wartość logiczna: prawda jeżeli danie z tego menu jest dostępne w restauracji, fałsz gdy w tej restauracji skończyły się składniki i danie zostało zdjęte [typ bit]

```
--START RestaurantMenuAvailability
CREATE TABLE [dbo].[RestaurantMenuAvailability](
      [RestaurantID] [int] NOT NULL,
      [DishID] [int] NOT NULL,
      [Available] [bit] NOT NULL,
 CONSTRAINT [PK_RestaurantMenuAvailability_1] PRIMARY KEY CLUSTERED
(
      [RestaurantID] ASC,
      [DishID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[RestaurantMenuAvailability]  WITH CHECK ADD
CONSTRAINT [FK_RestaurantMenuAvailability_Menu] FOREIGN KEY([DishID])
```

```
REFERENCES [dbo].[Menu] ([DishID])
GO

ALTER TABLE [dbo].[RestaurantMenuAvailability] CHECK CONSTRAINT
[FK_RestaurantMenuAvailability_Menu]
GO

ALTER TABLE [dbo].[RestaurantMenuAvailability]  WITH CHECK ADD
CONSTRAINT [FK_RestaurantMenuAvailability_Restaurants] FOREIGN
KEY([RestaurantID])
REFERENCES [dbo].[Restaurants] ([RestaurantID])
GO

ALTER TABLE [dbo].[RestaurantMenuAvailability] CHECK CONSTRAINT
[FK_RestaurantMenuAvailability_Restaurants]
GO
--END RestaurantMenuAvailability
```

8. Restaurants

Tabela zawierająca informację o restauracjach obsługiwanych przez bazę, klucz główny to RestaurantID.
Zawiera klucze:

-RestaurantID - identyfikator restauracji [typ int]
-Address - adres restauracji [nvarchar (50)]
-City - miasto [nvarchar (50)]
-Country - kraj [nvarchar (50)]

```
--START Restaurants
CREATE TABLE [dbo].[Restaurants](
      [RestaurantID] [int] NOT NULL,
      [Address] [nvarchar](50) NOT NULL,
      [City] [nvarchar](50) NOT NULL,
      [Country] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_Restaurants] PRIMARY KEY CLUSTERED
(
      [RestaurantID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
--END Restaurants
```

9. Dishes

Tabela zawierająca podstawowe informacje o oferowanych daniach. klucz główny to DishID.
Zawiera klucze:

-DishID - identyfikator dania [typ int]
-CategoryID - identyfikator kategorii, do której należy danie [typ int]
-DishName - nazwa dania [typ nvarchar(50)]
-UnitPrice - cena za jedną porcję dania [typ money]

Warunki integralności:

-UnitPrice jest liczbą większą od zera.

```
--START Dishes
CREATE TABLE [dbo].[Dishes](
	[DishID] [int] NOT NULL,
	[CategoryID] [int] NOT NULL,
	[DishName] [nvarchar](50) NOT NULL,
	[UnitPrice] [money] NOT NULL,
 CONSTRAINT [PK_Products_1] PRIMARY KEY CLUSTERED
(
	[DishID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

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

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

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

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

10. DishPreparationDetails

Tabela zawierająca informacje o składzie dań, klucz główny to para DishID-IngredientID.
Zawiera klucze:

-DishID - identyfikator dania [typ int]
-IngredientID - identyfikator składnika [typ int]
-QuantityRequired - wymagana ilość tego składnika [typ real]

Warunki integralności:

-QuantityRequired jest liczbą większą od zera.

```
--START DishPreparationDetails
CREATE TABLE [dbo].[DishPreparationDetails](
	[DishID] [int] NOT NULL,
	[IngredientID] [int] NOT NULL,
	[QuantityRequired] [real] NOT NULL,
 CONSTRAINT [PK_Products] PRIMARY KEY CLUSTERED
(
	[DishID] ASC,
	[IngredientID] 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].[DishPreparationDetails]  WITH CHECK ADD  CONSTRAINT
[FK_DishPreparationDetails_Dishes] FOREIGN KEY([DishID])
REFERENCES [dbo].[Dishes] ([DishID])
GO

ALTER TABLE [dbo].[DishPreparationDetails] CHECK CONSTRAINT
[FK_DishPreparationDetails_Dishes]
GO

ALTER TABLE [dbo].[DishPreparationDetails]  WITH CHECK ADD  CONSTRAINT
[FK_DishPreparationDetails_Ingredients] FOREIGN KEY([IngredientID])
REFERENCES [dbo].[Ingredients] ([IngredientID])
GO

ALTER TABLE [dbo].[DishPreparationDetails] CHECK CONSTRAINT
[FK_DishPreparationDetails_Ingredients]
GO

ALTER TABLE [dbo].[DishPreparationDetails]  WITH CHECK ADD  CONSTRAINT
[CK_DishPreparationDetails] CHECK  (([QuantityRequired]>(0)))
GO

ALTER TABLE [dbo].[DishPreparationDetails] CHECK CONSTRAINT
[CK_DishPreparationDetails]
GO
```

```
--END DishPreparationDetails
```

11. Ingredients

Tabela zawierająca informacje o składnikach, klucz główny to IngredientID.
Zawiera klucze:

-IngredientID - identyfikator składnika [typ int]
-Name - nazwa składnika (np. mąka pszenna 1kg) [typ nvarchar(50)]

```
--START Ingredients
CREATE TABLE [dbo].[Ingredients](
	[IngredientID] [int] NOT NULL,
	[Name] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_Ingredients] PRIMARY KEY CLUSTERED
(
	[IngredientID] 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
--END Ingredients
```

12. RestaurantIngredients

Tabela zawierająca informację o obecnym stanie składników dla danej restauracji, klucz główny to para RestaurantID-IngredientID. Zawiera klucze:

-RestaurantID - identyfikator restauracji [typ int]
-IngredientID - identyfikator składnika [typ int]
-UnitsInStock - ilość sztuk składnika dostępnych w magazynie [typ real]

Warunki integralności:

-UnitsInStock i UnitsInOrder są liczbami większymi bądź równymi zero.

```
--START RestaurantIngredients
CREATE TABLE [dbo].[RestaurantIngredients](
	[RestaurantID] [int] NOT NULL,
	[IngredientID] [int] NOT NULL,
	[UnitsInStock] [real] NOT NULL,
 CONSTRAINT [PK_RestaurantIngredients] PRIMARY KEY CLUSTERED
(
	[RestaurantID] ASC,
	[IngredientID] 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].[RestaurantIngredients]  WITH CHECK ADD  CONSTRAINT
[FK_RestaurantIngredients_Ingredients] FOREIGN KEY([IngredientID])
REFERENCES [dbo].[Ingredients] ([IngredientID])
GO

ALTER TABLE [dbo].[RestaurantIngredients] CHECK CONSTRAINT
[FK_RestaurantIngredients_Ingredients]
GO

ALTER TABLE [dbo].[RestaurantIngredients]  WITH CHECK ADD  CONSTRAINT
[FK_RestaurantIngredients_Restaurants] FOREIGN KEY([RestaurantID])
REFERENCES [dbo].[Restaurants] ([RestaurantID])
GO

ALTER TABLE [dbo].[RestaurantIngredients] CHECK CONSTRAINT
[FK_RestaurantIngredients_Restaurants]
GO

ALTER TABLE [dbo].[RestaurantIngredients]  WITH CHECK ADD  CONSTRAINT
[CK_RestaurantIngredients] CHECK  (([UnitsInStock]>=(0) AND
[UnitsInOrder]>=(0)))
GO

ALTER TABLE [dbo].[RestaurantIngredients] CHECK CONSTRAINT
[CK_RestaurantIngredients]
GO
--END RestaurantIngredients
```

13. Categories

Tabela przechowująca informacje o kategoriach produktów. Klucz główny to CategoryID.
Zawiera klucze:

-CategoryID - identyfikator kategorii [typ int]
-CategoryName - nazwa kategorii (np. zupa, deser)

```
--START Categories
CREATE TABLE [dbo].[Categories](
      [CategoryID] [int] NOT NULL,
      [CategoryName] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_Categories] PRIMARY KEY CLUSTERED
(
      [CategoryID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
```

```
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
--END Categories
```

14. IndividualDiscounts

Tabela przechowująca informacje o wszystkich oferowanych rodzajach zniżek dla klientów indywidualnych i kiedy można było je przyznać, klucz główny to DiscountID.
Zawiera klucze:

-DiscountID - identyfikator zniżki [typ int]
-Z - minimalna ilość złożonych zamówień [typ int]
-K1 - minimalny koszt na który musi zostać złożone Z zamówień by można było przyznać zniżkę [typ money]
-K2 - minimalny łączny koszt zamówienia dla którego można przyznać zniżkę [typ money]
-R - liczba która określa wielkość rabatu w procentach [typ real]
-StartDate - data od kiedy można było przyznać zniżkę [typ datetime]
-EndDate - data do kiedy można było przyznać zniżkę [typ datetime]
-Duration - ilość dni przez które zniżka jest ważna [typ int]
-SingleUse - czy zniżka jednorazowa [typ bit]

Warunki integralności:

-Z, K1, K2, Duration są liczbami większymi od zera.
-R jest większe od zera mniejsze od 1
-EndDate jest datą późniejszą niż StartDate lub nullem.

```
--START IndividualDiscounts
CREATE TABLE [dbo].[IndividualDiscounts](
      [DiscountID] [int] NOT NULL,
      [Z] [int]  NULL,
      [K1] [int]  NULL,
      [K2] [int]  NULL,
      [R] [real] NOT NULL,
      [StartDate] [datetime] NOT NULL,
      [EndDate] [datetime] NULL,
      [Duration] [int] NOT NULL,
      [SingleUse] [bit] NOT NULL,
 CONSTRAINT [PK_IndividualDiscounts] 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].[IndividualDiscounts]  WITH CHECK ADD  CONSTRAINT
[CK_IndividualDiscounts] CHECK  (([Z]>(0) AND [K1]>(0) AND [K2]>(0)))
GO

ALTER TABLE [dbo].[IndividualDiscounts] CHECK CONSTRAINT
[CK_IndividualDiscounts]
GO

ALTER TABLE [dbo].[IndividualDiscounts]  WITH CHECK ADD  CONSTRAINT
[CK_IndividualDiscounts_1] CHECK  (([R]>(0) AND [R]<(1)))
GO

ALTER TABLE [dbo].[IndividualDiscounts] CHECK CONSTRAINT
[CK_IndividualDiscounts_1]
GO

ALTER TABLE [dbo].[IndividualDiscounts]  WITH CHECK ADD  CONSTRAINT
[CK_IndividualDiscounts_2] CHECK  (([EndDate]>[StartDate] OR [EndDate]
IS NULL))
GO

ALTER TABLE [dbo].[IndividualDiscounts] CHECK CONSTRAINT
[CK_IndividualDiscounts_2]
GO

--END IndividualDiscounts
```

15. ConcernDiscounts

Tabela przechowująca informacje o wszystkich oferowanych rodzajach zniżek dla klientów firmowych i kiedy można było je przyznać, klucz główny to DiscountID.
Zawiera klucze:

-DiscountID - identyfikator zniżki [typ int]
-PeriodOfTime - przedział czasu co jaki naliczana jest zniżka [typ char[1]]
-FZ - minimalna liczba zamówień w miesiącu [typ int]
-FK - minimalna łączna kwota zamówień w podanym przedziale czasu [typ money]
-FR - liczba która określa wielkość rabatu w procentach [typ real]
-FM - maksymalny procent rabatu [typ real]
-StartDate - data od kiedy można było przyznać zniżkę [typ datetime]
-EndDate - data do kiedy można było przyznać zniżkę [typ datetime]

Warunki integralności:

-FZ i FK są liczbami większymi od zera.
-FR i FM są liczbami większymi od zera i mniejszymi od 1.
-FM jest liczbą większą bądź równą FR.

-EndDate jest datą późniejszą niż StartDate lub nullem.

```
--START ConcernDiscounts
CREATE TABLE [dbo].[ConcernDiscounts](
      [DiscountID] [int] NOT NULL,
      [PeriodOfTime] [char](1) NOT NULL,
      [FZ] [int] NULL,
      [FK] [money] NOT NULL,
      [FR] [real] NOT NULL,
      [FM] [real] NOT NULL,
      [StartDate] [datetime] NOT NULL,
      [EndDate] [datetime] NULL,
 CONSTRAINT [PK_ConcernDiscounts] 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].[ConcernDiscounts]  WITH CHECK ADD  CONSTRAINT
[CK_ConcernDiscounts] CHECK  (([FZ]>(0) AND [FK]>(0)))
GO

ALTER TABLE [dbo].[ConcernDiscounts] CHECK CONSTRAINT
[CK_ConcernDiscounts]
GO

ALTER TABLE [dbo].[ConcernDiscounts]  WITH CHECK ADD  CONSTRAINT
[CK_ConcernDiscounts_1] CHECK  (([FR]>(0) AND [FM]>(0) AND [FR]<(1) AND
[FM]<(1)))
GO

ALTER TABLE [dbo].[ConcernDiscounts] CHECK CONSTRAINT
[CK_ConcernDiscounts_1]
GO

ALTER TABLE [dbo].[ConcernDiscounts]  WITH CHECK ADD  CONSTRAINT
[CK_ConcernDiscounts_2] CHECK  (([FM]>=[FR]))
GO

ALTER TABLE [dbo].[ConcernDiscounts] CHECK CONSTRAINT
[CK_ConcernDiscounts_2]
GO
```

```
ALTER TABLE [dbo].[ConcernDiscounts]  WITH CHECK ADD  CONSTRAINT
[CK_ConcernDiscounts_3] CHECK  (([EndDate]>[StartDate] OR [EndDate] IS
NULL))
GO

ALTER TABLE [dbo].[ConcernDiscounts] CHECK CONSTRAINT
[CK_ConcernDiscounts_3]
GO

--END ConcernDiscounts
```

16. CustomerDiscounts

Zawiera informację o przyznanych zniżkach

Zawiera klucze:

-CustomerID - identyfikator klienta [typ int]

-DiscountID - identyfikator rodzaju zniżki [typ int]

-DateGranted - data przyznania zniżki [typ datetime]

-DateExpiring - data wygaśnięcia zniżki [typ datetime]

-CountDiscount - liczba mówiąca o tym ile razy dany typ zniżki jest liczony (na potrzeby zniżek kumulowanych) [typ int]

Warunki integralności:

-CountDiscount jest liczbą większą lub równą od zera.

-DateExpiring jest datą późniejszą niż DateGranted.

```
CREATE TABLE [dbo].[CustomerDiscounts](
      [CustomerID] [int] NOT NULL,
      [DiscountID] [int] NOT NULL,
      [DateGranted] [datetime] NOT NULL,
      [DateExpiring] [datetime] NULL,
      [CountDiscount] [int] NOT NULL,
 CONSTRAINT [PK_Discounts] PRIMARY KEY CLUSTERED
(
      [CustomerID] ASC,
      [DiscountID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[CustomerDiscounts]  WITH NOCHECK ADD  CONSTRAINT
[FK_CustomerDiscounts_Discounts] FOREIGN KEY([DiscountID])
REFERENCES [dbo].[Discounts] ([DiscountID])
GO
```

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

ALTER TABLE [dbo].[CustomerDiscounts]  WITH NOCHECK ADD  CONSTRAINT
[FK_Discounts_Customers] FOREIGN KEY([CustomerID])
REFERENCES [dbo].[Customers] ([CustomerID])
GO

ALTER TABLE [dbo].[CustomerDiscounts] CHECK CONSTRAINT
[FK_Discounts_Customers]
GO

ALTER TABLE [dbo].[CustomerDiscounts]  WITH NOCHECK ADD  CONSTRAINT
[CK_CustomerDiscounts_CountDiscount] CHECK  (([CountDiscount]>=(0)))
GO

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

17. Discounts

Zawiera informację o czasie obowiązywania zniżek.
Zawiera klucze:

-DiscountID - identyfikator rodzaju zniżki [typ int]
-StartDate - data obowiązywania zniżki [typ datetime]
-EndDate - data wygaśnięcia zniżki [typ datetime]

Warunki integralności:

-EndDate > StartDate

```
--START Discounts
CREATE TABLE [dbo].[Discounts](
      [DiscountID] [int] IDENTITY(1,1) NOT NULL,
      [StartDate] [datetime] NOT NULL,
      [EndDate] [datetime] NULL,
 CONSTRAINT [PK_Discounts_2] PRIMARY KEY CLUSTERED
(
      [DiscountID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
```

```
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Discounts]  WITH NOCHECK ADD  CONSTRAINT
[CK_Discounts] CHECK  (([StartDate]<[EndDate]))
GO

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

EXEC sys.sp_addextendedproperty @name=N'MS_Description', @value=N'' ,
@level0type=N'SCHEMA',@level0name=N'dbo',
@level1type=N'TABLE',@level1name=N'Discounts',
@level2type=N'CONSTRAINT',@level2name=N'CK_Discounts'
GO


--END Discounts
```

18. IndividualReservations

Tabela zawierająca zaakceptowane rezerwacje dla klientów indywidualnych, klucz główny to ReservationID.

Zawiera klucze:

-ReservationID - identyfikator rezerwacji [typ int]
-RestaurantID - identyfikator restauracji [typ int]
-OrderID - identyfikator zamówienia [typ int]
-CustomerID - identyfikator klienta [typ int]
-TableID - identyfikator stolika [typ int]
-ReservationDate - data złożenia rezerwacji [typ datetime]
-DateReserved - data zarezerwowana [typ datetime]
-Seats - ilość miejsc siedzących [typ int]
-Paid - czy opłacona z góry [typ bit]

Warunki integralności:

-DateReserved jest datą późniejszą niż ReservationDate.
-Seats jest liczbą większą od zera.

```
--START IndividualReservations
CREATE TABLE [dbo].[IndividualReservations](
      [ReservationID] [int] NOT NULL,
      [RestaurantID] [int] NOT NULL,
      [OrderID] [int] NOT NULL,
      [CustomerID] [int] NOT NULL,
      [TableID] [int] NOT NULL,
      [ReservationDate] [datetime] NOT NULL,
      [DateReserved] [datetime] NOT NULL,
```

```
      [Seats] [int] NOT NULL,
      [Paid] [bit] NOT NULL,
 CONSTRAINT [PK_dupa] 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].[IndividualReservations]  WITH CHECK ADD  CONSTRAINT
[FK_IndividualReservations_Customers] FOREIGN KEY([CustomerID])
REFERENCES [dbo].[Customers] ([CustomerID])
GO

ALTER TABLE [dbo].[IndividualReservations] CHECK CONSTRAINT
[FK_IndividualReservations_Customers]
GO

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

--END IndividualReservations
```

19. TableRestrictions

Tabela przechowująca aktualną ilość możliwych miejsc siedzących z powodu nałożenia restrykcji związanych z epidemią COVID-19, klucz główny to TableID.
Zawiera klucze:

-TableID - identyfikator stolika [typ int]
-Seats - ilość dostępnych miejsc [typ int]
-StartDate - data początkowa obowiązywania restrykcji [typ datetime]
-EndDate - data końcowa obowiązywania restrykcji [typ datetime]

Warunki integralności:

-EndDate jest datą późniejszą niż StartDate.
-Seats jest liczbą większą bądź równą zeru.

```
--START TableRestrictions
CREATE TABLE [dbo].[TableRestrictions](
      [TableID] [int] NOT NULL,
      [Seats] [tinyint] NOT NULL,
      [StartDate] [datetime] NOT NULL,
```

```
	[EndDate] [datetime] NOT NULL,
 CONSTRAINT [PK_TableRestrictions_1] 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].[TableRestrictions]  WITH CHECK ADD  CONSTRAINT
[CK_TableRestrictions] CHECK  (([EndDate]>[StartDate] AND [Seats]>=(0)))
GO

ALTER TABLE [dbo].[TableRestrictions] CHECK CONSTRAINT
[CK_TableRestrictions]
GO
--END TableRestrictions
```

20. Tables

Tabela przechowująca informacje o stolikach, klucz główny to TableID. Zawiera klucze:

-TableID - identyfikator stolika [typ int]

-RestaurantID - identyfikator restauracji [typ int]

-Seats - maksymalna ilość miejsc [typ int]

Warunki integralności:

-Seats jest liczbą większą od zera.

```
--START Tables
CREATE TABLE [dbo].[Tables](
	[TableID] [int] NOT NULL,
	[RestaurantID] [int] NOT NULL,
	[Seats] [int] NOT NULL,
 CONSTRAINT [PK_Tables_1] 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
[FK_Tables_TableRestrictions] FOREIGN KEY([TableID])
```

```
REFERENCES [dbo].[TableRestrictions] ([TableID])
GO

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

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

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

--END Tables
```

21. ConcernTablesReservations

Tabela zawierająca rezerwacje stolików wykonanych przez firmy. Zarówno na danego pracownika firmowego oraz imiennie. Klucz główny to para ReservationID-TableID (dla jednej rezerwacji można kilka stolików).
Zawiera klucze:

-ReservationID - identyfikator rezerwacji [typ int]
-TableID - identyfikator stolika [typ int]
-RestaurantID - identyfikator restauracji [typ int]
-CustomerID - identyfikator klienta [typ int]
-ReservationDate - data złożenia rezerwacji [typ datetime]
-DateReserved - data zarezerwowana [typ datetime]
-Guests - ilość osób [typ int]

Warunki integralności:

-DateReserved jest datą późniejszą niż ReservationDate.
-Guests jest liczbą większą od zera.

```
--START ConcernTableReservations
CREATE TABLE [dbo].[ConcernTablesReservations](
      [ReservationID] [int] NOT NULL,
      [TableID] [int] NOT NULL,
      [RestaurantID] [int] NOT NULL,
      [CustomerID] [int] NOT NULL,
      [ReservationDate] [datetime] NOT NULL,
      [DateReserved] [datetime] NOT NULL,
      [Guests] [int] NOT NULL,
 CONSTRAINT [PK_ConcernTablesReservations] PRIMARY KEY CLUSTERED
(
      [ReservationID] ASC,
      [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].[ConcernTablesReservations]  WITH CHECK ADD
CONSTRAINT [FK_ConcernTablesReservations_Customers] FOREIGN
KEY([CustomerID])
REFERENCES [dbo].[Customers] ([CustomerID])
GO

ALTER TABLE [dbo].[ConcernTablesReservations] CHECK CONSTRAINT
[FK_ConcernTablesReservations_Customers]
GO

ALTER TABLE [dbo].[ConcernTablesReservations]  WITH CHECK ADD
CONSTRAINT [FK_ConcernTablesReservations_Restaurants] FOREIGN
KEY([CustomerID])
REFERENCES [dbo].[Restaurants] ([RestaurantID])
GO

ALTER TABLE [dbo].[ConcernTablesReservations] CHECK CONSTRAINT
[FK_ConcernTablesReservations_Restaurants]
GO

ALTER TABLE [dbo].[ConcernTablesReservations]  WITH CHECK ADD
CONSTRAINT [FK_ConcernTablesReservations_TableRestrictions] FOREIGN
KEY([TableID])
REFERENCES [dbo].[TableRestrictions] ([TableID])
GO

ALTER TABLE [dbo].[ConcernTablesReservations] CHECK CONSTRAINT
[FK_ConcernTablesReservations_TableRestrictions]
GO

ALTER TABLE [dbo].[ConcernTablesReservations]  WITH CHECK ADD
CONSTRAINT [CK_ConcernTablesReservations_Guests] CHECK
(([Guests]>=(0)))
GO

ALTER TABLE [dbo].[ConcernTablesReservations] CHECK CONSTRAINT
[CK_ConcernTablesReservations_Guests]
GO

--END ConcernTableReservations
```

22. ConcernNamesReservation

Tabela zawierająca imiona gości wchodzących w skład rezerwacji stolika przez firmę.
Klucz główny to trójka ReservationID-TableID-GuestNumber.
Zawiera klucze:

-ReservationID - identyfikator rezerwacji [typ int]
-TableID - identyfikator stolika [typ int]
-GuestNumber - numer klienta (liczba porządkowa) [typ tinyint]
-GuestName - imię klienta [typ nvarchar(50)]

Warunki integralności :

-GuestNumber jest liczbą większą od zera.

```
--START ConcernNamesReservation
CREATE TABLE [dbo].[ConcernNamesReservation](
      [ReservationID] [int] NOT NULL,
      [TableID] [int] NOT NULL,
      [GuestNumber] [tinyint] NOT NULL,
      [GuestName] [varchar](50) NOT NULL,
 CONSTRAINT [PK_ConcernNamesReservation] PRIMARY KEY CLUSTERED
(
      [ReservationID] ASC,
      [TableID] ASC,
      [GuestNumber] 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].[ConcernNamesReservation]  WITH CHECK ADD  CONSTRAINT
[FK_ConcernNamesReservation_ConcernTablesReservations] FOREIGN
KEY([ReservationID], [TableID])
REFERENCES [dbo].[ConcernTablesReservations] ([ReservationID],
[TableID])
GO

ALTER TABLE [dbo].[ConcernNamesReservation] CHECK CONSTRAINT
[FK_ConcernNamesReservation_ConcernTablesReservations]
GO

ALTER TABLE [dbo].[ConcernNamesReservation]  WITH CHECK ADD  CONSTRAINT
[CK_ConcernNamesReservation] CHECK  (([GuestNumber]>(0)))
GO

ALTER TABLE [dbo].[ConcernNamesReservation] CHECK CONSTRAINT
```

```
[CK_ConcernNamesReservation]
GO
--END ConcernNamesReservation
```

23. Employees

Tabela zawierająca informacje o pracownikach, klucz główny to EmployeeID.
Zawiera klucze:

-EmployeeID - identyfikator pracownika [typ int]
-RestaurantID - identyfikator restauracji w której pracuje [typ int]
-FirstName - imię pracownika [typ nvarchar[50]]
-LastName - nazwisko pracownika [typ nvarchar[50]]
-DateEmployed - data zatrudnienia [typ datetime]
-DateFired - data zwolnienia, może być nullem [typ datetime]

Warunki integralności:

-DateFired jest datą późniejszą niż DateEmployed, lub jest nullem.

```
--START Employees
CREATE TABLE [dbo].[Employees](
	[EmployeeID] [int] NOT NULL,
	[RestaurantID] [int] NOT NULL,
	[FirstName] [nvarchar](50) NOT NULL,
	[LastName] [nvarchar](50) NOT NULL,
	[DateEmployed] [date] NOT NULL,
	[DateFired] [date] NULL,
 CONSTRAINT [PK_Employees] PRIMARY KEY CLUSTERED
(
	[EmployeeID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Employees]  WITH CHECK ADD  CONSTRAINT
[FK_Employees_Restaurants] FOREIGN KEY([RestaurantID])
REFERENCES [dbo].[Restaurants] ([RestaurantID])
GO

ALTER TABLE [dbo].[Employees] CHECK CONSTRAINT
[FK_Employees_Restaurants]
GO

ALTER TABLE [dbo].[Employees]  WITH CHECK ADD  CONSTRAINT [CK_Employees]
CHECK  (([DateFired] IS NULL OR [DateFired]>[DateEmployed]))
GO
```

```
ALTER TABLE [dbo].[Employees] CHECK CONSTRAINT [CK_Employees]
GO

--END Employees
```

4. Widoki

A. v_customers_orders_all(CustomerID,FirstName,LastName,Email,RestaurantID, OrderID, OrderDate,OrderFinishedDate)
Lista wszystkich zamówień dla każdego klienta

```
CREATE VIEW [dbo].[v_customers_orders_all]
AS
SELECT      dbo.Customers.CustomerID,
dbo.Customers.FirstName,
dbo.Customers.LastName,
dbo.IndividualCustomers.Email AS [Email IndividualCustomer],
dbo.ConcernCustomers.Email AS [Email ConcernCustomer],
dbo.Orders.RestaurantID,
dbo.Orders.OrderID,
dbo.Orders.OrderDate,
dbo.Orders.OrderFinishedDate
FROM        dbo.Customers
LEFT OUTER JOIN
            dbo.IndividualCustomers
ON    dbo.Customers.CustomerID = dbo.IndividualCustomers.CustomerID
LEFT OUTER JOIN
            dbo.ConcernCustomers
ON    dbo.Customers.CustomerID = dbo.ConcernCustomers.CustomerID
LEFT OUTER JOIN
            dbo.Orders
ON dbo.Customers.CustomerID = dbo.Orders.CustomerID
WHERE  (dbo.Orders.OrderDate IS NOT NULL)
GO
```

B. v_customers_orders_waiting(CustomerID,FirstName,LastName,Email,RestaurantID, OrderID, OrderDate)
Lista zamówień oczekujących dla każdego klienta (data zakończenia realizacji zamówienia jest nullem)

```
CREATE VIEW [dbo].[v_customers_orders_waiting]
AS
SELECT dbo.Customers.CustomerID,
dbo.Customers.FirstName,
dbo.Customers.LastName,
dbo.IndividualCustomers.Email AS [Email IndividualCustomer],
```

```
dbo.ConcernCustomers.Email AS [Email ConcernCustomer],
dbo.Orders.RestaurantID,
dbo.Orders.OrderID,
dbo.Orders.OrderDate
FROM     dbo.Customers
LEFT OUTER JOIN
                  dbo.ConcernCustomers
ON dbo.ConcernCustomers.CustomerID = dbo.Customers.CustomerID
LEFT OUTER JOIN
                  dbo.IndividualCustomers
ON dbo.Customers.CustomerID = dbo.IndividualCustomers.CustomerID
LEFT OUTER JOIN
                  dbo.Orders
ON dbo.Customers.CustomerID = dbo.Orders.CustomerID
WHERE  (dbo.Orders.OrderFinishedDate IS NULL) AND (dbo.Orders.OrderDate
IS NOT NULL)
GO
```

C. v_customers_orders_finished(CustomerId,FirstName,LastName,Email,RestaurantID, OrderID, OrderDate,OrderFinishedDate)
Lista zakończonych zamówień dla każdego klienta

```
CREATE VIEW [dbo].[v_customers_orders_finished]
AS
SELECT dbo.Customers.CustomerID,
dbo.Customers.FirstName,
dbo.Customers.LastName,
dbo.IndividualCustomers.Email AS [Email IndividualCustomer],
dbo.ConcernCustomers.Email AS [Email ConcernCustomer],
dbo.Orders.RestaurantID,
dbo.Orders.OrderID,
dbo.Orders.OrderDate,
dbo.Orders.OrderFinishedDate
FROM     dbo.Customers
LEFT OUTER JOIN
                  dbo.ConcernCustomers
ON dbo.ConcernCustomers.CustomerID = dbo.Customers.CustomerID
LEFT OUTER JOIN
                  dbo.IndividualCustomers
ON dbo.Customers.CustomerID = dbo.IndividualCustomers.CustomerID
LEFT OUTER JOIN
                  dbo.Orders
ON dbo.Customers.CustomerID = dbo.Orders.CustomerID
WHERE  (dbo.Orders.OrderFinishedDate IS NOT NULL) AND
```

```
(dbo.Orders.OrderDate IS NOT NULL)
GO
```

D. v_current_individual_reservations(CustomerID,FirstName,LastName,Email,ReservationID,RestaurantID,OrderID,TableID,ReservationDate,DateReserved,paid)
Lista wszystkich rezerwacji klientów indywidualnych z dołączonymi danymi osobowymi klienta

```
CREATE VIEW [dbo].[v_current_individual_reservations]
AS
SELECT dbo.Customers.CustomerID,
dbo.Customers.FirstName,
dbo.Customers.LastName,
dbo.IndividualCustomers.Email,
dbo.IndividualReservations.ReservationID,
dbo.IndividualReservations.RestaurantID,
dbo.IndividualReservations.OrderID,
dbo.IndividualReservations.TableID,
dbo.IndividualReservations.ReservationDate,
dbo.IndividualReservations.DateReserved,
dbo.IndividualReservations.Paid
FROM     dbo.IndividualCustomers
INNER JOIN
                  dbo.Customers
ON dbo.IndividualCustomers.CustomerID = dbo.Customers.CustomerID
INNER JOIN
                  dbo.IndividualReservations
ON dbo.Customers.CustomerID = dbo.IndividualReservations.CustomerID
WHERE  (dbo.IndividualReservations.DateReserved > GETDATE())
GO
```

E. v_current_individual_reservations_paid(CustomerID,FirstName,LastName,Email,ReservationID,RestaurantID,OrderID,TableID,ReservationDate,DateReserved,paid)
Lista opłaconych z góry rezerwacji klientów indywidualnych z dołączonymi danymi osobowymi klienta

```
CREATE VIEW [dbo].[v_current_individual_reservations_paid]
AS
SELECT dbo.Customers.CustomerID,
dbo.Customers.FirstName,
dbo.Customers.LastName,
dbo.IndividualCustomers.Email,
dbo.IndividualReservations.ReservationID,
dbo.IndividualReservations.RestaurantID,
```

```
dbo.IndividualReservations.OrderID,
dbo.IndividualReservations.TableID,
dbo.IndividualReservations.ReservationDate,
dbo.IndividualReservations.DateReserved,
dbo.IndividualReservations.Paid
FROM     dbo.Customers
INNER JOIN
                  dbo.IndividualCustomers
ON dbo.Customers.CustomerID = dbo.IndividualCustomers.CustomerID
INNER JOIN
                  dbo.IndividualReservations
ON dbo.Customers.CustomerID = dbo.IndividualReservations.CustomerID
WHERE  (dbo.IndividualReservations.Paid = 1) AND
(dbo.IndividualReservations.DateReserved > GETDATE())
GO
```

F. v_current_individual_reservations_unpaid(CustomerID,FirstName,LastName, Email,ReservationID,RestaurantID,OrderID,TableID,ReservationDate,DateReserved,paid)
Lista rezerwacji klientów indywidualnych nie opłaconych z góry, z dołączonymi danymi osobowymi klienta

```
CREATE VIEW [dbo].[v_current_individual_reservations_unpaid]
AS
SELECT dbo.Customers.CustomerID,
dbo.Customers.FirstName,
dbo.Customers.LastName,
dbo.IndividualCustomers.Email,
dbo.IndividualReservations.ReservationID,
dbo.IndividualReservations.RestaurantID,
dbo.IndividualReservations.OrderID,
dbo.IndividualReservations.TableID,
dbo.IndividualReservations.ReservationDate,
dbo.IndividualReservations.DateReserved,
dbo.IndividualReservations.Paid
FROM     dbo.Customers
INNER JOIN
                  dbo.IndividualCustomers
ON dbo.Customers.CustomerID = dbo.IndividualCustomers.CustomerID INNER
JOIN
                  dbo.IndividualReservations
ON dbo.Customers.CustomerID = dbo.IndividualReservations.CustomerID
WHERE  (dbo.IndividualReservations.DateReserved > GETDATE()) AND
(dbo.IndividualReservations.Paid = 0)
GO
```

G. v_menu_dishes(DishID,DishName,CategoryName,UnitPrice)
Lista dań w aktualnym menu z ich ceną, nazwą oraz nazwą kategorii do której przynależy

```
CREATE VIEW [dbo].[v_menu_dishes]
AS
SELECT        dbo.Menu.DishID,
dbo.Dishes.DishName,
dbo.Categories.CategoryName,
dbo.Dishes.UnitPrice
FROM            dbo.Menu
INNER JOIN
                         dbo.Dishes
ON dbo.Menu.DishID = dbo.Dishes.DishID
INNER JOIN
                         dbo.Categories
ON dbo.Dishes.CategoryID = dbo.Categories.CategoryID
WHERE        (GETDATE() BETWEEN dbo.Menu.StartDate AND dbo.Menu.EndDate)
GO
```

H. v_menu_dishes_available(RestaurantID,DishID,DishName,CategoryName,UnitPrice)
Lista dań w menu dostępnych osobno dla każdej restauracji w sieci.

```
CREATE VIEW [dbo].[v_menu_dishes_available]
AS
SELECT        dbo.RestaurantMenuAvailability.RestaurantID,
dbo.RestaurantMenuAvailability.DishID,
dbo.Dishes.DishName,
dbo.Categories.CategoryName,
dbo.Dishes.UnitPrice
FROM            dbo.Menu INNER JOIN
                         dbo.RestaurantMenuAvailability
ON dbo.Menu.DishID = dbo.RestaurantMenuAvailability.DishID
INNER JOIN
                         dbo.Dishes
ON dbo.Menu.DishID = dbo.Dishes.DishID
INNER JOIN
                         dbo.Categories
ON dbo.Dishes.CategoryID = dbo.Categories.CategoryID
WHERE        (GETDATE() BETWEEN dbo.Menu.StartDate AND dbo.Menu.EndDate)
AND (dbo.RestaurantMenuAvailability.Available = 1)
GO
```

I. v_menu_dishes_ingredients(DishID,DishName,CategoryName,IngredientID,Name,QuantityRequired)

Lista składników potrzebnych do wytworzenia dań w obecnym menu.

```
CREATE VIEW [dbo].[v_menu_dishes_ingredients]
AS
SELECT        dbo.Menu.DishID,
dbo.Dishes.DishName,
dbo.Categories.CategoryName,
dbo.Ingredients.IngredientID,
dbo.Ingredients.Name AS IngredientName,
dbo.DishPreparationDetails.QuantityRequired
FROM            dbo.Menu INNER JOIN
                         dbo.Dishes
ON dbo.Menu.DishID = dbo.Dishes.DishID
INNER JOIN
                         dbo.Categories
ON dbo.Dishes.CategoryID = dbo.Categories.CategoryID
INNER JOIN
                         dbo.DishPreparationDetails
ON dbo.Dishes.DishID = dbo.DishPreparationDetails.DishID
INNER JOIN
                         dbo.Ingredients
ON dbo.DishPreparationDetails.IngredientID =
dbo.Ingredients.IngredientID
WHERE        (GETDATE() BETWEEN dbo.Menu.StartDate AND dbo.Menu.EndDate)
GO
```

J. v_customers_discounts(CustomerID,FirstName,LastName, DiscountID, DateGranted,DateExpiring, CountDiscount)
Lista klientów wraz z obowiązującymi ich zniżkami.

```
CREATE VIEW [dbo].[v_customers_discounts]
AS
SELECT dbo.Customers.CustomerID, dbo.Customers.FirstName,
dbo.Customers.LastName, dbo.Discounts.DiscountID,
dbo.Discounts.DateGranted, dbo.Discounts.DateExpiring,
dbo.Discounts.CountDiscount
FROM     dbo.Customers INNER JOIN
                  dbo.Discounts ON dbo.Customers.CustomerID =
dbo.Discounts.CustomerID INNER JOIN
                  dbo.IndividualDiscounts ON dbo.Discounts.DiscountID =
dbo.IndividualDiscounts.DiscountID INNER JOIN
                  dbo.ConcernDiscounts ON dbo.Discounts.DiscountID =
dbo.ConcernDiscounts.DiscountID
GO
```

K. v_reservations_tables_names(ReservationID, TableID, RestaurantID,

CustomerID, ReservationDate, DateReserved, GuestNames)
Lista rezerwacji stolików wraz z listą imienną wszystkich osób dla danej rezerwacji.

```
CREATE VIEW [dbo].[v_reservations_tables_names]
AS
SELECT dbo.ConcernTablesReservations.ReservationID,
    dbo.ConcernTablesReservations.TableID,
    dbo.ConcernTablesReservations.RestaurantID,
    dbo.ConcernTablesReservations.CustomerID,
    dbo.ConcernTablesReservations.ReservationDate,
    dbo.ConcernTablesReservations.DateReserved,
    dbo.ConcernNamesReservation.GuestName
FROM dbo.ConcernTablesReservations
INNER JOIN
    dbo.ConcernNamesReservation
ON
    dbo.ConcernTablesReservations.ReservationID =
    dbo.ConcernNamesReservation.ReservationID AND
    dbo.ConcernTablesReservations.TableID =
    dbo.ConcernNamesReservation.TableID
GO
```

```
CREATE VIEW [dbo].[v_reservations_tables_names]
AS
SELECT dbo.ConcernTablesReservations.ReservationID,
    dbo.ConcernTablesReservations.TableID,
    dbo.ConcernTablesReservations.RestaurantID,
    dbo.ConcernTablesReservations.CustomerID,
    dbo.ConcernTablesReservations.ReservationDate,
    dbo.ConcernTablesReservations.DateReserved,
    dbo.ConcernNamesReservation.GuestName
FROM dbo.ConcernTablesReservations INNER JOIN
    dbo.ConcernNamesReservation ON
    dbo.ConcernTablesReservations.ReservationID
    dbo.ConcernNamesReservation.ReservationID AND
    dbo.ConcernTablesReservations.TableID =
    dbo.ConcernNamesReservation.TableID
GO
```

- v_report_tables_reservations
- v_report_tables_reservations_month
- v_report_tables_reservations_week
-
- v_report_discounts_added
- v_report_discounts_added_month
- v_report_discounts_expired_month

- v_report_discounts_expired
- v_report_discounts_added_week
- v_reprot_discounts_expired_week
-
- v_report_menu_changed_month
- v_report_menu_changed_week
-
- v_report_individual_order_statistics_month
- v_report_individual_order_statistics_week
- v_report_concern_order_statistics_month
- v_report_concern_order_statistics_week
-
- v_report_individual_customers_orders
- v_report_concern_customers_orders
- v_report_individual_customers_discounts
- v_report_concern_customers_discounts
-
-
-

5. Procedury

- **Dodające dane**
  - p_add_individual_customer(FirstName,LastName,Email,Phone)

```
CREATE procedure [dbo].[p_add_individual_customer]
@FirstName NVARCHAR(50), @LastName NVARCHAR(50), @Email NVARCHAR(50),
@Phone VARCHAR(20)
AS
BEGIN
 INSERT Customers(FirstName,LastName)
 VALUES(@FirstName,@LastName)

 INSERT IndividualCustomers(CustomerID, Email, Phone)
 VALUES(SCOPE_IDENTITY(), @Email, @Phone)

END
GO
```

  - p_add_concern_customer(FirstName,LastName,CompanyName ,Email,Phone,NIP)

```
CREATE procedure [dbo].[p_add_concern_customer]
@FirstName NVARCHAR(50), @LastName NVARCHAR(50),@CompanyName
NVARCHAR(50),  @Email NVARCHAR(50), @Phone VARCHAR(20), @NIP char(10)
AS
BEGIN
 INSERT Customers(FirstName,LastName)
```

```
 VALUES(@FirstName,@LastName)

 INSERT ConcernCustomers(CustomerID, Email, Phone, CompanyName, NIP)
 VALUES(SCOPE_IDENTITY(), @Email, @Phone, @CompanyName, @NIP)

END
GO
```

- p_add_new_order(RestaurantID, CustomerID, OrderDate, EmployeeID, ConcernOrder, DishID_Quantity_List)

```
CREATE PROCEDURE [dbo].[p_add_new_order]
@RestaurantID int, @CustomerID int, @OrderDate datetime,@FinishedDate
datetime, @EmployeeID int, @ConcernOrder bit, @DishIDQuantityList
DishID_Quantity_List READONLY
AS
BEGIN
 INSERT Orders(RestaurantID, CustomerID, EmployeeID,
OrderDate,OrderFinishedDate, ConcernOrder)
 VALUES(@RestaurantID, @CustomerID, @EmployeeID,
@OrderDate,@FinishedDate, @ConcernOrder)

  INSERT OrderDetails(OrderID, DishID, Quantity)
  SELECT SCOPE_IDENTITY(), DishID, Quantity FROM @DishIDQuantityList

END
GO
```

- p_add_dish_to_order(OrderID, DishID, Quantity)

```
CREATE procedure [dbo].[p_add_dish_to_order]
@OrderID INT, @DishID INT, @Quantity INT
AS
BEGIN
 INSERT OrderDetails(OrderID,DishID,Quantity)
 VALUES(@OrderID,@DishID,@Quantity)
END

GO
```

- p_add_new_restaurant(Adress, City, Country)

```
CREATE procedure [dbo].[p_add_new_restaurant]
@Adress NVARCHAR(50), @City NVARCHAR(50), @Country NVARCHAR(50)
AS
```

```
BEGIN
 INSERT Restaurants(Address, City, Country)
 VALUES(@Adress, @City, @Country)
END
GO
```

- p_add_new_employee(FirstName,LastName,RestaurantID,DateEmployed)

```
CREATE procedure [dbo].[p_add_new_employee]
@FirstName NVARCHAR(50), @LastName NVARCHAR(50), @RestaurantID INT,
@DateEmployed DATETIME
AS
BEGIN
 INSERT Employees(FirstName, LastName, RestaurantID, DateEmployed)
 VALUES(@FirstName,@LastName,@RestaurantID,@DateEmployed)
END
GO
```

- p_add_new_category(CategoryName)

```
CREATE procedure [dbo].[p_add_new_category]
@CategoryName NVARCHAR(50)
AS
BEGIN
 INSERT Categories(CategoryName)
 VALUES(@CategoryName)
END

GO
```

- **p_add_new_dish(CategoryID, DishName, UnitPrice)**

```
CREATE procedure [dbo].[p_add_new_dish]
@CategoryID INT, @DishName NVARCHAR(50), @UnitPrice MONEY
AS
BEGIN
 INSERT Dishes(CategoryID,DishName, UnitPrice)
 VALUES(@CategoryID, @DishName, @UnitPrice)
END

GO
```

- **p_add_new_dish_with_ingredients (CategoryID, DishName,**

UnitPrice)

```
CREATE PROCEDURE [dbo].[p_add_new_dish_with_ingredients]
@CategoryID INT, @DishName NVARCHAR(50), @UnitPrice MONEY,
@IngredientIDQuantityRequiredList IngredientID_QuantityRequired_List
READONLY
AS
BEGIN
 INSERT Dishes(CategoryID,DishName, UnitPrice)
 VALUES(@CategoryID, @DishName, @UnitPrice)
 INSERT DishPreparationDetails(DishID,IngredientID,QuantityRequired)
 SELECT SCOPE_IDENTITY(), IngredientID, QuantityRequired FROM
@IngredientIDQuantityRequiredList
END
GO
```

- **p_add_dish_to_menu (DishID, StartDate, EndDate)**

```
CREATE procedure [dbo].[p_add_dish_to_menu]
@DishID INT, @StartDate DATETIME , @EndDate DATETIME
AS
BEGIN
IF (EXISTS (SELECT * FROM Menu WHERE DishID=@DishID))
      BEGIN
            UPDATE Menu
            SET StartDate=@StartDate, EndDate=@EndDate
            WHERE DishID=@DishID
      END
ELSE
      BEGIN
             INSERT Menu(DishID, StartDate, EndDate)
             VALUES(@DishID, @StartDate, @EndDate)
       END
END
GO
```

- p_add_new_ingredient(IngredientName)

```
CREATE procedure [dbo].[p_add_new_ingredient]
@IngredientName NVARCHAR(50)
AS
BEGIN
 INSERT Ingredients(Name)
 VALUES(@IngredientName)
END

GO
```

- p_add_new_table(RestaurantID, Seats)

```
CREATE procedure [dbo].[p_add_new_table]
@RestaurantID INT, @Seats INT
AS
BEGIN
 INSERT Tables(RestaurantID, Seats)
 VALUES(@RestaurantID,@Seats)
END

GO
```

- p_add_tablerestriction(TableID, Seats, StartDate, EndDate)

```
CREATE procedure [dbo].[p_add_tablerestriction]
@TableID INT, @Seats INT, @StardDate DATETIME , @EndDate DATETIME
AS
BEGIN
 INSERT TableRestrictions(TableID, Seats, StartDate, EndDate)
 VALUES(@TableID, @Seats, @StardDate, @EndDate)
END

GO
```

- p_add_new_individual_discount(Z, K1, K2, R, StartDate, EndDate, Duration, SingleUse)

```
CREATE procedure [dbo].[p_add_new_individual_discount]
@Z INT, @K1 INT, @K2 INT, @R REAL, @StartDate DATETIME, @EndDate
DATETIME ,@Duration INT, @SingleUse BIT
AS
BEGIN
 INSERT Discounts(StartDate, EndDate)
 VALUES(@StartDate, @EndDate)

 INSERT IndividualDiscounts(DiscountID, Z, K1, K2, R, Duration,
SingleUse)
 VALUES (SCOPE_IDENTITY(), @Z, @K1, @K2, @R, @Duration, @SingleUse)
END

GO
```

- p_add_new_concern_discount(PeriodOfTime, FZ, FK, FR, FM StartDate, EndDate)

```
CREATE PROCEDURE [dbo].[p_add_new_concern_discount]
```

```
      @PeriodOfTime char(1),
      @FZ int,
      @FK money,
      @FR real,
      @FM real,
      @StartDate datetime,
      @EndDate datetime

AS
BEGIN
      SET NOCOUNT ON;

      INSERT Discounts(StartDate, EndDate)
      VALUES (@StartDate, @EndDate)

      INSERT ConcernDiscounts(DiscountID, PeriodOfTime, FZ, FK, FR, FM)
      VALUES (SCOPE_IDENTITY(), @PeriodOfTime, @FZ, @FK, @FR, @FM)
END
GO
```

- p_add_individual_reservation(CustomerID,RestaurantID,OrderID,TableID,ReservationDate,DateReserved,Guests,Paid)

```
CREATE PROCEDURE [dbo].[p_add_individual_reservation]
      @CustomerID int,
      @RestaurantID int,
      @OrderID int,
      @TableID int,
      @ReservationDate datetime,
      @DateReserved datetime,
      @Guests int,
      @Paid bit
AS
BEGIN
      SET NOCOUNT ON;

      INSERT IndividualReservations(CustomerID, RestaurantID, OrderID,
TableID, ReservationDate, DateReserved, Guests, Paid)
      VALUES (@CustomerID, @RestaurantID, @OrderID, @TableID,
@ReservationDate, @DateReserved, @Guests, @Paid)
END
GO
```

- p_add_concern_table_reservation(TableID, RestaurantID, CustomerID, ReservationDate, DateReserved, GuestList)

```
CREATE PROCEDURE [dbo].[p_add_concern_table_reservation]
```

```
	@TableID int,
	@CustomerID int,
	@ReservationDate datetime,
	@DateReserved datetime,
	@GuestList Guest_List READONLY
AS
BEGIN
	SET NOCOUNT ON;

	DECLARE @RestaurantID int
	DECLARE @Guests int


	SELECT @RestaurantID=RestaurantID FROM Tables WHERE TableID =
@TableID
	SELECT @Guests=COUNT(*) FROM @GuestList


	INSERT ConcernTablesReservations(TableID, RestaurantID,
CustomerID, ReservationDate, DateReserved, Guests)
	VALUES (@TableID, @RestaurantID, @CustomerID, @ReservationDate,
@DateReserved, @Guests)


	INSERT ConcernNamesReservation(ConcernReservationID, TableID,
GuestNumber, GuestName)
	SELECT SCOPE_IDENTITY(), @TableID, GuestNumber, GuestName FROM
@GuestList

END
GO
```

- p_add_individual_reservation_with_order(CustomerID,RestaurantID int,TableID,ReservationDate,DateReserved,Guests,Paid, EmployeeID int, DishIDQuantityList)

```
CREATE PROCEDURE [dbo].[p_add_individual_reservation_with_order]
		-- Add the parameters for the stored procedure here
	@CustomerID int,
	@RestaurantID int,
	@TableID int,
	@ReservationDate datetime,
	@DateReserved datetime,
```

```
	@Guests int,
	@Paid bit,
	@EmployeeID int,
	@DishIDQuantityList DishID_Quantity_List READONLY
AS
BEGIN
	-- SET NOCOUNT ON added to prevent extra result sets from
	-- interfering with SELECT statements.
	SET NOCOUNT ON;

	DECLARE @FinishedDate datetime = NULL
	DECLARE @OrderID int

	IF (@DateReserved<GETDATE())
	BEGIN
		SELECT @FinishedDate=@DateReserved
	END

	EXEC @OrderID=[dbo].[p_add_new_order] @RestaurantID, @CustomerID,
@ReservationDate, @FinishedDate, @EmployeeID, 0, @DishIdQuantityList
	IF (@OrderID IS NOT NULL AND @OrderID>0)
	BEGIN
		EXEC [dbo].[p_add_individual_reservation] @CustomerID,
@RestaurantID, @OrderID, @TableID, @ReservationDate, @DateReserved,
@Guests, @Paid
	END
END
GO
```

- p_add_new_dish_with_ingredients(CategoryID, DishName, UnitPrice, IngredientIDQuantityRequiredList)

```
CREATE PROCEDURE [dbo].[p_add_new_dish_with_ingredients]
	-- Add the parameters for the stored procedure here
@CategoryID INT, @DishName NVARCHAR(50), @UnitPrice MONEY,
@IngredientIDQuantityRequiredList IngredientID_QuantityRequired_List
READONLY
AS
BEGIN
 INSERT Dishes(CategoryID,DishName, UnitPrice)
 VALUES(@CategoryID, @DishName, @UnitPrice)
 INSERT DishPreparationDetails(DishID,IngredientID,QuantityRequired)
 SELECT SCOPE_IDENTITY(), IngredientID, QuantityRequired FROM
@IngredientIDQuantityRequiredList
```

```
END
GO
```

- **Przypisujące dane**
  - **p_add_ingredient_to_dish(DishID, IngredientID, QuantityRequired)**

```
CREATE procedure [dbo].[p_add_ingredient_to_dish]
@DishID INT, @IngredientID INT, @QuantityRequired INT
AS
BEGIN
 INSERT DishPreparationDetails(DishID,IngredientID,QuantityRequired)
 VALUES(@DishID, @IngredientID, @QuantityRequired)
END

GO
```

  - p_add_dish_to_menu(DishID,StartDate,EndDate)

```
CREATE procedure [dbo].[p_add_dish_to_menu]
@DishID INT, @StartDate DATETIME , @EndDate DATETIME
AS
BEGIN
IF (EXISTS (SELECT * FROM Menu WHERE DishID=@DishID))
      BEGIN
            UPDATE Menu
            SET StartDate=@StartDate, EndDate=@EndDate
            WHERE DishID=@DishID
      END
ELSE
      BEGIN
             INSERT Menu(DishID, StartDate, EndDate)
             VALUES(@DishID, @StartDate, @EndDate)
       END
END
```

  - p_give_discounts_to_customer (CustomerID, DateGranted)

```
CREATE PROCEDURE [dbo].[p_give_discounts_to_customer]

      @CustomerID int,
      @DateGranted datetime

AS
BEGIN

      SET NOCOUNT ON;
```

```
	DECLARE @DiscountsToAdd TABLE
	(
		DiscountID int,
		DateExpiring datetime,
		CountDiscount int
	)



	--Individual, not SingleUse
	INSERT INTO @DiscountsToAdd (DiscountID, DateExpiring,
CountDiscount)
	SELECT tmp.DiscountID, (NULL), ([dbo].[f_min_two_ints]
(tmp.amount/tmp.Z, 2)) FROM (
		SELECT D.DiscountID, StartDate, EndDate, Z, SingleUse,

([dbo].[f_get_amount_of_orders_by_individual_customers_costing_at_least]
(@CustomerID, K1)) AS amount
		FROM Discounts AS D
		INNER JOIN IndividualDiscounts ID on D.DiscountID =
ID.DiscountID
	) as tmp
	WHERE @DateGranted>=tmp.StartDate AND (tmp.EndDate IS NULL OR
@DateGranted<=tmp.EndDate)
	AND tmp.SingleUse=0 AND tmp.amount >= tmp.Z


	--Individual, SingleUse
	DECLARE @Duration int
	--w tym selekcie leci warning
	print('a')
	INSERT INTO @DiscountsToAdd(DiscountID, DateExpiring,
CountDiscount)
	SELECT tmp.DiscountID, DATEADD(DAY, tmp.Duration, @DateGranted), 1
FROM (
		SELECT D.DiscountID, StartDate, EndDate, SingleUse, K2,
Duration,

([dbo].[f_calculate_total_amount_spent_by_individual_customer]
(@CustomerID)) AS amount
		FROM Discounts AS D
		INNER JOIN IndividualDiscounts ID on D.DiscountID =
ID.DiscountID
	) as tmp
	WHERE @DateGranted>=tmp.StartDate AND (tmp.EndDate IS NULL OR
```

```
@DateGranted<=tmp.EndDate)
      AND tmp.SingleUse=1 AND tmp.amount >= tmp.K2
      print('b')


      --Concern, monthly discount
      INSERT INTO @DiscountsToAdd(DiscountID, DateExpiring,
CountDiscount)
      SELECT tmp.DiscountID, DATEADD(MONTH, 1, @DateGranted),
[dbo].[f_calculate_count_discount_for_monthly_concern_discounts](@DateGr
anted, @CustomerID, FK, FZ) FROM (
            SELECT D.DiscountID, StartDate, EndDate, FZ, FK, FR,
PeriodOfTime
            FROM Discounts AS D
            INNER JOIN ConcernDiscounts CD on D.DiscountID =
CD.DiscountID
      ) as tmp
      WHERE @DateGranted>=tmp.StartDate AND (tmp.EndDate IS NULL OR
@DateGranted<=tmp.EndDate)
      AND tmp.PeriodOfTime LIKE 'M'


      --Concern, quarterly discount
      INSERT INTO @DiscountsToAdd(DiscountID, DateExpiring,
CountDiscount)
      SELECT tmp.DiscountID, DATEADD(QUARTER, 1, @DateGranted),
[dbo].[f_calculate_count_discount_for_quarterly_concern_discounts](@Date
Granted, @CustomerID, FK) FROM (
            SELECT D.DiscountID, StartDate, EndDate, FZ, FK, FR,
PeriodOfTime
            FROM Discounts AS D
            INNER JOIN ConcernDiscounts CD on D.DiscountID =
CD.DiscountID
      ) as tmp
      WHERE @DateGranted>=tmp.StartDate AND (tmp.EndDate IS NULL OR
@DateGranted<=tmp.EndDate)
      AND tmp.PeriodOfTime LIKE 'Q'


      -- add all discounts to customer
      DECLARE @DiscountID int
      DECLARE @DateExpiring datetime
      DECLARE @CountDiscount int
      DECLARE @DiscountsLeft int = (SELECT COUNT(*) FROM
@DiscountsToAdd)
```

```
	WHILE (@DiscountsLeft>0)
	BEGIN
		SELECT TOP 1 @DiscountID=DiscountID,
@DateExpiring=DateExpiring, @CountDiscount=CountDiscount FROM
@DiscountsToAdd
		EXEC [dbo].[p_add_discount_to_customer] @CustomerID,
@DiscountID, @DateGranted, @DateExpiring, @CountDiscount

		DELETE @DiscountsToAdd WHERE DiscountID=@DiscountID
		SELECT @DiscountsLeft = @DiscountsLeft-1
	END

END
GO
```

- p_add_discount_to_customer(CustomerID,DiscountID,DateGranted,DateExpiring)

```
CREATE PROCEDURE [dbo].[p_add_discount_to_customer]
	-- Add the parameters for the stored procedure here
	@CustomerID int,
	@DiscountID int,
	@DateGranted datetime,
	@DateExpiring datetime,
	@CountDiscount INT
AS
BEGIN
	-- SET NOCOUNT ON added to prevent extra result sets from
	-- interfering with SELECT statements.
	SET NOCOUNT ON;


    -- Insert statements for procedure here
	IF (EXISTS (SELECT * FROM CustomerDiscounts WHERE
DiscountID=@DiscountID AND CustomerID=@CustomerID))
	BEGIN
		UPDATE CustomerDiscounts
		SET CountDiscount=@CountDiscount, DateExpiring=@DateExpiring
		WHERE DiscountID=@DiscountID AND CustomerID=@CustomerID
	END
	ELSE
	BEGIN
		INSERT CustomerDiscounts(CustomerID, DiscountID,
DateGranted, DateExpiring,CountDiscount)
```

```
            VALUES (@CustomerID, @DiscountID, @DateGranted,
@DateExpiring, @CountDiscount)
      END
END
GO
```

- **Modyfikujące dane**
  - p_finish_order(OrderID, OrderFinishedDate)

```
CREATE procedure [dbo].[p_finish_order]
@OrderID INT, @OrderFinishedDate DATETIME
AS
BEGIN
 UPDATE Orders
 SET OrderFinishedDate = @OrderFinishedDate
 WHERE OrderID = @OrderID
END


GO
```

  - p_fire_employee(EmployeeID,DateFired)

```
CREATE procedure [dbo].[p_fire_employee]
@EmployeeID INT, @DateFired DATETIME
AS
BEGIN
 UPDATE Employees
 SET DateFired = @DateFired
 WHERE EmployeeID = @EmployeeID
END

GO
```

  - p_change_dish_price(DishID,UnitPrice)

```
CREATE procedure [dbo].[p_change_dish_price]
@DishID INT, @UnitPrice MONEY
AS
BEGIN
 UPDATE Dishes
```

```
 SET UnitPrice = @UnitPrice
 WHERE DishID = @DishID
END

GO
```

- p_change_dish_availability_in_restaurant(RestaurantID,DishID, value)

```
CREATE procedure [dbo].[p_change_dish_availability_in_restaurant]
@RestaurantID INT, @DishID INT, @Value BIT
AS
BEGIN
 UPDATE RestaurantMenuAvailability
 SET Available = @Value
 WHERE RestaurantID = @RestaurantID
END
GO
```

- p_change_count_discount(DiscountID,CustomerID,CountDiscount)

```
CREATE PROCEDURE [dbo].[p_change_count_discount]
      -- Add the parameters for the stored procedure here
      @DiscountID int,
      @CustomerID int,
      @CountDiscount int
AS
BEGIN
      -- SET NOCOUNT ON added to prevent extra result sets from
      -- interfering with SELECT statements.
      SET NOCOUNT ON;

    -- Insert statements for procedure here
      UPDATE CustomerDiscounts
      SET CountDiscount = @CountDiscount
      WHERE DiscountID=@DiscountID AND CustomerID=@CustomerID
END
GO
```

- p_subtract_dish_ingredients_in_restaurant (RestaurantID, DishID, Quantity)

```
CREATE procedure [dbo].[p_subtract_dish_ingredients_in_restaurant]
@RestaurantID INT, @DishID INT, @Quantity INT
```

```
AS
BEGIN

	DECLARE @IngredientIDQuantityList TABLE (
		IngredientID int,
		QuantityToSubtract int
	)
	DECLARE @IngredientID int, @QuantityToSubtract int, @QuantityLeft
int

	INSERT INTO @IngredientIDQuantityList(IngredientID,
QuantityToSubtract)
	SELECT IngredientID, (QuantityRequired*@Quantity) FROM
DishPreparationDetails
	WHERE DishID=@DishID

	DECLARE @IngredientsLeft int = (SELECT COUNT(*) FROM
@IngredientIDQuantityList)

	BEGIN TRANSACTION
	SAVE TRANSACTION Ingredients_transaction

	IF(@IngredientsLeft<1)
	BEGIN
		ROLLBACK TRANSACTION Ingredients_transaction
		COMMIT
		RAISERROR('Restaurant doesn''t have magazine entry', 1, 1,
69693)
		RETURN (-2)
	END
	WHILE (@IngredientsLeft>0)
	BEGIN
		SELECT TOP 1 @IngredientID=IngredientID,
@QuantityToSubtract=QuantityToSubtract FROM @IngredientIDQuantityList
		UPDATE RestaurantIngredients
		SET UnitsInStock=UnitsInStock-@QuantityToSubtract
		WHERE IngredientID=@IngredientID AND RestaurantID =
@RestaurantID

		SELECT @QuantityLeft=UnitsInStock FROM RestaurantIngredients
WHERE IngredientID=@IngredientID AND RestaurantID=@RestaurantID
		IF(@QuantityLeft<0)
		BEGIN
			ROLLBACK TRANSACTION Ingredients_transaction
			COMMIT
			RAISERROR('Not enough ingredients left!', 1, 1, 69694)
```

```
                  RETURN (-1)
            END
            ELSE
            BEGIN
                  DELETE @IngredientIDQuantityList WHERE
IngredientID=@IngredientID
                  SELECT @IngredientsLeft = @IngredientsLeft-1
            END
      END
      COMMIT
      RETURN 1
END
```

- **Pobierające dane**
  - p_table_reservation_details (TableID)

```
CREATE PROCEDURE [dbo].[p_table_reservation_details]
@TableID INT
AS
BEGIN
 select TableID,RestaurantID,  CustomerID, DateReserved, Guests from
ConcernTablesReservations
 where TableID = @TableID
END
GO
```

  - **p_individual_order_month_statistics (Year, Month)**

```
CREATE PROCEDURE [dbo].[p_individual_order_month_statistics]
      -- Add the parameters for the stored procedure here
      @year int,
      @month int
AS
BEGIN
      -- SET NOCOUNT ON added to prevent extra result sets from
      -- interfering with SELECT statements.
      SET NOCOUNT ON;

    -- Insert statements for procedure here
```

```
	DECLARE @max_Cost money = (
		SELECT MAX([dbo].[f_calculate_total_order_cost] (OrderID))
		FROM Orders
		WHERE YEAR(OrderDate)=@year AND MONTH(OrderDate)=@month AND
ConcernOrder=0
	)

	DECLARE @avg_Cost money = (
		SELECT AVG([dbo].[f_calculate_total_order_cost] (OrderID))
		FROM Orders
		WHERE YEAR(OrderDate)=@year AND MONTH(OrderDate)=@month AND
ConcernOrder=0
	)

	DECLARE @most_spending_customer int
	SELECT @most_spending_customer="CUSTOMER" FROM (
			SELECT TOP 1 CustomerID AS "CUSTOMER", COUNT(*) as
AMOUNT
			FROM Orders
			WHERE YEAR(OrderDate)=@year AND
MONTH(OrderDate)=@month AND ConcernOrder=0
			GROUP BY CustomerID
			ORDER BY AMOUNT DESC
	) AS temp

	DECLARE @busiest_day int
	SELECT @busiest_day="DAY" FROM (
		SELECT TOP 1 DAY(OrderDate) AS "DAY", COUNT(*) as AMOUNT
FROM Orders
		WHERE YEAR(OrderDate)=@year AND MONTH(OrderDate)=@month AND
ConcernOrder=0
		GROUP BY DAY(OrderDate)
		ORDER BY AMOUNT DESC
	) AS temp

	DECLARE @busiest_hour int
	SELECT @busiest_hour="HOUR" FROM (
		SELECT TOP 1 DATEPART(HOUR, OrderDate) AS "HOUR", COUNT(*)
as AMOUNT FROM Orders
		WHERE YEAR(OrderDate)=@year AND MONTH(OrderDate)=@month AND
ConcernOrder=0
		GROUP BY DATEPART(HOUR, (OrderDate))
		ORDER BY AMOUNT DESC
	) AS temp
```

```
      SELECT
            @year AS "Year",
            @month AS "Month",
            @max_Cost AS "Biggest Order Cost",
            @avg_Cost AS "Average Order Cost",
            @most_spending_customer AS "Most Spending Customer",
            @busiest_day AS "Busiest Day",
            @busiest_hour AS "Busiest Hour"
END
GO
```

- **p_table_reservation_details(TableID)**

```
CREATE PROCEDURE [dbo].[p_table_reservation_details]
@TableID INT
AS
BEGIN
 select TableID,RestaurantID,  CustomerID, DateReserved, Guests from
ConcernTablesReservations
 where TableID = @TableID
END
GO
```

- p_subtract_dish_ingredients_in_restaurant(RestaurantID,DishID,Quantity)

```
CREATE procedure [dbo].[p_substract_dish_ingredients_in_restaurant]
@RestaurantID INT, @DishID INT, @Quantity INT
AS
BEGIN
 INSERT RestaurantIngredients(RestaurantID, IngredientID, UnitsInStock)
 VALUES(@RestaurantID,
      (SELECT IngredientID FROM DishPreparationDetails as DPD WHERE
DPD.DishID = @DishID),
      (SELECT QuantityRequired FROM DishPreparationDetails as DPD WHERE
DPD.DishID = @DishID)*@Quantity)
END

GO
```

- p_add_ingredients_in_restaurant(RestaurantID,IngredientID,Quantity)

```
CREATE procedure [dbo].[p_add_ingredients_in_restaurant]
@RestaurantID INT, @IngredientID INT, @Quantity INT
AS
```

```
BEGIN
 INSERT RestaurantIngredients(RestaurantID, IngredientID, UnitsInStock)
 VALUES(@RestaurantID, @IngredientID, @Quantity)
END

GO
```

- p_change_count_discount(DiscountID, CustomerID, CountDiscount)

```
CREATE PROCEDURE [dbo].[p_change_count_discount]

      @DiscountID int,
      @CustomerID int,
      @CountDiscount int
AS
BEGIN
      SET NOCOUNT ON;

      UPDATE CustomerDiscounts
      SET CountDiscount = @CountDiscount
      WHERE DiscountID=@DiscountID AND CustomerID=@CustomerID
END
GO
```

p_change_menu(@DishIDStartDateEndDateList) - procedura zmieniająca dania w menu

```
CREATE PROCEDURE [dbo].[p_change_menu]
      @DishIDStartDateEndDateList DishID_StartDate_EndDate_List READONLY
AS
BEGIN
      -- SET NOCOUNT ON added to prevent extra result sets from
      -- interfering with SELECT statements.
      SET NOCOUNT ON;

    -- Insert statements for procedure here
      DECLARE @NewMenu DishID_StartDate_EndDate_List

      INSERT INTO @NewMenu
      SELECT * FROM @DishIDStartDateEndDateList

      DECLARE @NewMenuDishCount int = (SELECT COUNT(*) FROM @NewMenu)
      DECLARE @OldMenuDishCount int = (SELECT COUNT(*) FROM
v_menu_dishes)
```

```
	DECLARE @DishID int, @StartDate datetime, @EndDate datetime

	IF (@NewMenuDishCount < @OldMenuDishCount/2)
	BEGIN
		RAISERROR ('Not enough dishes changed from old menu!', 1,
696962)
		RETURN
	END

	BEGIN TRANSACTION
	WHILE (@NewMenuDishCount>0)
	BEGIN
		SELECT TOP 1 @DishID=DishID, @StartDate=StartDate,
@EndDate=EndDate FROM @NewMenu
		EXEC p_add_dish_to_menu @DishID, @StartDate, @EndDate

		DELETE @NewMenu WHERE DishID=@DishID
		SELECT @NewMenuDishCount = @NewMenuDishCount - 1
	END
	COMMIT
END
GO
```

6. Funkcje

- f_calculate_order_cost(@OrderID) - Oblicza całkowity koszt zamówienia (bez zniżek)

```
CREATE FUNCTION [dbo].[f_calculate_order_cost]
(
	@OrderID int
)
RETURNS money
AS
BEGIN

	DECLARE @OrderCost money

	SELECT @OrderCost = SUM(D.UnitPrice*OD.Quantity) FROM Orders AS O
	INNER JOIN OrderDetails AS OD ON O.OrderID = OD.OrderID
	INNER JOIN Dishes AS D ON D.DishID = OD.DishID
	WHERE O.OrderID=@OrderID
```

```
	RETURN @OrderCost

END
GO
```

- f_calculate_total_order_cost(@OrderID) - Jak wyżej, uwzględnia zniżki

```
CREATE FUNCTION [dbo].[f_calculate_total_order_cost]
(
	@OrderId INT
)
RETURNS money
AS

BEGIN
	DECLARE @OrderTotalCost money
	DECLARE @Discount real
	SET @Discount =
[dbo].[f_calculate_individual_customer_discount_for_order] (@OrderID)
	SELECT @OrderTotalCost = SUM(D.UnitPrice*OD.Quantity * (1 -
@Discount)) FROM Orders AS O
	INNER JOIN OrderDetails AS OD ON O.OrderID = OD.OrderID
	INNER JOIN Dishes AS D ON D.DishID = OD.DishID

	RETURN @OrderTotalCost

END
GO
```

- f_calculate_discount_for_order(@OrderID) - oblicza zniżkę dla podanego zamówienia.

```
CREATE FUNCTION [dbo].[f_calculate_discount_for_order](@OrderID int)
RETURNS real
AS
BEGIN
    DECLARE @CustomerTotalDiscount real = 0
	DECLARE @CustomerID int = (SELECT CustomerID FROM Orders WHERE
OrderID = @OrderID)
	DECLARE @OrderDate datetime = (SELECT OrderDate FROM Orders WHERE
OrderID = @OrderID)
	DECLARE @DiscountID int = (SELECT DiscountID FROM
CustomerDiscounts WHERE DateGranted<=@OrderDate AND
(DateExpiring>=@OrderDate OR DateExpiring IS NULL))
	DECLARE @ConcernOrder bit = (SELECT ConcernOrder FROM Orders WHERE
```

```
OrderID=@OrderID)

	IF (@ConcernOrder=1)
	BEGIN
		SELECT @CustomerTotalDiscount=SUM([dbo].[f_min_two_reals]
(FR*CountDiscount, FM)) FROM CustomerDiscounts AS Customer
		INNER JOIN ConcernDiscounts AS Concern ON
Customer.DiscountID = Concern.DiscountID
		WHERE Customer.DiscountID = @DiscountID
		AND DateGranted<=@OrderDate AND (DateExpiring IS NULL OR
DateExpiring>=@OrderDate)
	END
	ELSE
	BEGIN
		SELECT @CustomerTotalDiscount=SUM(R*CountDiscount) FROM
CustomerDiscounts AS Customer
		INNER JOIN IndividualDiscounts AS Individual ON
Customer.DiscountID = Individual.DiscountID
		WHERE Customer.DiscountID = @DiscountID
		AND DateGranted<=@OrderDate AND (DateExpiring IS NULL OR
DateExpiring>=@OrderDate)
	END
	IF (@CustomerTotalDiscount IS NULL)
	BEGIN
	RETURN 0
	END


	RETURN @CustomerTotalDiscount

END
GO
```

Obliczające wydatki klienta

- f_calculate_total_amount_spent_by_individual_customer(@CustomerID) - Oblicza całkowitą wydaną kwotę przez klienta indywidualnego o zadanym ID.

```
CREATE FUNCTION
[dbo].[f_calculate_total_amount_spent_by_individual_customer]

(
	@CustomerID int
)
RETURNS money
AS
```

```
BEGIN
	DECLARE @result money

	SELECT @result=SUM(dbo.f_calculate_order_cost(O.OrderID))
	FROM Customers AS C
	INNER JOIN Orders AS O ON C.CustomerID=O.CustomerID
	WHERE C.CustomerID=@CustomerID AND O.ConcernOrder=0

	RETURN @result

END
GO
```

- f_calculate_total_amount_spent_by_concern_customer(@CustomerID) Jak wyżej, dla klienta firmowego.

```
CREATE FUNCTION
[dbo].[f_calculate_total_amount_spent_by_concern_customer]
(
	@CustomerID int
)
RETURNS money
AS
BEGIN
	DECLARE @result money

	SELECT @result=SUM(dbo.f_calculate_order_cost(O.OrderID))
	FROM Customers AS C
	INNER JOIN Orders AS O ON C.CustomerID=O.CustomerID
	WHERE C.CustomerID=@CustomerID AND O.ConcernOrder=1

	RETURN @result

END
GO
```

- f_calculate_amount_spent_by_individual_customer_month(@CustomerID,@year,@month) - Oblicza kwotę wydaną przez klienta indywidualnego o zadanym ID w podanym miesiącu.

```
CREATE FUNCTION
[dbo].[f_calculate_amount_spent_by_individual_customer_month]
(
	@CustomerID INT,
	@Year INT,
	@Month INT
```

```
)
RETURNS money
BEGIN

      DECLARE @result money

      SELECT @result=SUM(dbo.f_calculate_order_cost(O.OrderID))
      FROM Customers AS C
      INNER JOIN Orders AS O ON C.CustomerID=O.CustomerID
      WHERE (
            C.CustomerID=@CustomerID
            AND O.OrderFinishedDate IS NOT NULL
            AND O.ConcernOrder=0
            AND DATEPART(MONTH, O.OrderDate ) = @Month
            AND DATEPART(MONTH, O.OrderFinishedDate) = @Month
            AND DATEPART(YEAR, O.OrderDate) = @Year
            AND DATEPART(YEAR, O.OrderFinishedDate) = @Year
      )

      RETURN @result
END
GO
```

- f_calculate_amount_spent_by_concern_customer_month(@CustomerID,@year,@month) - Jak wyżej, dla klienta firmowego.

```
CREATE FUNCTION
[dbo].[f_calculate_amount_spent_by_concern_customer_month](@CustomerID
INT, @Year INT, @Month INT)
RETURNS money
BEGIN
      DECLARE @result money

      SELECT @result=SUM(dbo.f_calculate_order_cost(O.OrderID)) FROM
Customers AS C
      INNER JOIN Orders AS O ON C.CustomerID=O.CustomerID
      WHERE C.CustomerID=@CustomerID
      AND O.OrderFinishedDate IS NOT NULL
      AND O.ConcernOrder=1
      AND DATEPART(MONTH, O.OrderDate ) = @Month
      AND DATEPART(MONTH, O.OrderFinishedDate) = @Month
      AND DATEPART(YEAR, O.OrderDate) = @Year
      AND DATEPART(YEAR, O.OrderFinishedDate) = @Year

    RETURN @result
```

```
END
GO
```

- f_calculate_amount_spent_by_individual_customer_week(@CustomerID,@year,@week) - Oblicza kwotę wydaną przez klienta indywidualnego o zadanym ID w podanym tygodniu.

```
CREATE FUNCTION
[dbo].[f_calculate_amount_spent_by_individual_customer_week]
(
      @CustomerID INT,
      @Year INT,
      @Week INT
)
RETURNS money
BEGIN

      DECLARE @result money
      DECLARE @startOfTheWeek DATETIME

      SELECT @result=SUM(dbo.f_calculate_order_cost(O.OrderID))
      FROM Customers AS C
      INNER JOIN Orders AS O ON C.CustomerID=O.CustomerID
      WHERE  (
             C.CustomerID=@CustomerID
             AND O.OrderFinishedDate IS NOT NULL
             AND O.ConcernOrder=0
             AND DATEPART(week, O.OrderDate ) = @Week
             AND DATEPART(week, O.OrderFinishedDate) = @Week
             AND DATEPART(year, O.OrderDate) = @Year
             AND DATEPART(year, O.OrderFinishedDate) = @Year
     )

    RETURN @result
END
GO
```

- f_calculate_amount_spent_by_concern_customer_week(@CustomerID,@year,@week) - Jak wyżej, dla klienta firmowego

```
CREATE FUNCTION
[dbo].[f_calculate_amount_spent_by_concern_customer_week]
(
      @CustomerID INT,
      @Year INT,
      @Week INT
```

```
)
RETURNS money
BEGIN
	-- Declare the return variable here
	DECLARE @result money


	SELECT @result=SUM(dbo.f_calculate_order_cost(O.OrderID)) FROM
Customers AS C
	INNER JOIN Orders AS O ON C.CustomerID=O.CustomerID
	WHERE C.CustomerID=@CustomerID AND O.OrderFinishedDate IS NOT NULL
AND O.ConcernOrder=1 AND DATEPART(week, O.OrderDate ) = @Week AND
DATEPART(week, O.OrderFinishedDate) = @Week AND DATEPART(year,
O.OrderDate) = @Year AND DATEPART(year, O.OrderFinishedDate) = @Year


    RETURN @result
END

GO
```

Obliczające ilość zamówień klienta

- f_calculate_total_number_of_orders_by_individual_customer(@CustomerID) - Oblicza ile zamówień złożył klient indywidualny o zadanym ID

```
CREATE FUNCTION
[dbo].[f_calculate_total_number_of_orders_by_individual_customer]
(
	@CustomerID int
)
RETURNS int
AS
BEGIN
	DECLARE @result int

	SELECT @result=COUNT(*) FROM Customers AS C
	INNER JOIN Orders AS O ON C.CustomerID=O.CustomerID
	WHERE C.CustomerID=@CustomerID AND O.ConcernOrder=0

	RETURN @result

END
GO
```

- f_calculate_total_number_of_orders_by_concern_customer(@CustomerID) - Jak wyżej, dla klienta firmowego

```
CREATE FUNCTION
[dbo].[f_calculate_total_number_of_orders_by_concern_customer]
(
	@CustomerID int
)
RETURNS int
AS
BEGIN
	DECLARE @result int

	SELECT @result=COUNT(*) FROM Customers AS C
	INNER JOIN Orders AS O ON C.CustomerID=O.CustomerID
	WHERE C.CustomerID=@CustomerID AND O.ConcernOrder=1

	RETURN @result

END
GO
```

- f_get_amount_of_orders_by_individual_customers_costing_at_least(CustomerID , Price money)

```
CREATE FUNCTION
[dbo].[f_get_amount_of_orders_by_individual_customers_costing_at_least]
(
	@CustomerID int,
	@Price money
)
RETURNS int
AS
BEGIN
	-- Declare the return variable here
	DECLARE @result int

	SELECT @result=COUNT(*) FROM [dbo].[v_customers_orders_finished]
WHERE ([dbo].[f_calculate_order_cost] (OrderID))>=@Price AND
CustomerID=@CustomerID AND ConcernOrder=0

	-- Return the result of the function
	RETURN @result

END
GO
```

- f_calculate_number_of_orders_by_individual_customer_month(@CustomerI

D, @year, @month) - Oblicza ile zamówień złożył klient indywidualny o zadanym ID w danym miesiącu

```
CREATE FUNCTION
[dbo].[f_calculate_number_of_orders_by_individual_customer_month]
(
	@CustomerID int,
	@year int,
	@month int
)
RETURNS int
AS
BEGIN
	DECLARE @result int

	SELECT @result=COUNT(*) FROM Customers AS C
	INNER JOIN Orders AS O ON C.CustomerID=O.CustomerID
	WHERE (
		C.CustomerID=@CustomerID
		AND O.ConcernOrder=0
		AND YEAR(O.OrderDate)=@year
		AND MONTH(O.OrderDate)=@month
	)

	RETURN @result

END
GO
```

- f_calculate_number_of_orders_by_concern_customer_month(@CustomerID, @year, @month) - Jak wyżej, dla klienta firmowego

```
CREATE FUNCTION
[dbo].[f_calculate_amount_spent_by_concern_customer_month]
(
	@CustomerID INT,
	@Year INT,
	@Month INT
)
RETURNS money
BEGIN

	DECLARE @result money

	SELECT @result=SUM(dbo.f_calculate_order_cost(O.OrderID))
	FROM Customers AS C
	INNER JOIN Orders AS O ON C.CustomerID=O.CustomerID
```

```
      WHERE C.CustomerID=@CustomerID AND O.OrderFinishedDate IS NOT NULL
      AND O.ConcernOrder=1 AND DATEPART(MONTH, O.OrderDate ) = @Month
      AND DATEPART(MONTH, O.OrderFinishedDate) = @Month
      AND DATEPART(YEAR, O.OrderDate) = @Year
      AND DATEPART(YEAR, O.OrderFinishedDate) = @Year

    RETURN @result
END
GO
```

- f_calculate_number_of_orders_by_individual_customer_week(@CustomerID, @year, @week) - Oblicza ile zamówień złożył klient indywidualny o zadanym ID w danym tygodniu

```
CREATE FUNCTION
[dbo].[f_calculate_number_of_orders_by_individual_customer_week]
(
      @CustomerID int,
      @year int,
      @month int,
      @week int
)
RETURNS int
AS
BEGIN
      DECLARE @result int
      SELECT @result=COUNT(*) FROM Customers AS C
      INNER JOIN Orders AS O ON C.CustomerID=O.CustomerID
      WHERE (
            C.CustomerID=@CustomerID
            AND O.ConcernOrder=0
            AND YEAR(O.OrderDate)=@year
            AND MONTH(O.OrderDate)=@month
            AND DATEPART("ww", O.OrderDate)=@week
      )

      RETURN @result

END
GO
```

- f_calculate_number_of_orders_by_concern_customer_week(CustomerID, year, week) - Jak wyżej, dla klienta firmowego

```
CREATE FUNCTION
[dbo].[f_calculate_number_of_orders_by_concern_customer_week]
(
```

```
      @CustomerID int,
      @year int,
      @month int,
      @week int
)
RETURNS int
AS
BEGIN
      DECLARE @result int

      SELECT @result=COUNT(*) FROM Customers AS C
      INNER JOIN Orders AS O ON C.CustomerID=O.CustomerID
      WHERE (
            C.CustomerID=@CustomerID
            AND O.ConcernOrder=1
            AND YEAR(O.OrderDate)=@year
            AND MONTH(O.OrderDate)=@month
            AND DATEPART("ww", O.OrderDate)=@week
      )

      RETURN @result

END
GO
```

Pozostałe:

- f_min_two_reals (a,b) - oblicza minimum z dwóch wartości

```
CREATE FUNCTION [dbo].[f_min_two_reals]
(
      -- Add the parameters for the function here
      @a real,
      @b real
)
RETURNS int
AS
BEGIN

      RETURN CASE
            WHEN @a < @b THEN @a
            ELSE ISNULL(@b, @a)
        END

END

GO
```

- f_min_two_ints (a, b) - jak wyżej

```
CREATE FUNCTION [dbo].[f_min_two_ints]
(
	-- Add the parameters for the function here
	@a int,
	@b int
)
RETURNS int
AS
BEGIN

	RETURN CASE
		WHEN @a < @b THEN @a
		ELSE ISNULL(@b, @a)
	  END

END
GO
```

- f_get_amount_of_orders_by_individual_customers_costing_at_least (CustomerID, Price)

```
CREATE FUNCTION
[dbo].[f_get_amount_of_orders_by_individual_customers_costing_at_least]
(
	@CustomerID int,
	@Price money
)
RETURNS int
AS
BEGIN
	-- Declare the return variable here
	DECLARE @result int

	SELECT @result=COUNT(*) FROM [dbo].[v_customers_orders_finished]
WHERE ([dbo].[f_calculate_order_cost] (OrderID))>=@Price AND
CustomerID=@CustomerID AND ConcernOrder=0

	-- Return the result of the function
	RETURN @result

END
GO
```

- f_check_table_available_on_date (@TableID, @DateReserved) - zwraca 0 jeśli stolika nie można zarezerwować na podany dzień, 1 jeśli można

```
CREATE FUNCTION [dbo].[f_check_table_available_on_date]
(
	-- Add the parameters for the function here
	@TableID int,
	@DateReserved datetime
)
RETURNS bit
AS
BEGIN

	DECLARE @ConcernReservationsOnThisDate int = ISNULL((SELECT
COUNT(*) FROM ConcernTablesReservations WHERE DateReserved=@DateReserved
AND TableID=@TableID), 0)
	DECLARE @IndividualReservationsOnThisDate int = ISNULL((SELECT
COUNT(*) FROM IndividualReservations WHERE DateReserved=@DateReserved
AND TableID=@TableID), 0)
	IF ((@ConcernReservationsOnThisDate +
@IndividualReservationsOnThisDate) > 1) RETURN 0
	RETURN 1
END
GO
```

- f_calculate_count_discount_for_quarterly_concern_discounts (StartDate, CustomerID, Price) - oblicza ile kwartałów pod rząd (począwszy od StartDate) klient wykonał zamówienia za kwotę Price.

```
CREATE FUNCTION
[dbo].[f_calculate_count_discount_for_quarterly_concern_discounts]
(
	@StartDate datetime,
	@CustomerID int,
	@Price money
)
RETURNS int
AS
BEGIN
	DECLARE @CountDiscount int = 0

	DECLARE @OrderDates TABLE
	(
		OrderID int,
		OrderDate datetime
	)
```

```
	DECLARE @LastOrderInQuarterDate datetime = @StartDate
	DECLARE @OrderDatesLeft int = (SELECT COUNT(*) FROM @OrderDates)
	DECLARE @CurrentOrderDate datetime
	DECLARE @CurrentOrder int


	INSERT INTO @OrderDates (OrderDate)
	SELECT OrderDate FROM Orders WHERE CustomerID=@CustomerID AND
ConcernOrder=1


	IF (EXISTS (SELECT * FROM @OrderDates WHERE
MONTH(OrderDate)=MONTH(@StartDate)))
	BEGIN
		SELECT @CountDiscount=1
	END

	WHILE (@OrderDatesLeft > 0)
	BEGIN
		SELECT @CurrentOrder = (SELECT TOP 1 OrderID FROM
@OrderDates ORDER BY OrderDate DESC)
		SELECT @CurrentOrderDate = (SELECT TOP 1 OrderDate FROM
@OrderDates ORDER BY OrderDate DESC)
		IF (DATEDIFF(QUARTER, @CurrentOrderDate,
@LastOrderInQuarterDate)=1)
		BEGIN
			IF
([dbo].[f_calculate_amount_spent_by_concern_customer_quarter]
(@CustomerID, YEAR(@CurrentOrderDate), DATEPART(QUARTER,
@CurrentOrderDate)) > @Price)
			BEGIN
				SELECT @LastOrderInQuarterDate=@CurrentOrderDate
				SELECT @CountDiscount = @CountDiscount + 1
			END
		END
		ELSE IF (DATEDIFF(QUARTER, @CurrentOrderDate,
@LastOrderInQuarterDate)>1)
		BEGIN
			BREAK
		END

		DELETE @OrderDates WHERE OrderID=@CurrentOrder
		SELECT @OrderDatesLeft = @OrderDatesLeft - 1
	END

	RETURN @CountDiscount
```

```
END
GO
```

- f_calculate_count_discount_for_monthly_concern_discounts (StartDate, CustomerID, FK, FZ) - oblicza ile miesięcy pod rząd (począwszy od StartDate) klient wykonał przynajmniej FZ zamówień za łączną kwotę przynajmniej FK

```
CREATE FUNCTION
[dbo].[f_calculate_count_discount_for_monthly_concern_discounts]
(
	@StartDate datetime,
	@CustomerID int,
	@FK money,
	@FZ int
)
RETURNS int
AS
BEGIN
	DECLARE @CountDiscount int = 0

	DECLARE @OrderDates TABLE
	(
		OrderID int,
		OrderDate datetime
	)

	DECLARE @LastOrderInMonthDate datetime = @StartDate
	DECLARE @OrderDatesLeft int = (SELECT COUNT(*) FROM @OrderDates)
	DECLARE @CurrentOrderDate datetime
	DECLARE @CurrentOrder int


	INSERT INTO @OrderDates (OrderID, OrderDate)
	SELECT OrderID, OrderDate FROM Orders WHERE CustomerID=@CustomerID
AND ConcernOrder=1

	IF (EXISTS (SELECT * FROM @OrderDates WHERE
MONTH(OrderDate)=MONTH(@StartDate)))
	BEGIN
		SELECT @CountDiscount=1
	END

	WHILE (@OrderDatesLeft > 0)
	BEGIN
		SELECT @CurrentOrder = (SELECT TOP 1 OrderID FROM
```

```
@OrderDates ORDER BY OrderDate DESC)
            SELECT @CurrentOrderDate = (SELECT TOP 1 OrderDate FROM
@OrderDates ORDER BY OrderDate DESC)
            IF (DATEDIFF(MONTH, @CurrentOrderDate,
@LastOrderInMonthDate)=1)
            BEGIN
                  IF (

([dbo].[f_calculate_amount_spent_by_concern_customer_month]
(@CustomerID, YEAR(@CurrentOrderDate), MONTH(@CurrentOrderDate)))>=@FK
                        AND
[dbo].[f_calculate_number_of_orders_by_concern_customer_month]
(@CustomerID, YEAR(@CurrentOrderDate), MONTH(@CurrentOrderDate))>=@FZ
                        )
                  BEGIN
                        SELECT @LastOrderInMonthDate=@CurrentOrderDate
                        SELECT @CountDiscount = @CountDiscount + 1
                  END
            END
            ELSE IF (DATEDIFF(MONTH, @CurrentOrderDate,
@LastOrderInMonthDate)>1)
            BEGIN
                  BREAK
            END

            DELETE @OrderDates WHERE OrderID=@CurrentOrder
            SELECT @OrderDatesLeft = @OrderDatesLeft - 1
      END

      RETURN @CountDiscount

END
GO
```

- f_get_current_table_seats (@TableID) - zwraca ilość miejsc dostępnych przy stoliku według aktualnych restrykcji COVID.

```
CREATE FUNCTION [dbo].[f_get_current_table_seats]
(
      @TableID int
)
RETURNS int
AS
BEGIN
      DECLARE @Seats int
```

```
	SELECT @Seats=Seats
	FROM TableRestrictions
	WHERE StartDate>=GETDATE() AND EndDate<=GETDATE()

	RETURN @Seats
END
GO
```

7. Triggery

- t_dish_availability_in_restaurant_menu - Sprawdza dostępność dań w restauracjach po dodaniu nowego dania do menu

```
CREATE TRIGGER [dbo].[t_check_dish_availability_in_restaurant_menu]
ON [dbo].[Menu]
AFTER INSERT, UPDATE, DELETE
AS
DECLARE @DISH_ID_LIST_TO_REMOVE Vector2ID
DECLARE @RestaurantID INT = 0
DECLARE @IngredientID INT = 0
DECLARE @UnitsInStock INT = 0
DECLARE @QuantityRequired INT = 0
DECLARE @DishID INT = 0
WHILE (1=1)
BEGIN
	SELECT TOP 1 @RestaurantID = RestaurantID
	FROM Restaurants
	WHERE RestaurantID > @RestaurantID
	ORDER BY RestaurantID

	IF (@@ROWCOUNT = 0) BREAK;

	WHILE (1=1)
	BEGIN
		SELECT TOP 1 @DishID = DishID
		FROM Menu
		WHERE DishID > @DishID
		ORDER BY DishID

		IF (@@ROWCOUNT = 0) BREAK;


		WHILE (1=1)
		BEGIN
```

```
                  SELECT TOP 1 @IngredientID = IngredientID,
@QuantityRequired = QuantityRequired
                  FROM DishPreparationDetails
                  WHERE IngredientID > @IngredientID AND DishID =
@DishID
                  ORDER BY IngredientID

                  IF (@@ROWCOUNT = 0) BREAK;

                  IF (EXISTS (SELECT * FROM RestaurantIngredients WHERE
RestaurantID = @RestaurantID AND IngredientID = @IngredientID))
                  BEGIN
                        IF(EXISTS (SELECT * FROM RestaurantIngredients
WHERE IngredientID = @IngredientID AND @QuantityRequired > UnitsInStock
AND RestaurantID = @RestaurantID))
                        BEGIN
                              IF(NOT EXISTS (SELECT * FROM
@DISH_ID_LIST_TO_REMOVE WHERE FirstID = @RestaurantID AND SecondID =
@DishID))
                              BEGIN
                                    INSERT  @DISH_ID_LIST_TO_REMOVE
(FirstID,SecondID)
                                    VALUES (@RestaurantID, @DishID)
                              END
                        END
                  END
                  ELSE
                  BEGIN
                        IF(NOT EXISTS (SELECT * FROM
@DISH_ID_LIST_TO_REMOVE WHERE FirstID = @RestaurantID AND SecondID =
@DishID))
                        BEGIN
                              INSERT @DISH_ID_LIST_TO_REMOVE (FirstID,
SecondID)
                              VALUES (@RestaurantID, @DishID)
                        END
                  END
            END

            SET @IngredientID = 0
      END
      SET @DishID = 0

END
```

```
SET @RestaurantID = 0
SET @DishID = 0


WHILE ( 1 = 1)
BEGIN
      SELECT TOP 1 @RestaurantID = FirstID
      FROM @DISH_ID_LIST_TO_REMOVE
      WHERE FirstID > @RestaurantID
      ORDER BY FirstID

      IF (@@ROWCOUNT = 0) BREAK;

      WHILE (1 = 1)
      BEGIN
            SELECT TOP 1 @DishID = SecondID
            FROM @DISH_ID_LIST_TO_REMOVE
            WHERE SecondID > @DishID AND FirstID = @RestaurantID
            ORDER BY SecondID

            IF (@@ROWCOUNT = 0) BREAK;

            IF (NOT EXISTS (SELECT * FROM RestaurantMenuAvailability
WHERE RestaurantID = @RestaurantID AND DishID = @DishID))
            BEGIN
                  INSERT RestaurantMenuAvailability(RestaurantID,
DishID, Available)
                  VALUES (@RestaurantID, @DishID, 0)
            END
            ELSE
            BEGIN
                  UPDATE RestaurantMenuAvailability
                  SET Available = 0
                  WHERE RestaurantID = @RestaurantID AND DishID =
@DishID
            END
      END
      SET @DishID = 0
END

-- INSERTING AVAILABLE = 1
DECLARE @AllDishesID ID_List
SET @DishID = 0
SET @RestaurantID = 0
WHILE (1=1)
BEGIN
```

```
	SELECT TOP 1 @DishID = DishID
	FROM Menu
	WHERE DishID > @DishID
	ORDER BY DishID

	IF (@@ROWCOUNT = 0) BREAK;


	WHILE (1=1)
	BEGIN
		SELECT TOP 1 @RestaurantID = RestaurantID
		FROM Restaurants
		WHERE RestaurantID > @RestaurantID
		ORDER BY RestaurantID

		IF (@@ROWCOUNT = 0) BREAK;

		IF (NOT EXISTS (SELECT * FROM @DISH_ID_LIST_TO_REMOVE WHERE
FirstID = @RestaurantID AND SecondID = @DishID))
		BEGIN
			IF (NOT EXISTS (SELECT * FROM
RestaurantMenuAvailability WHERE RestaurantID = @RestaurantID AND DishID
= @DishID))
			BEGIN
				INSERT RestaurantMenuAvailability(RestaurantID,
DishID, Available)
				VALUES (@RestaurantID, @DishID, 1)
			END
			ELSE
			BEGIN
				EXEC p_change_dish_availability_in_restaurant
@RestaurantID, @DishID, 1
			END
		END
	END
END
GO

ALTER TABLE [dbo].[Menu] ENABLE TRIGGER
[t_check_dish_availability_in_restaurant_menu]
GO
```

- t_menu_dish_availability - sprawdza czy można dodać danie do menu

```
CREATE TRIGGER [dbo].[t_menu_dish_availability]
ON [dbo].[Menu]
AFTER UPDATE
AS
BEGIN
	SET NOCOUNT ON;

	DECLARE @DishID int, @StartDate DATETIME, @EndDate DATETIME
	SELECT @DishID=DishID, @StartDate = StartDate , @EndDate = EndDate
FROM deleted

	DECLARE @InsertedDishID int, @InsertedStartDate DATETIME,
@InsertedEndDate DATETIME
	SELECT @InsertedDishID=DishID, @InsertedStartDate = StartDate ,
@InsertedEndDate = EndDate FROM inserted

	IF (DATEDIFF(MONTH, @StartDate, @InsertedStartDate) < 1)
	BEGIN
		RAISERROR ('There must be at least a monthly difference
between dates in menu', 10,1)
		ROLLBACK
	END

END
GO

ALTER TABLE [dbo].[Menu] ENABLE TRIGGER [t_menu_dish_availability]
GO
```

- t_dish_availability_in_restaurant - sprawdza dostępność dania po zmienieniu wartość UnitIsStock w RestaurantIngredients

```
CREATE TRIGGER [dbo].[t_check_dish_availability_in_restaurant]
ON [dbo].[RestaurantIngredients]
AFTER INSERT, UPDATE, DELETE
AS
DECLARE @DISH_ID_LIST_TO_REMOVE Vector2ID
DECLARE @RestaurantID INT = 0
DECLARE @IngredientID INT = 0
DECLARE @UnitsInStock INT = 0
DECLARE @QuantityRequired INT = 0
DECLARE @DishID INT = 0
WHILE (1=1)
BEGIN
	SELECT TOP 1 @RestaurantID = RestaurantID
```

```
      FROM Restaurants
      WHERE RestaurantID > @RestaurantID
      ORDER BY RestaurantID

      IF (@@ROWCOUNT = 0) BREAK;

      WHILE (1=1)
      BEGIN
            SELECT TOP 1 @DishID = DishID
            FROM Menu
            WHERE DishID > @DishID
            ORDER BY DishID

            IF (@@ROWCOUNT = 0) BREAK;


            WHILE (1=1)
            BEGIN

                  SELECT TOP 1 @IngredientID = IngredientID,
@QuantityRequired = QuantityRequired
                  FROM DishPreparationDetails
                  WHERE IngredientID > @IngredientID AND DishID =
@DishID
                  ORDER BY IngredientID

                  IF (@@ROWCOUNT = 0) BREAK;

                  IF (EXISTS (SELECT * FROM RestaurantIngredients WHERE
RestaurantID = @RestaurantID AND IngredientID = @IngredientID))
                  BEGIN
                        IF(EXISTS (SELECT * FROM RestaurantIngredients
WHERE IngredientID = @IngredientID AND @QuantityRequired > UnitsInStock
AND RestaurantID = @RestaurantID))
                        BEGIN
                              IF(NOT EXISTS (SELECT * FROM
@DISH_ID_LIST_TO_REMOVE WHERE FirstID = @RestaurantID AND SecondID =
@DishID))
                              BEGIN
                                    INSERT  @DISH_ID_LIST_TO_REMOVE
(FirstID,SecondID)
                                    VALUES (@RestaurantID, @DishID)
                              END
                        END
                  END
                  ELSE
```

```
                  BEGIN
                        IF(NOT EXISTS (SELECT * FROM
@DISH_ID_LIST_TO_REMOVE WHERE FirstID = @RestaurantID AND SecondID =
@DishID))
                        BEGIN
                              INSERT @DISH_ID_LIST_TO_REMOVE (FirstID,
SecondID)
                              VALUES (@RestaurantID, @DishID)
                        END
                  END
            END

            SET @IngredientID = 0
      END
      SET @DishID = 0

END


SET @RestaurantID = 0
SET @DishID = 0


WHILE ( 1 = 1)
BEGIN
      SELECT TOP 1 @RestaurantID = FirstID
      FROM @DISH_ID_LIST_TO_REMOVE
      WHERE FirstID > @RestaurantID
      ORDER BY FirstID

      IF (@@ROWCOUNT = 0) BREAK;

      WHILE (1 = 1)
      BEGIN
            SELECT TOP 1 @DishID = SecondID
            FROM @DISH_ID_LIST_TO_REMOVE
            WHERE SecondID > @DishID AND FirstID = @RestaurantID
            ORDER BY SecondID

            IF (@@ROWCOUNT = 0) BREAK;

            IF (NOT EXISTS (SELECT * FROM RestaurantMenuAvailability
WHERE RestaurantID = @RestaurantID AND DishID = @DishID))
            BEGIN
                  INSERT RestaurantMenuAvailability(RestaurantID,
DishID, Available)
```

```
                  VALUES (@RestaurantID, @DishID, 0)
            END
            ELSE
            BEGIN
                  UPDATE RestaurantMenuAvailability
                  SET Available = 0
                  WHERE RestaurantID = @RestaurantID AND DishID =
@DishID
            END
      END
      SET @DishID = 0
END

-- INSERTING AVAILABLE = 1
DECLARE @AllDishesID ID_List
SET @DishID = 0
SET @RestaurantID = 0
WHILE (1=1)
BEGIN
      SELECT TOP 1 @DishID = DishID
      FROM Menu
      WHERE DishID > @DishID
      ORDER BY DishID

      IF (@@ROWCOUNT = 0) BREAK;


      WHILE (1=1)
      BEGIN
            SELECT TOP 1 @RestaurantID = RestaurantID
            FROM Restaurants
            WHERE RestaurantID > @RestaurantID
            ORDER BY RestaurantID

            IF (@@ROWCOUNT = 0) BREAK;

            IF (NOT EXISTS (SELECT * FROM @DISH_ID_LIST_TO_REMOVE WHERE
FirstID = @RestaurantID AND SecondID = @DishID))
            BEGIN
                  IF (NOT EXISTS (SELECT * FROM
RestaurantMenuAvailability WHERE RestaurantID = @RestaurantID AND DishID
= @DishID))
                  BEGIN
                        INSERT RestaurantMenuAvailability(RestaurantID,
DishID, Available)
                        VALUES (@RestaurantID, @DishID, 1)
```

```
                END
                ELSE
                BEGIN
                      EXEC p_change_dish_availability_in_restaurant
@RestaurantID, @DishID, 1
                END
            END
      END
END
GO

ALTER TABLE [dbo].[RestaurantIngredients] ENABLE TRIGGER
[t_check_dish_availability_in_restaurant]
GO
```

- t_order_insert - Sprawdza możliwość złożenia zamówienia.

```
CREATE TRIGGER [dbo].[t_order_insert]
   ON [dbo].[Orders]
   AFTER INSERT
AS
BEGIN
      -- SET NOCOUNT ON added to prevent extra result sets from
      -- interfering with SELECT statements.
      SET NOCOUNT ON;
      DECLARE @CustomerID int, @OrderDate datetime, @OrderID int,
@RestaurantID int, @EmployeeID int, @ConcernOrder bit, @DishList
DishID_Quantity_List
      SELECT @CustomerID=CustomerID, @OrderDate=OrderDate,
@OrderID=OrderID, @RestaurantID=RestaurantID , @EmployeeID = EmployeeID,
@ConcernOrder=ConcernOrder FROM inserted


      -- Employee not fired
      IF ((SELECT DateFired FROM Employees WHERE EmployeeID =
@EmployeeID) IS NOT NULL)
      BEGIN
            RAISERROR ('Employee is fired', 1, 69691)
            ROLLBACK
      END

      -- Concern order by not ConcernCustomer
      IF (@ConcernOrder=1 AND NOT EXISTS (SELECT * FROM ConcernCustomers
WHERE CustomerID=@CustomerID))
```

```
      BEGIN
            RAISERROR ('Can''t add concern order, customer is not
concern customer!', 1, 69691)
            ROLLBACK
      END


      -- Individual order by not IndividualCustomer
      IF (@ConcernOrder=0 AND NOT EXISTS (SELECT * FROM
IndividualCustomers WHERE CustomerID=@CustomerID))
      BEGIN
            RAISERROR ('Can''t add individual order, customer is not
individual customer!', 1, 69691)
            ROLLBACK
      END

      --give discounts to customer
      EXEC [dbo].[p_give_discounts_to_customer] @CustomerID, @OrderDate

END
GO

ALTER TABLE [dbo].[Orders] ENABLE TRIGGER [t_order_insert]
GO
```

- t_concern_reservation_insertion - Sprawdza czy dodana rezerwacja firmowa jest poprawna: czy zarezerwowany stolik jest wolny oraz czy nie przydzielono zbyt dużo gości do jednego stolika.

```
CREATE TRIGGER [dbo].[t_concern_reservation_insertion]
   ON [dbo].[ConcernTablesReservations]
   AFTER INSERT
AS
BEGIN
      -- SET NOCOUNT ON added to prevent extra result sets from
      -- interfering with SELECT statements.
      SET NOCOUNT ON;

    -- Insert statements for trigger here

      DECLARE @DateReserved datetime = (SELECT DateReserved FROM
inserted)
      DECLARE @TableID int = (SELECT TableID FROM inserted)
      DECLARE @Seats int = [dbo].[f_get_current_table_seats] (@TableID,
```

```
@DateReserved)
	DECLARE @Guests int = (SELECT Guests FROM inserted)
	IF ([dbo].[f_check_table_available_on_date] (@TableID,
@DateReserved) = 0)
	BEGIN
		RAISERROR ('Table already reserved for this date!', 1,
69691)
		ROLLBACK
	END

	IF (@Guests>@Seats)
	BEGIN
		RAISERROR ('This table is too small!', 1, 69692)
		ROLLBACK
	END
END
GO

ALTER TABLE [dbo].[ConcernTablesReservations] ENABLE TRIGGER
[t_concern_reservation_insertion]
GO
```

- t_individual_reservation_insertion - Sprawdza czy dodana rezerwacja jest poprawna: czy zarezerwowany stolik jest wolny, czy nie przydzielono zbyt dużo gości do jednego stolika, oraz czy klient jest uprawniony do składania rezerwacji.

```
CREATE TRIGGER [dbo].[t_individual_reservation_insertion]
   ON  [dbo].[IndividualReservations]
   AFTER INSERT
AS
BEGIN
	-- SET NOCOUNT ON added to prevent extra result sets from
	-- interfering with SELECT statements.
	SET NOCOUNT ON;

    -- Insert statements for trigger here

	DECLARE @DateReserved datetime = (SELECT DateReserved FROM
inserted)
	DECLARE @TableID int = (SELECT TableID FROM inserted)
	DECLARE @Seats int = [dbo].[f_get_current_table_seats] (@TableID,
@DateReserved)
	DECLARE @CustomerID int = (SELECT CustomerID FROM inserted)
	DECLARE @Guests int = (SELECT Guests FROM inserted)
```

```
	DECLARE @OrderID int = (SELECT OrderID FROM inserted)
	DECLARE @OrderCost money = [dbo].[f_calculate_order_cost]
(@OrderID)
	DECLARE @Orders int =
[dbo].[f_calculate_total_number_of_orders_by_individual_customer]
(@CustomerID)

	DECLARE @BadReservation bit = 0


	IF ([dbo].[f_check_table_available_on_date] (@TableID,
@DateReserved) = 0)
	BEGIN
		RAISERROR ('Table already reserved for this date!', 1,
69691)
		SELECT @BadReservation=1
	END

	IF (@Guests>@Seats)
	BEGIN
		RAISERROR ('This table is too small!', 1, 69692)
		SELECT @BadReservation=1
	END

	IF ((@Orders<5 AND @OrderCost<200) OR (@Orders>=5 AND
@OrderCost<50))
	BEGIN
		RAISERROR ('Customer not allowed to place reservation!', 1,
69693)
		SELECT @BadReservation=1
	END

	IF(@BadReservation=1)
	BEGIN
		ROLLBACK
		DELETE OrderDetails WHERE OrderID=@OrderID
		DELETE Orders WHERE OrderID=@OrderID
	END
END
GO

ALTER TABLE [dbo].[IndividualReservations] ENABLE TRIGGER
[t_individual_reservation_insertion]
GO
```

- t_check_dish_availability - Sprawdza możliwość utworzenia zamówienia z danym daniem.

```
CREATE TRIGGER [dbo].[t_check_dish_availability]
ON [dbo].[OrderDetails]
AFTER INSERT

AS
BEGIN
	DECLARE @DishID INT
	DECLARE @OrderID INT
	DECLARE @Quantity INT

	SELECT @DishID= DishID , @OrderID= OrderID, @Quantity= Quantity
FROM inserted

	DECLARE @RestaurantID INT

	SELECT @RestaurantID = RestaurantID FROM Orders WHERE OrderID =
@OrderID

	DECLARE @IngredientID INT = 0
	DECLARE @QuantityRequired INT = 0

	WHILE (1=1)
	BEGIN
		SELECT TOP 1 @IngredientID = IngredientID, @QuantityRequired
= QuantityRequired*@Quantity
		FROM DishPreparationDetails
		WHERE IngredientID > @IngredientID AND DishID = @DishID
		ORDER BY IngredientID

		IF (@@ROWCOUNT = 0) BREAK;

		IF (NOT EXISTS (SELECT * FROM RestaurantIngredients WHERE
RestaurantID = @RestaurantID AND IngredientID = @IngredientID AND
UnitsInStock >= @QuantityRequired))
		BEGIN
			RAISERROR ('Dish is not available in this restaurant',
1, 69691)
			ROLLBACK
		END

	END
```

```
END
GO

ALTER TABLE [dbo].[OrderDetails] ENABLE TRIGGER
[t_check_dish_availability]
GO
```

8. Zdefiniowane typy

```
CREATE TYPE [dbo].[DishID_Quantity_List] AS TABLE(
      [DishID] [int] NOT NULL,
      [Quantity] [int] NOT NULL,
      PRIMARY KEY CLUSTERED
(
      [DishID] ASC
)WITH (IGNORE_DUP_KEY = OFF)
)
GO
```

```
CREATE TYPE [dbo].[Guest_List] AS TABLE(
      [GuestNumber] [int] IDENTITY(1,1) NOT NULL,
      [GuestName] [nvarchar](50) NOT NULL,
      PRIMARY KEY CLUSTERED
(
      [GuestNumber] ASC
)WITH (IGNORE_DUP_KEY = OFF)
)
GO
```

```
CREATE TYPE [dbo].[ID_List] AS TABLE(
      [DishID] [int] NOT NULL,
      PRIMARY KEY CLUSTERED
(
      [DishID] ASC
)WITH (IGNORE_DUP_KEY = OFF)
)
GO
```

```
CREATE TYPE [dbo].[IngredientID_QuantityRequired_List] AS TABLE(
      [IngredientID] [int] NOT NULL,
      [QuantityRequired] [real] NOT NULL,
```

```
      PRIMARY KEY CLUSTERED
(
      [IngredientID] ASC
)WITH (IGNORE_DUP_KEY = OFF)
)
GO
```

```
CREATE TYPE [dbo].[Vector2ID] AS TABLE(
      [FirstID] [int] NOT NULL,
      [SecondID] [int] NOT NULL
)
GO
```

```
CREATE TYPE [dbo].[DishID_StartDate_EndDate_List] AS TABLE(
      [DishID] [int] NOT NULL,
      [StartDate] [datetime] NOT NULL,
      [EndDate] [datetime] NOT NULL,
      PRIMARY KEY CLUSTERED
(
      [DishID] ASC
)WITH (IGNORE_DUP_KEY = OFF)
)
GO
```